SIAM J. COMPUT.
Vol. 53, No. 6, pp. FOCS20-368–FOCS20-415

SYMMETRIES, GRAPH PROPERTIES, AND QUANTUM SPEEDUPS*

SHALEV BEN-DAVID†, ANDREW M. CHILDS‡, ANDRÁS GILYÉN§, WILLIAM KRETSCHMER¶, SUPARTHA PODDER‖, AND DAOCHEN WANG#

Abstract. Aaronson and Ambainis [*Theory Comput.*, 10 (2014), pp. 133–166] and Chailloux [*Proceedings of the* 10*th Innovations in Theoretical Computer Science Conference*, 2018, pp. 19:1–19:7] showed that fully symmetric (partial) functions do not admit exponential quantum query speedups. This raises a natural question: how symmetric must a function be before it cannot exhibit a large quantum speedup? In this work, we prove that hypergraph symmetries in the adjacency matrix model allow at most a polynomial separation between randomized and quantum query complexities. We also show that, remarkably, permutation groups constructed out of these symmetries are essentially the *only* permutation groups that prevent superpolynomial quantum speedups. We prove this by fully characterizing the primitive permutation groups that allow superpolynomial quantum speedups. In contrast, in the adjacency list model for bounded-degree graphs—where graph symmetry is manifested differently—we exhibit a property testing problem that shows an exponential quantum speedup. These results resolve open questions posed by Ambainis, Childs, and Liu [*Lecture Notes in Comput. Sci.* 6845, Springer, 2011, pp. 365–376] and Montanaro and de Wolf [*Theory Comput.*, 7 (2016)].

Key word. quantum query complexity

*Received by the editors June 6, 2023; accepted for publication (in revised form) September 10, 2024; published electronically December 13, 2024. This manuscript is our original work. A 12-page extended abstract appeared in *Proceedings of the* 61*st Symposium on Foundations of Computer Science (FOCS* 2020*)*, and the work was presented at the 24th *Annual Conference on Quantum Information Processing (QIP* 2021*)*, which does not have proceedings. Except that, no part of this work has been published or is in consideration for publication elsewhere. A preprint of the full version appears at https://arxiv.org/abs/2006.12760.

https://doi.org/10.1137/23M1573975

Funding: The second and sixth authors acknowledge support from the Army Research Office (grant W911NF-20-1-0015); the Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Quantum Algorithms Teams and Accelerated Research in Quantum Computing programs; and the National Science Foundation (grant CCF-1813814). The third author acknowledges funding provided by Samsung Electronics Co., Ltd., for the project "The Computational Power of Sampling on Quantum Computers." Additional support was provided by the Institute for Quantum Information and Matter, an NSF Physics Frontiers Center (NSF grant PHY-1733907), and the EU's Horizon 2020 Marie Skłodowska-Curie program QuantOrder-891889. The first author is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), DGECR-2019-00027 and RGPIN-2019-04804. (Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), DGECR-2019-00027 et RGPIN-2019-04804.) The fifth author is supported by the US Department of Energy (grant DE-SC0023179) and by the US National Science Foundation (NSF grant 1954311). The fourth author acknowledges support from a Vannevar Bush Fellowship from the US Department of Defense and also from the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Quantum Systems Accelerator.

†Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada (shalev.b@uwaterloo.ca).

‡Department of Computer Science, Institute for Advanced Computer Studies, and Joint Center forr Quantum Information and Computer Science, University of Maryland, College Park, MD 20742 USA (amchilds@umd.edu).

§Alfréd Rényi Institute of Mathematics, Budapest, Hungary (gilyen@renyi.hu).

¶Simons Institute for the Theory of Computing, University of California, Berkeley, Berkeley, CA 94720 USA (kretsch@berkeley.edu).

‖Department of Computer Science, Stony Brook University, Stony Brook, NY 11794 USA (supartha@cs.stonybrook.edu).

#Department of Computer Science, University of British Columbia, Vancouver, BC, Canada (wdaochen@gmail.com).

MSC codes. 03D15, 11Y16, 68Q05, 68R10, 68W20, 68W40, 81P68

DOI. 10.1137/23M1573975

1. Introduction. One of the most fundamental problems in the field of quantum computing is the question of when quantum algorithms substantially outperform classical ones. While polynomial quantum speedups are known in many settings, superpolynomial quantum speedups are known (or even merely conjectured) for only a few select problems. Crucially, exponential quantum speedups only occur for certain "structured" problems such as period-finding (used in Shor's factoring algorithm [51]) and Simon's problem [52], in which the input is known in advance to have a highly restricted form. In contrast, for "unstructured" problems such as black-box search or NP-complete problems, only polynomial speedups are known (and in some models, it can be formally shown that only polynomial speedups are possible).

In this work, we are interested in formalizing and characterizing the structure necessary for fast quantum algorithms. In particular, we study the *types of symmetries* a function can have while still exhibiting superpolynomial quantum speedups.

1.1. Prior work. Despite the strong intuition in the field that structure is necessary for superpolynomial quantum speedups, only a handful of works have attempted to formalize this and characterize the required structure. All of them study the problem in the query complexity (black-box) model of quantum computation, which is a natural framework in which both period-finding and Simon's problem can be formally shown to give exponential quantum speedups (see [20] for a survey of query complexity or [27] for a formalization of period-finding specifically).

In the query complexity model, the goal is to compute a Boolean function $f : \Sigma^n \to \{0,1\}$ using as few queries to locations x_i of the input $x \in \Sigma^n$ as possible, where Σ is some finite alphabet. For classical query algorithms, each query specifies an index $i \in [n] := \{1,2,\ldots,n\}$ and receives the response $x_i \in \Sigma$. Quantum query algorithms are allowed to make queries in superposition, and we are interested in how much advantage this gives them over randomized classical algorithms. A query algorithm must output $f(x)$ with bounded-error probability for all $x \in \Sigma^n$ using as few queries to x as possible. For formal definitions of these notions, see [20].

Beals et al. [13] showed that all *total Boolean functions* $f : \Sigma^n \to \{0,1\}$ have a polynomial relationship between their classical and quantum query complexities (which we denote $\mathrm{R}(f)$ and $\mathrm{Q}(f)$, respectively). This means that superpolynomial speedups are not possible in query complexity unless we impose a promise on the input: that is, unless we define $f : P \to \{0,1\}$ with $P \subsetneq \Sigma^n$ and allow an algorithm computing f to behave arbitrarily on inputs outside of the promise set P. For such promise problems (also called partial functions), provable exponential quantum speedups are known. This is the setting in which Simon's problem and period-finding reside.

The question, then, is what we can say about the structure necessary for a partial Boolean function f to exhibit a superpolynomial quantum speedup. Toward this end, Aaronson and Ambainis [1] showed that *symmetric* functions do not allow superpolynomial quantum speedups, even with a promise. Chailloux [23] improved this result by reducing the degree of the polynomial relationship between randomized and quantum algorithms for symmetric functions and removing a technical requirement on the symmetry of those functions.[1]

[1] Aaronson and Ambainis required the function to be symmetric both under permuting the n bits of the input and under permuting the alphabet symbols in Σ; Chailloux showed that the latter is not necessary.

Other work attempted to characterize the structure necessary for quantum speedups in alternative ways. Ben-David [16] showed that certain types of symmetric promises do not admit any function with a superpolynomial quantum speedup, a generalization of [13] (which showed this when the promise set is Σ^n). Aaronson and Ben-David [3] showed that small promise sets, which contain only $\mathrm{poly}(n)$ inputs out of $|\Sigma|^n$, also do not admit functions separating quantum and classical algorithms by more than a polynomial factor.

A class of problems with significant symmetry, though much less than full permutation symmetry, is the class of graph properties. For such problems, the input describes a graph, and the output depends only on the isomorphism class of that graph. Thus the vertices can be permuted arbitrarily, but such a permutation induces a structured permutation on the edges, about which queries provide information. The quantum query complexity of graph properties has been extensively studied in both adjacency matrix and adjacency list models (see, for example, [21, 18, 54, 60, 29, 9, 48, 8, 14, 26, 61, 45, 44, 10]). Most of these works focus on the query complexity of total functions, so the quantum speedups are at most polynomial.

The setting of graph property *testing* provides a natural class of partial graph properties. Here we are promised that the input graph either has a property or is ϵ-far from having the property, meaning that we must change at least an ϵ fraction of the edges to make the property hold. Graph property testing has been extensively studied since its introduction by Goldreich, Goldwasser, and Ron [32].

Quantum algorithms for testing properties of bounded-degree graphs in the adjacency list model were studied by Ambainis, Childs, and Liu [7]. They gave upper bounds of $\tilde{O}(n^{1/3})$ for testing bipartiteness and expansion, demonstrating polynomial quantum speedups. Furthermore, they showed that at least $\Omega(n^{1/4})$ quantum queries are required to test expansion, ruling out the possibility of an exponential quantum speedup. This work naturally raises the question (also highlighted by Montanaro and de Wolf [49]) of whether there can ever be exponential quantum speedup for graph property testing or for graph properties more generally.

1.2. Our contributions. In this work, we extend the results of Aaronson and Ambainis and of Chailloux to other symmetry groups. We characterize general symmetries of functions as follows.

DEFINITION 1.1. *Let $f : P \to \{0,1\}$ be a function with $P \subseteq \Sigma^n$, where Σ is a finite alphabet and $n \in \mathbb{N}$. We say that f is* symmetric *with respect to a permutation group G acting on domain $[n]$ if for all $x \in P$ and all $\pi \in G$, the string $x \circ \pi$ defined by $(x \circ \pi)_i := x_{\pi(i)}$ satisfies $x \circ \pi \in P$ and $f(x \circ \pi) = f(x)$.*

The case where $G = S_n$, the fully symmetric permutation group, is the scenario handled in Chailloux's work [23]: he showed that $\mathrm{R}(f) = O(\mathrm{Q}(f)^3)$ if f is symmetric under S_n. Aaronson and Ambainis [1] required an even stronger symmetry property. We note that when Σ is large, say, $|\Sigma| = n$ or larger, the class of functions symmetric under S_n is already highly nontrivial: among others, it includes functions such as COLLISION, an important function whose quantum query complexity was established in [5]; k-SUM, whose quantum query complexity required the negative-weight adversary to establish [15]; and k-DISTINCTNESS, whose quantum query complexity is still open [22]. Additionally, computational geometry functions such as CLOSESTPAIR (a function studied in recent work by Aaronson et al. [4]) are typically symmetric under S_n, as the points are usually represented as alphabet symbols. If we round the

alphabet to be finite, the $G = S_n$ case already shows that no computational geometry problem of this form can have supercubic quantum speedups.

In this work, we examine what happens when we relax the full symmetry S_n to smaller symmetry groups G. We introduce some tools for showing that particular classes of permutation groups G do not allow superpolynomial quantum speedups; that is, we provide tools for showing that every f symmetric with respect to G satisfies $\mathrm{Q}(f) = \mathrm{R}(f)^{\Omega(1)}$. Our first main result is the following theorem, in which G is the *graph symmetry*: the permutation group acting on strings of length $\binom{n}{2}$ that includes all permutations of the string positions, labeled by edges of an n-vertex graph, which are induced by one of the $n!$ relabelings of the n vertices. Functions that take the adjacency matrix of a graph as input, and whose output depends only on the isomorphism class of the graph (not on the labeling of its vertices), are symmetric with respect to the graph symmetry G.

THEOREM 1.2 (informal version of Corollary 4.14). *Any Boolean function f that takes the adjacency matrix of a graph as input and is symmetric with respect to renaming the vertices of the graph has $\mathrm{R}(f) = O(\mathrm{Q}(f)^6)$. This holds even if f is a partial function.*

This theorem holds even when the alphabet of f is non-Boolean. We note that this is a strict generalization of the result of Aaronson and Ambainis [1], since any fully symmetric function is necessarily symmetric under the graph symmetry as well. It is also a generalization of [23], except that our polynomial degree (power 6) is larger than the power 3 of Chailloux. Our results also extend to other types of graph symmetries, including hypergraph symmetries and bipartite graph symmetries.

Next, we extend Theorem 1.2 to give a more general characterization of which classes of symmetries are inconsistent with superpolynomial quantum speedups. Our main result in this regard is a dichotomy theorem for *primitive* permutation groups. Primitive permutation groups are sometimes described as the "building blocks" of permutation groups, because arbitrary permutation groups can always be decomposed into primitive groups (in a certain formal sense). We give a complete classification of which primitive groups G allow superpolynomial quantum speedups and which do not, in terms of the *minimal base size* $b(G)$, which is a key quantity in computational group theory. For a permutation group acting on n points, the minimal base size ranges between 0 and n, and it roughly captures the size of G relative the full symmetric group S_n. Thus, the following theorem amounts to showing that symmetries of "large" primitive permutation groups do not allow large quantum speedups, while symmetries of "small" primitive permutation groups do.

THEOREM 1.3 (informal version of Corollary 5.6). *Let G be a primitive permutation group acting on $[n]$, and let $b(G)$ denote the size of a minimal base for G. If $b(G) = n^{\Omega(1)}$, then $R(f) = Q(f)^{O(1)}$ for every f that is symmetric under G. Otherwise, if $b(G) = n^{o(1)}$, then there exists a partial function f that is symmetric under G such that $R(f) = Q(f)^{\omega(1)}$.*

With some additional work that involves decomposing an arbitrary permutation group into primitive groups, we also extend the above theorem to show that if a permutation group G does not allow superpolynomial speedups, then it must be constructed out of a constant number of primitive groups H that satisfy $b(H) = n^{\Omega(1)}$, where H acts on n points. Put another way, if any of the primitive factors H of G satisfy $b(H) = n^{o(1)}$, or if G contains more than a constant number of primitive

factors, then we exhibit a function that is symmetric under G with a superpolynomial quantum speedup.

Remarkably, the proof of Theorem 1.3 also includes a strong characterization of what the primitive permutation groups G that satisfy $b(G) = n^{\Omega(1)}$ look like. Indeed, we show that as a consequence of the classification of finite simple groups, all such groups essentially look like hypergraph symmetries or minor extensions thereof. Thus, in some sense, permutation groups built out of hypergraph symmetries are the *only* permutation groups that are inconsistent with superpolynomial speedups. We consider this one of the most surprising consequences of our work; a priori, one could expect there to be many different kinds of symmetries that disallow superpolynomial speedups.

Finally, we return to the topic of graph properties. While the aforementioned results show that no exponential speedup is possible for graph properties in the adjacency *matrix* model, they do not resolve the original open question of Ambainis, Childs, and Liu [7], which specifically addressed graph property testing in the adjacency *list* model.

In the adjacency list model, a graph with n vertices and bounded degree d is specified by some[2] $x : [n] \times [d] \to [n] \cup \{*\}$ such that $x(v, i)$ is the ith neighbor of vertex v if v has at least i neighbors and is $*$ otherwise. Graph symmetries in this model manifest themselves in a different way that is not captured by Definition 1.1, so Theorem 1.2 does not apply. To elaborate, the isomorphism class of x essentially consists of graphs of the form $\pi^{-1} \circ x \circ (\pi \times \mathrm{id}_{[d]})$, where $\pi \in S_n$ is a relabeling of vertices and π^{-1} is defined as the inverse of π but also maps $*$ to $*$. A function f is a graph property in this model if and only if its domain P is a union of such isomorphism classes and f is constant on each isomorphism class.

In the adjacency list model, we show the following.

THEOREM 1.4 (informal version of Theorem 6.1). *There exists a graph property testing problem in the adjacency list model for which there is an exponential quantum speedup.*

This is in stark contrast to the situation in the adjacency matrix case. Together, Theorems 1.2 and 1.4 fully settle the open question about the possibility of exponential quantum speedup for graph properties, showing that its answer is highly model-dependent: exponential speedup is impossible in the adjacency matrix model but is possible in the adjacency list model, even in the restrictive setting of graph property *testing*.

1.3. Techniques. Here we briefly discuss the techniques used to prove the main results.

1.3.1. Well-shuffling symmetries do not allow speedups. Our analysis begins with a basic observation of Chailloux [23]. Suppose that f is symmetric under the full symmetric group S_n acting on $[n]$. Zhandry [59] showed that distinguishing a random element of S_n from a random small-range function $\alpha : [n] \to [n]$ with $|\alpha([n])| = r$ requires $\Omega(r^{1/3})$ quantum queries.[3] Now, if Q is a quantum algorithm solving f using T queries, then Q also outputs $f(x)$ on input $x \circ \pi$ (the input x with bits shuffled according to π) for any $\pi \in S_n$, since f is symmetric under S_n. In particular, $Q(x \circ \pi)$ for a random $\pi \in S_n$ still outputs $f(x)$. However, the T-query

[2]The specification is not unique due to different possible orderings of the neighbors of each vertex.

[3]Actually, we show that a version of Zhandry's result that is sufficient for our purposes follows easily from the collision lower bound, so his techniques are not necessary for our results.

algorithm Q cannot distinguish a random $\pi \in S_n$ from a random function $\alpha : [n] \to [n]$ with range $r \approx T^3$. Therefore Q must output $f(x)$ with at most constant error even when run on $x \circ \alpha$ for a random small-range function α. This property can be used to simulate Q classically: a classical algorithm R can simply sample a small-range function α, explicitly query the entire string $x \circ \alpha$ (using only $O(T^3)$ queries since α has range $O(T^3)$), and then simulate Q on the string $x \circ \alpha$. This is an $O(T^3)$-query classical algorithm for computing f, created from a T-query quantum algorithm for f.

The above trick can be generalized from the fully symmetric group S_n to other permutation groups G, provided we can show that it is hard for a T-query quantum algorithm to distinguish G from a function with range $\text{poly}(T)$. The question of whether there exists an arbitrary symmetric function f with a quantum speedup is therefore reduced to the question of whether the concrete task of distinguishing G from the set of all small-range functions can be done quickly using a quantum algorithm. That is, if $D_{n,r}$ is the set of all strings in $[n]^n$ that use only r unique symbols, then we care about the quantum query cost of distinguishing $D_{n,r}$ from G; if this cost is $r^{\Omega(1)}$, then no function that is symmetric under G can exhibit a superpolynomial quantum speedup. In this case, we call G *well-shuffling*.

We show that the well-shuffling property is preserved under various operations one might perform on a permutation group, which allows us to prove that many permutation groups are well-shuffling simply by reduction to S_n. As an example, for undirected graph properties, the relevant transformation is from a permutation group G acting on a set $[n]$ to the induced action H on the $\binom{n}{2}$ unordered pairs of $[n]$. We then show that if G is well-shuffling, then H is also well-shuffling, because the problem of distinguishing G from a small-range function has a reduction to the problem of distinguishing H from a small-range function. More generally, this reduction works for ordered pairs or for tuples of any constant length.

1.3.2. Classification of symmetries with and without speedups. To characterize which permutation groups allow superpolynomial quantum speedups, we first show that large quantum speedups exist for any sufficiently small permutation group. In particular, if the minimal base size $b(G)$ of a permutation group G acting on n points satisfies $b(G) = n^{o(1)}$, then we exhibit a partial function that is symmetric under G with a superpolynomial quantum speedup. We construct such a function by starting with an arbitrary promise problem that has a sufficiently large separation between randomized and quantum query complexities (e.g., Simon's problem [52] or Forrelation [2]). We then modify it to be symmetric under G in a way that preserves the superpolynomial separation between randomized and quantum query complexities.

One might wonder whether a sort of converse holds, i.e., whether $b(G) = n^{\Omega(1)}$ implies that G is well-shuffling, and therefore that all partial functions symmetric under G admit at most a polynomial quantum speedup. We show this for primitive permutation groups G, essentially because the structure of large primitive groups is very well-understood. We base our proof on a theorem due to Liebeck [46] that characterizes minimal base sizes of primitive groups via the classification of finite simple groups. Liebeck's theorem allows us to show that primitive groups G with $b(G) = n^{\Omega(1)}$ can all be constructed out of the symmetric group via transformations that preserve the well-shuffling property. As a result, the quantity $b(G)$ completely characterizes whether a primitive permutation group admits a superpolynomial speedup. Moreover, the transformations involved are the same transformations we use to show that hypergraph symmetries are well-shuffling, so primitive groups that satisfy $b(G) = n^{\Omega(1)}$ all essentially look like hypergraph symmetries.

To go beyond primitive groups, we use the fact that an arbitrary permutation group can be decomposed into transitive groups (by looking at its orbits), which can then be decomposed into primitive groups (by looking at its nontrivial block systems). We show that for a group G, if any of the primitive factors in such a decomposition of G are consistent with superpolynomial speedups, if the number of primitive factors is $\omega(1)$, then there exists a partial function that is symmetric under G with a superpolynomial quantum speedup. These functions are generally easy to construct, though in one case that involves tree-like symmetries we make essential use of the 1-Fault Direct Trees problem of Zhan, Kimmel, and Hassidim [57]: a partial function that has a superpolynomial quantum speedup, constructed by adding a promise to a Boolean evaluation tree in a way that preserves the symmetries of the tree. Interestingly, these results also show that the quantity $b(G)$ does *not* determine the well-shuffling property in general, because there exist permutation groups G acting on n points with $b(G) = \Omega(n)$ that are consistent with exponential quantum speedups. Nevertheless, we can at least say that if a permutation group G is inconsistent with superpolynomial quantum speedups, then it must be decomposable into $O(1)$ well-shuffling primitive groups.

1.3.3. Graph property testing in the adjacency list model. Finally, we show that the possibility of exponential quantum speedups for graph properties depends strongly on the input model: while the adjacency matrix model allows at most polynomial speedups, we give an example of an exponential speedup in the adjacency list model for bounded-degree graphs. In particular, we show that such a separation holds even in the more restrictive setting of property testing, where "no" instances are promised to be far from "yes" instances.

The main idea for this separation comes from work by Childs et al. [25], demonstrating an exponential algorithmic speedup by quantum walk. More specifically, they design a "welded trees" graph that consists of two depth-k binary trees welded together by an alternating cycle on the leaves of the two trees, with the two degree-2 vertices (roots) referred to as "ENTRANCE" and "EXIT." The main result of [25] is that a quantum computer, given the ENTRANCE, can find the EXIT in time $\mathrm{poly}(k)$, whereas a classical computer needs exponential time $2^{\Omega(k)}$ to find the EXIT. However, it is not immediately clear how to lift this separation to the realm of graph property testing: it is not even a graph property (since the ENTRANCE is given and the EXIT can be labeled differently in isomorphic graphs), and it is still more challenging to find a related classically hard property *testing* problem.

Our approach is to leverage the quantum advantage for finding ENTRANCE-EXIT pairs to test whether the given graph has a certain form derived from welded trees. The difficulty is that it is unclear how finding ENTRANCE-EXIT pairs helps a quantum computer test the structure of a welded trees graph. One of our main observations is that once the vertices in the weld of the binary trees (i.e., the leaves of the binary trees that are welded) are marked, it becomes easy to test the entire structure: we can separately test the two binary trees and then test the weld itself. However, marking the weld vertices also enables a classical computer to find ENTRANCE-EXIT pairs, so it seems that this approach has no quantum advantage.

Our resolution is to use many copies of the welded trees graph and randomly assign a bit to every root (degree-2) vertex. Then we mark every nonroot vertex v by connecting v to a root vertex r (possibly in another copy) via an "advice edge," such that

1. if v is in the weld, then the sum of the bits assigned to r and its pair (the other degree-2 vertex in the welded trees graph containing r) is odd, and

2. if v is not in the weld, then the sum of the bits assigned to r and its pair is even.

To keep the degree bounded, instead of directly connecting advice edges to the roots, we attach a binary tree "antenna" to the roots and connect the advice edges there, as shown in Figures 1 and 2. The key point is that the marking of the weld vertices can only be read efficiently by a quantum computer.

To show that this property is hard to test classically, we show that it is exponentially difficult to distinguish a randomly selected graph of the form described above, from a graph that looks like the above, except the binary trees are self-welded, i.e., the leaves of the binary trees are connected with a random cycle only containing vertices from the same binary tree (this then disconnects the root-pairs, so the advice edges no longer have the original role and are chosen arbitrarily). The intuition behind our classical lower bound proof is the same as that behind the lower bound proof of [25]: given a root of a (self)-welded trees graph, the graph appears to be an exponentially deep binary tree for a (random) classical walker who can only explore the graph locally. Finally, we show that a random welded trees graph is $\Omega(1)$-far from a random self-welded trees graph by observing that the welded trees graph is bipartite, whereas the self-welded trees graph is $\Omega(1)$-far from being bipartite with very high probability.

1.4. Subsequent work. We briefly review some related subsequent work.

Gilyén, Hastings, and Vazirani demonstrated (sub)exponential advantage of stoquastic adiabatic quantum computation for solving a certain sparse graph problem related to another modification of the glued-trees problem [30]. Toward a better understanding of quantum speedups in the adjacency list model, Balasubramanian, Li, and Harrow studied a broad spectrum of random hierarchical graphs (generalizations of glued-tree graphs) and showed that quantum walk algorithms can exhibit exponential quantum advantage [11].

Continuing the investigation of symmetric Boolean functions, Podder, Yao, and Ye studied a fine-grained version of the Watrous conjecture [50], which states that for any quantum algorithm computing a Boolean function using T queries, there always exists a randomized algorithm using $\mathrm{poly}(T)$ queries that achieves the same success probability, even if the success probability is arbitrarily close to $1/2$. Chakraborty, Kayal, and Paraashar studied transitive Boolean functions and explored the maximum possible separation between various pairs of combinatorial measures [24].

Despite significant progress in understanding quantum advantages in the query model, similar results ruling out superpolynomial quantum speedups for various families of functions are not known for communication complexity. To bridge this gap, Guan et al. studied the communication complexity of permutation-invariant functions and showed that just as for query complexity, quantum and classical communication complexity are polynomially related [35].

1.5. Organization. The remainder of the paper is organized as follows.

In section 2 we introduce some basic notions from query complexity, the theory of permutation groups, and symmetric functions. Section 3 introduces the notion of well-shuffling permutation groups and shows that they do not allow superpolynomial quantum speedups. In section 4 we present techniques for showing that permutation groups are well-shuffling, establishing in particular that hypergraph symmetries cannot have superpolynomial quantum speedups. In section 5 we classify well-shuffling permutation groups, showing that hypergraph symmetries are essentially the only groups that are inconsistent with superpolynomial speedups. Section 6 presents an exponential separation between classical and quantum property testing in the adjacency

list model for bounded-degree graphs. Finally, in section 7 we briefly discuss some open problems. The appendices present two technical results: Appendix A gives a proof of the quantum minimax lemma, and Appendix B establishes a superpolynomial quantum speedup for deep wreath product symmetries.

2. Preliminaries.

2.1. Query complexity. We begin by introducing some standard notation from query complexity. A *Boolean function* is a $\{0,1\}$-valued function f on strings of length $n \in \mathbb{N}$ with alphabet Σ. We use $\mathrm{Dom}(f)$ to denote the domain of f, and we always have $\mathrm{Dom}(f) \subseteq \Sigma^n$, where Σ is a finite alphabet. The function f is called *total* if $\mathrm{Dom}(f) = \Sigma^n$; otherwise it is called *partial.*

For a (possibly partial) Boolean function f, we use $\mathrm{R}_\epsilon(f)$ to denote its *randomized query complexity with error* ϵ, as defined in [20]. This is the minimum number of queries required in the worst case by a randomized algorithm that computes f with worst-case error ϵ. We use $\mathrm{Q}_\epsilon(f)$ to denote the *quantum query complexity with error* ϵ of f, also defined in [20]. This is the minimum number of queries required in the worst case by a quantum algorithm that computes f with worst-case error ϵ. When $\epsilon = 1/3$, we omit it and simply write $\mathrm{R}(f)$ and $\mathrm{Q}(f)$.

An important tool for lower bounding query complexity is the minimax theorem, the original version of which was given by Yao for zero-error (Las Vegas) randomized algorithms [56]. Here we use a bounded-error, quantum version of the minimax theorem. Bounded-error versions of the minimax theorem can be shown using linear programming duality (see also [55], which proved a minimax theorem in the setting where both the error and the expected query complexity are measured against the same hard distribution). A similar technique works for quantum query complexity; this result is folklore, and we prove it in Appendix A.

LEMMA 2.1 (minimax for bounded-error quantum algorithms). *Let f be a (possibly partial) Boolean function with $\mathrm{Dom}(f) \subseteq \Sigma^n$, and let $\epsilon \in (0, 1/2)$. Then there is a distribution μ supported on $\mathrm{Dom}(f)$ which is* hard *for f in the following sense: any quantum algorithm using fewer than $\mathrm{Q}_\epsilon(f)$ quantum queries for computing f must have average error more than ϵ on inputs sampled from μ.*

Note that achieving average error ϵ against a known distribution μ is always easier than achieving worst-case error ϵ; the minimax theorem says that there is a hard distribution against which achieving average error ϵ is just as hard as achieving worst-case error ϵ.

2.2. Permutation groups. We review some basic definitions about permutation groups, following Hulpke [37].

DEFINITION 2.2 (permutation group). *A* permutation group *is a pair (D, G) where D is a set and G is a set of bijections $\pi : D \to D$, such that G forms a group under composition (i.e., G contains the identity function and is closed under composition and inverse of the bijections). In this case, G is said to* act *on D. We will often denote a permutation group simply by G, with the domain D being implicit.*

In other words, a permutation group is simply a set of permutations of a domain D which is closed under composition and inverse. In this work we will generally take $D = [n]$, where $[n]$ denotes the set $\{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. The set $[n]$ will represent the indices of an input string, or equivalently, the queries an algorithm is allowed to make.

We define orbits, transitivity, and stabilizers of permutation groups, all of which are standard.

DEFINITION 2.3 (orbit). *Let G be a permutation group on domain D, and let $i \in D$. Then the* orbit *of i is the set $\{\pi(i) : \pi \in G\}$. A subset of D is an orbit of G if it is the orbit of some $i \in D$ with respect to G.*

DEFINITION 2.4 (transitivity). *We say that a permutation group G on domain D is* k-transitive *if for all distinct $i_1, i_2, \ldots, i_k \in D$ and distinct $j_1, j_2, \ldots, j_k \in D$, there exists some $\pi \in G$ such that $\pi(i_t) = j_t$ for all $t = 1, 2, \ldots, k$. A group that is 1-transitive is called simply* transitive.

DEFINITION 2.5 (stabilizer). *Let G be a permutation group on domain D. The* pointwise stabilizer *of a set $S \subseteq D$ is the subgroup* $\mathrm{Stab}_G(S) := \{\pi \in G : \text{ for all } i \in S\, \pi(i) = i\}$. *The* setwise stabilizer *of a set $S \subseteq D$ is the subgroup* $\mathrm{Stab}_G(\{S\}) := \{\pi \in G : \text{ for all } i \in S\, \pi(i) \in S\}$.

We use some facts about the structure of permutation groups. For this, we need some additional definitions, starting with the notions of primitive groups and wreath products.

DEFINITION 2.6 (block system). *Let G be a permutation group on domain D. A partition $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$ of D is called a* block system *for G if $\mathcal{B}$ is G-invariant. Formally, this means that $\pi(B_i) \in \mathcal{B}$ for every $\pi \in G$ and $i \in [k]$, where $\pi(B_i) := \{\pi(x) : x \in B_i\}$. The sets B_i are called* blocks *for G.*

As an example, consider the action of the group of linear transformations on the nonzero elements of a vector space. The set of lines through the origin (i.e., nonzero scalar multiples of a nonzero vector) form a block system.

It follows from this definition that if $\mathcal{B}$ is a block system for G, then for every block B_i and permutation $\pi \in G$, either $\pi(B_i) = B_i$ or $\pi(B_i) \cap B_i = \emptyset$. Moreover, if G acts transitively, then all blocks have the same size.

DEFINITION 2.7 (primitive group). *Let G be a transitive permutation group on domain D. G is called* primitive *if the only block systems for G are $\{D\}$ and the partition of D into singletons. A permutation group that is not primitive is called* imprimitive.

Primitive groups are sometimes described as the building blocks of permutation groups, for reasons we will see later.

DEFINITION 2.8. *Let G be an abstract group. The* direct power *of G with exponent m, denoted $G^{\times m}$, is the direct product of m copies of G:*

$$G^{\times m} := \underbrace{G \times \cdots \times G}_{m \text{ times}}.$$

If G is a permutation group acting on $[n]$, there are two ways to construct $G^{\times m}$ as a permutation group. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_m) \in G^{\times m}$.

(i) *The action of $G^{\times m}$ on $[m] \times [n]$ is defined by $\pi(i, j) = (i, \pi_i(j))$.*
(ii) *The action of $G^{\times m}$ on $[n]^m$ is defined by $\pi(i_1, i_2, \ldots, i_m) = (\pi_1(i_1), \pi_2(i_2), \ldots, \pi_m(i_m))$.*

These two actions of $G^{\times m}$ give rise to two actions of the *wreath product* of two groups.

DEFINITION 2.9 (wreath product). *Let G and H be permutation groups acting on $[m]$ and $[n]$, respectively. The* wreath product *of G and H, denoted $G \wr H$, is one of two permutation groups:*

(i) *The* imprimitive action *acts on $[m] \times [n]$ and is the group generated by the action of $H^{\times m}$, along with permutations of the form $(i, j) \to (\sigma(i), j)$ for every $\sigma \in G$.*

(ii) *The* product action *acts on $[n]^m$ and is the group generated by the action of $H^{\times m}$, along with permutations of the form $(i_1, i_2, \ldots, i_m) \to (i_{\sigma(1)}, i_{\sigma(2)}, \ldots, i_{\sigma(m)})$ for every $\sigma \in G$.*

A few notes are in order about our definition of the wreath product:

- The wreath product is not commutative. This can be confusing because some authors instead denote this wreath product as $H \wr G$. We prefer the notation used in this definition because of the correspondence to block composition of functions, which we discuss further below.
- This definition differs from a more general definition of the wreath product, which is an abstract group that can be defined even if H is an abstract group. However, the definitions agree when H is a permutation group, which is the only case we consider.
- If it is clear from context which of the two actions we are referring to, then we may sometimes simply write $G \wr H$ without specifying the type of action.

The order of the wreath product is given as below.

FACT 2.10. *Let G and H be permutation groups acting on $[m]$ and $[n]$, respectively. If $n > 1$, then the imprimitive action and the product action of $G \wr H$ are isomorphic as groups, and both have order $|G| \cdot |H|^m$.*

The wreath product imprimitive action carries a useful intuitive interpretation: if g and h are functions that are symmetric under permutation groups G and H, respectively, then $G \wr H$ is the (visible)[4] group of symmetries of the block-composed function $g \circ h$. Thus, the wreath product imprimitive action is associative.

DEFINITION 2.11 (base). *Let G be a permutation group on domain D. A* base *for G is a set $S \subseteq D$ such that the pointwise stabilizer* $\mathrm{Stab}_G(S)$ *is the trivial group. The* minimal base size *for G, denoted $b(G)$, is the size of the smallest base for G.*

Bases of permutation groups are important in computational group theory for the following reason: if S is a base for G, then a permutation $\pi \in G$, viewed as a function $D \to D$, is uniquely determined by its restriction to S.[5] This can be useful particularly when S is small compared to D.

In many applications, the minimal base size is a useful proxy for the size of a permutation group. This is quantified by the following well-known lemma, which we prove for completeness.

LEMMA 2.12. *Let G be a permutation group on $[n]$. Then the order of the group $|G|$ and the minimal base size $b(G)$ satisfy $2^{b(G)} \leq |G| \leq n^{b(G)}$. Equivalently, $\log_n(|G|) \leq b(G) \leq \log_2(|G|)$.*

[4] In rare cases, depending on g and h, there can be more symmetries. For example, let $g = h = \mathsf{OR}_n$, which both have symmetry group S_n. The wreath product $S_n \wr S_n$ is *not* the same as S_{n^2}, which is the group of symmetries of $g \circ h = \mathsf{OR}_{n^2}$. However, we can say that $g \circ h$ is guaranteed to be *at least* symmetric under $G \wr H$ for any functions g and h.

[5] Interestingly, the problem of computing $b(G)$ exactly is known to be NP-hard, but there is a polynomial-time classical algorithm that finds a base of size at most $O(b(G) \log\log |D|)$ [19].

Proof. Without loss of generality, identify a minimal base for G with $[k]$, where $k = b(G)$. Let $H_i = \mathrm{Stab}_G([i])$ be the pointwise stabilizer of the first i points of the base, with the understanding that $H_0 = G$. We have the inclusion

$$1 = H_k \subseteq H_{k-1} \subseteq \cdots \subseteq H_1 \subseteq H_0 = G.$$

Let $O_i \subseteq [n]$ denote the orbit of i in H_{i-1}. By the orbit-stabilizer theorem, we have $|H_{i-1}| = |H_i| \cdot |O_i|$. Thus, we have $|G| = \prod_{i=1}^{k} |O_i|$. We also know that $2 \leq |O_i| \leq n$, where the upper bound is trivial, and the lower bound holds because $[k]$ was assumed to be a minimal base. We get the desired bound on $|G|$ by bounding each term in the product. ∎

2.3. Symmetric functions. We introduce some notation that is used throughout the paper to talk about symmetric functions.

DEFINITION 2.13 (permuting strings). *Let π be a permutation on $[n]$, and let $x \in \{0,1\}^n$. We write $x \circ \pi$ to denote the string whose characters have been permuted by π; that is, $(x \circ \pi)_i := x_{\pi(i)}$. More generally, $x \circ \pi$ is similarly defined when π is merely a function $[n] \to [n]$ rather than a permutation.*

Note that if we view a string $x \in \Sigma^n$ as a function $[n] \to \Sigma$ with $x(i) := x_i$, then $x \circ \pi$ is simply the usual function composition of x and π. This notation allows us to easily define symmetric functions.

DEFINITION 2.14 (symmetric function). *Let G be a permutation group on $[n]$, and let f be a (possibly partial) Boolean function with $\mathrm{Dom}(f) \subseteq \Sigma^n$. We say f is* symmetric under *G if for all $x \in \mathrm{Dom}(f)$ and all $\pi \in G$ we have $x \circ \pi \in \mathrm{Dom}(f)$ and $f(x \circ \pi) = f(x)$.*

For asymptotic bounds such as $\mathrm{Q}(f) = \mathrm{R}(f)^{\Omega(1)}$ to be well-defined, we need to talk about *classes* of functions rather than individual functions. To do that, we need to talk about classes of permutation groups. We introduce the following definition, which defines, for a class of permutation groups $\mathcal{G}$, the set of all functions symmetric under some group in $\mathcal{G}$. We denote this set by $F(\mathcal{G})$.

DEFINITION 2.15 (class of symmetric functions). *Let $\mathcal{G} = \{G_i\}_{i \in I}$ be a (possibly infinite) set of finite permutation groups, with G_i acting on $[n_i]$ for each $i \in I$. Here I is an arbitrary index set and $n_i \in \mathbb{N}$ for all $i \in I$. Then define $F(\mathcal{G})$ to be the set of all (possibly partial) Boolean functions that are symmetric under some G_i. That is, we have $f \in F(\mathcal{G})$ if and only if $f : \mathrm{Dom}(f) \to \{0,1\}$ is a function with $\mathrm{Dom}(f) \subseteq [m]^n$ for some $n, m \in \mathbb{N}$, and f is symmetric under G_i for some $i \in I$ such that $n_i = n$. (Here $[m]$ represents the alphabet Σ.)*

3. Well-shuffling permutation groups. In this section we first define the notion of a well-shuffling class of permutation groups, which is a class $\mathcal{G}$ of permutation groups G that are hard to distinguish from the set of small-range functions via a quantum query algorithm. We then show that a well-shuffling class of permutation groups does not allow superpolynomial quantum speedups. This result (Theorem 3.4) converts the task of showing permutation groups do not allow quantum speedups into the task of showing those permutation groups are well-shuffling, a much simpler objective.

We start by defining the set of small-range strings $D_{n,r}$.

DEFINITION 3.1 (small-range strings). *For $n, r \in \mathbb{N}$, let $D_{n,r}$ be the set of all strings α in $[n]^n$ for which the number of unique alphabet symbols in α is at most r.*

We identify a string $\alpha \in [n]^n$ with a function $[n] \to [n]$. Then $D_{n,r}$ is the set of all functions $[n] \to [n]$ with range size at most r. Next, we define $\mathrm{cost}(G,r)$ as the quantum query complexity of distinguishing G from $D_{n,r}$ (where G is a permutation group acting on $[n]$).

DEFINITION 3.2 (cost). *Identify a permutation on $[n]$ with a string in $[n]^n$ in which each alphabet symbol occurs exactly once. Then a permutation group G on $[n]$ corresponds to a subset of $[n]^n$. For $r < n$, let $\mathrm{cost}_\epsilon(G,r)$ be the minimum number of quantum queries needed to distinguish G from $D_{n,r}$ to worst-case error ϵ; that is, $\mathrm{cost}_\epsilon(G,r) := \mathrm{Q}_\epsilon(f)$, where f has domain $G \cup D_{n,r} \subseteq [n]^n$ and is defined by $f(x) = 1$ if $x \in G$ and $f(x) = 0$ if $x \in D_{n,r}$. When $r \geq n$, we set $\mathrm{cost}_\epsilon(G,r) := \infty$. When $\epsilon = 1/3$, we omit it and write $\mathrm{cost}(G,r)$.*

We note that since $\mathrm{cost}_\epsilon(G,r)$ is defined as the worst-case quantum query complexity of a Boolean function, it satisfies amplification, meaning that the precise value of ϵ does not matter as long as it is a constant in $(0,1/2)$ and as long as we do not care about constant factors.

We define a well-shuffling class of permutation groups as follows.

DEFINITION 3.3 (well-shuffling permutation groups). *Let $\mathcal{G}$ be a collection of permutation groups. We say $\mathcal{G}$ is* well-shuffling *if $\mathrm{cost}(G,r) = r^{\Omega(1)}$ for $G \in \mathcal{G}$ and $r \in \mathbb{N}$. More explicitly, we say $\mathcal{G}$ is well-shuffling* with power *$a > 0$ if there exists $b > 0$ such that $\mathrm{cost}(G,r) \geq r^{1/a}/b$ for all $G \in \mathcal{G}$ and all $r \in \mathbb{N}$.*

We note that $\mathrm{cost}(G,r) \geq r^{1/a}/b$ is always satisfied when r is greater than or equal to the domain size of G, since in that case $\mathrm{cost}(G,r) = \infty$. Hence to show well-shuffling we only need to consider r smaller than n, the domain size of the permutation group G.

The following theorem plays a central role in this work: it shows that a well-shuffling collection of permutation groups does not allow superpolynomial quantum speedups.

THEOREM 3.4. *Let $f : \mathrm{Dom}(f) \to \{0,1\}$ be a partial Boolean function on $n \in \mathbb{N}$ bits, with $\mathrm{Dom}(f) \subseteq \Sigma^n$ (where Σ is a finite alphabet). Let G be a permutation group on $[n]$, and suppose that f is symmetric under G. Then*

$$\mathrm{R}(f) \leq \min\{r \in \mathbb{N} : \mathrm{cost}(G,r) \geq 120\,\mathrm{Q}(f)\}.$$

Consequently, if $\mathcal{G}$ is a well-shuffling collection of permutation groups with power a, then for all $f \in F(\mathcal{G})$ we have $\mathrm{R}(f) = O(\mathrm{Q}(f)^a)$.

To prove this theorem, we use the following minimax theorem for the cost measure.

LEMMA 3.5 (minimax for cost). *Let $r, n \in \mathbb{N}$ satisfy $r < n$, let $\epsilon \in [0,1/2)$, and let G be a permutation group on $[n]$. Then there is a distribution μ on $D_{n,r}$ that is* hard *in the following sense. Let μ' be the uniform distribution on $G \subseteq [n]^n$. Then any quantum algorithm for distinguishing G from $D_{n,r}$ which uses fewer than $\mathrm{cost}_\epsilon(G,r)$ queries must either make error $> \epsilon$ against μ or else make error $> \epsilon$ against μ' (i.e., it fails to distinguish μ on $D_{n,r}$ from μ' on G).*

Proof. Let $f : G \cup D_{n,r} \to \{0,1\}$ be such that $f(x) = 1$ if $x \in G$ and $f(x) = 0$ if $x \in D_{n,r}$. Then by the minimax theorem (Lemma 2.1), there is a hard distribution ν on $G \cup D_{n,r}$, such that any quantum algorithm using fewer than $\mathrm{Q}_\epsilon(f) = \mathrm{cost}_\epsilon(G,r)$ queries must make more than ϵ error against ν. Let ν' be the distribution we get by applying a uniformly random permutation from G to a sample from ν. Then ν' is still a hard distribution in the same sense as ν. Indeed, if ν' were not hard, there would

be some quantum algorithm Q solving f against ν' using too few queries; but in that case, we could design an algorithm Q' for solving f against ν simply by taking the input x, implicitly applying a uniformly random π from G to permute the bits of x (this can be done without querying x, simply by redirecting all future queries $i \in [n]$ through the permutation π), and then running Q on the permuted string.

Now, note that composing a uniformly random permutation from G with an arbitrary fixed permutation from G gives a uniformly random permutation from G. This means that ν' is some mixture of the uniform distribution μ' on $G \subseteq [n]^n$ and another distribution μ on $D_{n,r}$. Then any algorithm that succeeds on both μ and μ' with error at most ϵ will also succeed on ν' with error at most ϵ, which is a contradiction. Hence the lemma follows. ∎

Using this lemma, we now prove Theorem 3.4.

Proof. We claim the following amplification calculations: first, by repeating an algorithm that makes error at most $1/3$ a total of 9 times and taking the majority vote, we can reduce the error below $10/69$; second, by repeating an algorithm that makes error at most $46/105$ a total of 13 times and taking the majority vote, we can reduce the error below $1/3$. Such calculations can be easily checked (for example, using software).

Let Q be a quantum algorithm for f that uses $\mathrm{Q}(f)$ queries. Get a new algorithm Q' by repeating Q 9 times and taking the majority vote; then Q' uses $9\,\mathrm{Q}(f)$ queries and makes worst-case error $< 10/69$ instead of $1/3$. We fix an error level $\epsilon := 46/105$ and apply Lemma 3.5 to get a distribution μ on $D_{n,r}$ such that any quantum algorithm which distinguishes μ from the uniform distribution μ' on G with at most ϵ error must make at least $\mathrm{cost}_\epsilon(G, r)$ queries. Note that we can convert an algorithm distinguishing G from $D_{n,r}$ with worst-case error $46/105$ to one with worst-case error $1/3$ by repeating 13 times; therefore, $\mathrm{cost}(G, r) \leq 13\,\mathrm{cost}_\epsilon(G, r)$, so an algorithm which distinguishes μ from μ' with at most ϵ error must make at least $\mathrm{cost}(G, r)/13$ quantum queries.

We now describe a randomized algorithm R for f. On input x, the algorithm samples α from μ. It then queries the indices of x which are in the range of α (a total of r queries) and constructs the string $x \circ \alpha$ using these queried bits. On this string, R runs the algorithm Q' offline (this requires no queries and therefore has no cost). It outputs the result.

Assuming that $\mathrm{cost}(G, r) \geq 120\,\mathrm{Q}(f)$, we show the correctness of R. In this case, we have $\mathrm{cost}(G, r)/13 > 9\,\mathrm{Q}(f)$. This means that the number of queries used by Q' is not sufficient to distinguish μ from μ' to error $\epsilon = 46/105$. However, if R is not correct to error $1/3$, we will show that Q' can be converted to such a distinguishing algorithm, getting a contradiction.

To this end, let x be an input on which R makes error more than $1/3$. Then when α is sampled from μ, the value of $Q'(x \circ \alpha)$ is not equal to $f(x)$ with probability q greater than $1/3$. However, we know that for π sampled from μ', the value of $Q'(x \circ \pi)$ is equal to $f(x \circ \pi) = f(x)$ except for error probability $p < 10/69$. So Q' acts differently on $x \circ \alpha$ versus $x \circ \pi$, where $\alpha \sim \mu$ and $\pi \sim \mu'$. Note that applying Q' to one of these (with the input x hard-coded in) uses only $9\,\mathrm{Q}(f)$ quantum queries to α or π.

To get a distinguishing algorithm from here, we do the following: given α sampled from μ or μ', output $1 - f(x)$ with probability a and run $Q'(x \circ \alpha)$ with probability $1-a$, outputting the result. Then on μ and μ', the probabilities of outputting $1-f(x)$ will be $a+(1-a)p$ and $a+(1-a)q$, respectively. Picking $a = 12/35$ and using $p < 10/69$ and $q > 1/3$, we see that on μ and μ', the probabilities of outputting $1 - f(x)$ will

be $< 46/105$ and $> 59/105$, respectively. However, this is a contradiction since this distinguishing algorithm distinguishes μ from μ' with at most $\epsilon = 46/105$ error and uses $9\,\mathrm{Q}(f) < \mathrm{cost}(G,r)/13$ queries.

Therefore, on all inputs, the randomized algorithm R is correct with probability at least $2/3$, and it uses at most r queries, as desired. ∎

The upshot of Theorem 3.4 is that we can show that a class of permutation groups $\mathcal{G}$ does not allow superpolynomial quantum speedups simply by showing that $\mathcal{G}$ is well-shuffling—that is, by showing that $G \in \mathcal{G}$ is hard to distinguish from the set of small-range functions $D_{n,r}$ using a quantum query algorithm.

4. Showing permutation groups are well-shuffling. In this section, we introduce some tools for showing that a collection of permutation groups is well-shuffling. Due to Theorem 3.4, a well-shuffling collection of permutation groups does not allow any superpolynomial quantum speedups for the class of functions symmetric under it, so these tools can be directly used to show that certain symmetries are not consistent with large quantum speedups.

4.1. The symmetric group. The first fundamental result is that the class of full symmetric permutation groups S_n is well-shuffling. This was shown by Zhandry [59] in a different context, though we also provide a simpler proof by a reduction from the collision problem.

THEOREM 4.1. *There is a universal constant C such that any quantum algorithm distinguishing a permutation in S_n from a string in $D_{n,r}$ must make at least $r^{1/3}/C$ queries.*

This theorem says that S_n is hard to distinguish from $D_{n,r}$ (moreover, Zhandry [59] showed that the hard distribution over $D_{n,r}$ is uniform, but we do not need this fact).

Proof. When n is a multiple of r, then each (n/r)-to-1 function has range r and each 1-to-1 function is a permutation; hence distinguishing (n/r)-to-1 from 1-to-1 functions is a subproblem of distinguishing $D_{n,r}$ from S_n. This subproblem is the collision problem, from which an $\Omega(r^{1/3})$ lower bound directly follows [5, 6, 43].

When n is not a multiple of r, set $r' = \lfloor r/2 \rfloor$ and $n' = r'\lfloor n/r' \rfloor$. We argue analogously that distinguishing (n'/r')-to-1 from 1-to-1 functions reduces to distinguishing $D_{n,r}$ from S_n. Given a function $x : [n'] \to [n']$ which is either (n'/r')-to-1 or 1-to-1, we extend its domain to $x : [n] \to [n]$ by defining $x(i) = i$ for each $i > n'$. Then, each (n'/r')-to-1 function extends to a function of range at most $2r' \le r$, whereas each 1-to-1 function extends to a permutation. Hence, distinguishing (n'/r')-to-1 from 1-to-1 functions with domain size n' is a subproblem of distinguishing $D_{n,r}$ from S_n. The same $\Omega(r^{1/3})$ lower bound on the collision problem applies, because r' and n' differ from r and n by constant multiplicative factors. ∎

From Theorem 4.1, the following two corollaries immediately follow (in light of Theorem 3.4).

COROLLARY 4.2. *The set of symmetric groups $\mathcal{S} = \{S_n\}_{n\in\mathbb{N}}$ is well-shuffling with power 3.*

COROLLARY 4.3. *All (possibly partial) Boolean functions f that are symmetric under the full symmetric group S_n satisfy $\mathrm{R}(f) = O(\mathrm{Q}(f)^3)$.*

Apart from Theorem 4.1, the main tools we use to prove the well-shuffling property are transformations on permutation groups that approximately preserve $\mathrm{cost}(G,r)$. We outline several such transformations and invariances. Since we prove Theorem 4.1 by a reduction from collision, and since our main tools in the remainder of this section are additional reductions, effectively all lower bounds in this section work by reductions from collision.

4.2. Reduction between similar-looking permutation groups. We next show that similar-looking permutation groups have similar costs. Here, two groups G and H are "similar looking" if a random permutation from G is indistinguishable from a random permutation from H when queried in sufficiently few places. More formally, we have the following theorem.

THEOREM 4.4 (similar-looking permutation groups have similar costs). *Suppose G and H are permutation groups on $[n]$ and $k \le n$ is a positive integer such that for each $i_1, i_2, \ldots, i_k, j_1, j_2, \ldots, j_k \in [n]$,*

$$\Pr_{\pi \leftarrow G}[\forall \ell\, \pi(i_\ell) = j_\ell] = \Pr_{\pi \leftarrow H}[\forall \ell\, \pi(i_\ell) = j_\ell].$$

Then $\mathrm{cost}(H,r) \ge \min\{k/2, \mathrm{cost}(G,r)\}$. In particular, if $k \ge n^{\Omega(1)}$ and $\mathrm{cost}(G,r) \ge r^{\Omega(1)}$, then $\mathrm{cost}(H,r) \ge r^{\Omega(1)}$.

The proof of Theorem 4.4 is based on the polynomial method and closely mirrors some existing arguments in the literature that k-wise indistinguishability fools $k/2$-query quantum algorithms (e.g., [39, 58]).

Proof of Theorem 4.4. Let Q be a quantum algorithm for distinguishing H from $D_{n,r}$ which uses $\mathrm{cost}(H,r)$ and achieves worst-case error $1/3$. If $\mathrm{cost}(H,r) \ge k/2$, we are done, so assume $\mathrm{cost}(H,r) < k/2$. Now, Q can be converted into a polynomial of degree at most $d := 2\,\mathrm{cost}(H,r)$ in the variables z_{ij}, where $z_{ij} = 1$ if the input π satisfies $\pi(i) = j$ and otherwise $z_{ij} = 0$ (see [5]). This polynomial p satisfies $p(\pi) \in [0, 1/3]$ if $\pi \in D_{n,r}$ and $p(\pi) \in [2/3, 1]$ if $\pi \in H$. It has n^2 variables and degree $d < k$.

Now, the expected output of $p(\pi)$ when π is sampled uniformly from H is a linear combination of the expectations of the monomials of p. For each monomial $\prod_{\ell=1}^{d} z_{i_\ell j_\ell}$, this expectation is just the probability that, for every $\ell \in [d]$, $\pi(i_\ell) = j_\ell$. By the condition on G and H, this is equal to the expectation under $\pi \leftarrow G$, because $d < k$. It follows that the expectation of $p(\pi)$ when $\pi \leftarrow H$ is equal to the expectation of $p(\pi)$ when $\pi \leftarrow G$. But this expectation is simply the acceptance probability of Q. Hence the acceptance probability of Q on the uniform distribution on H equals its acceptance probability on the uniform distribution on G.

Since Q distinguishes H from $D_{n,r}$, it distinguishes the uniform distribution on H from any string in $D_{n,r}$ to error $1/3$. Since it does not distinguish the uniform distribution on H from the uniform distribution on G, Q must also distinguish the uniform distribution on G from any input in $D_{n,r}$ to error $1/3$. Hence, by convexity, Q also distinguishes (to error $1/3$) the uniform distribution on G from the distribution μ on $D_{n,r}$ given in Lemma 3.5. By Lemma 3.5, this implies that Q makes at least $\mathrm{cost}(G,r)$ queries, and therefore $\mathrm{cost}(H,r) \ge \mathrm{cost}(G,r)$, as desired. ∎

An immediate consequence is that highly transitive permutation groups are well-shuffling.

COROLLARY 4.5. *If H is k-transitive, then $\mathrm{cost}(H,r) = \Omega(\min\{k, r^{1/3}\})$, where the constant in the big-Ω is universal.*

Proof. Suppose H acts on $[n]$. Consider any distinct $i_1, \ldots, i_k \in [n]$ and distinct $j_1, \ldots, j_k \in [n]$. Because H is k-transitive (Definition 2.4), there exists a permutation $\sigma \in H$ such that $\sigma(i_\ell) = j_\ell$ for all $\ell \in [k]$. As a consequence, we have

$$\begin{aligned}\Pr_{\pi \leftarrow H}[\forall \ell \in [k]\, \pi(i_\ell) = j_\ell] &= \Pr_{\pi \leftarrow H}[\forall \ell \in [k]\, \sigma^{-1}(\pi(i_\ell)) = i_\ell] \\ &= \Pr_{\pi \leftarrow H}[\forall \ell \in [k]\, \pi(i_\ell) = i_\ell] \\ &= \prod_{\ell=1}^{k} \frac{1}{n-\ell+1},\end{aligned}$$

where the last line uses the k-transitivity of H and the orbit-stabilizer theorem.

Set $G = S_n$. By the same argument as above, observe that

$$\Pr_{\pi \leftarrow G}[\forall \ell \in [k]\, \pi(i_\ell) = j_\ell] = \Pr_{\pi \leftarrow H}[\forall \ell \in [k]\, \pi(i_\ell) = j_\ell],$$

because G is k-transitive. Since this equality holds for any distinct $i_1, \ldots, i_k \in [n]$ and distinct $j_1, \ldots, j_k \in [n]$, the statement of the corollary now follows directly from Theorem 4.4. ∎

However, Corollary 4.5 is less powerful than it seems: it is a consequence of the classification of finite simple groups that the only 6-transitive permutation groups are the symmetric group S_n for $n \geq 6$ and the alternating group A_n for $n \geq 8$, both in their natural action on $[n]$ [28]. Nevertheless, it will be important for us later that the alternating group A_n, which is $(n-2)$-transitive, satisfies $\mathrm{cost}(A_n, r) = \Omega(r^{1/3})$.

4.3. Transformations for graph symmetries. Next, we introduce some additional transformations on permutation groups that approximately preserve the cost. The transformations in this section will allow us to show that graph property permutation groups (and several variants of them) are well-shuffling.

4.3.1. Transformation for directed graphs. We start by defining an extension of a permutation group G on $[n]$ to a permutation group acting on $[n]^\ell$. The notation in the definition below comes from [40].

DEFINITION 4.6. *Let G be a permutation group on domain D, and let $\ell \in \mathbb{N}$. Define $G^{(\ell)}$ to be the permutation group that acts on domain D^ℓ by $\pi(i_1, i_2, \ldots, i_\ell) = (\pi(i_1), \pi(i_2), \ldots, \pi(i_\ell))$ for each $\pi \in G$ (so the number of permutations in $G^{(\ell)}$ is the same as the number of permutations in G).*

Define $G^{\langle \ell \rangle}$ to be the permutation group $G^{(\ell)}$ with domain restricted to the subset $D^{\langle \ell \rangle} \subseteq D^\ell$ consisting of all distinct ℓ-tuples of elements of D.

We show that these transformations both preserve the cost, at least when ℓ is constant. We start with $G^{(\ell)}$.

THEOREM 4.7. *Let G be a permutation group on $[n]$, and let H be the permutation group $G^{(\ell)}$. Then $\mathrm{cost}(H, r^\ell) \geq \mathrm{cost}(G, r)/\ell$.*

Proof. Let Q be an algorithm distinguishing H from D_{n^ℓ, r^ℓ} that makes at most $\mathrm{cost}(H, r^\ell)$ queries. We construct an algorithm Q' for distinguishing G from $D_{n,r}$ as follows. Given an input $\alpha' \in G \cup D_{n,r}$ to Q', Q' runs Q on $\alpha : [n]^\ell \to [n]^\ell$ defined by $\alpha(z) = (\alpha'(z_1), \alpha'(z_2), \ldots, \alpha'(z_\ell))$. Note that if α' has range r, then α has range at most r^ℓ, whereas if $\alpha' \in G$, then $\alpha \in H$. This is to say that Q' distinguishes G from $D_{n,r}$. Because each query to α can be simulated using ℓ queries to α', the number of queries made by Q' to α' is ℓ times the number of queries made by Q to α, from which it follows that $\mathrm{cost}(G, r) \leq \ell \cdot \mathrm{cost}(H, r^\ell)$. ∎

To handle $G^{\langle \ell \rangle}$, we first observe that restricting the domain of a permutation group to a union of some of its orbits does not decrease its cost.

LEMMA 4.8. *Let G be a permutation group on $[n]$, and let $S \subseteq [n]$ be a union of some orbits of G. Let G' be the permutation group G acting only on S. Then* $\mathrm{cost}(G', r) \geq \mathrm{cost}(G, r)$.

Proof. We identify S with $[|S|]$ without loss of generality. If Q distinguishes G' from $D_{|S|,r}$, then we can turn it into Q' distinguishing G from $D_{n,r}$ by having Q' run Q and make queries only from $[|S|]$. ∎

The fact that $G^{\langle \ell \rangle}$ does not decrease the cost of G too much then follows as a corollary of Theorem 4.7 and Lemma 4.8.

COROLLARY 4.9. *Let G be a permutation group on $[n]$, and let H be the permutation group $G^{\langle \ell \rangle}$. Then* $\mathrm{cost}(H, r^\ell) \geq \mathrm{cost}(G, r)/\ell$.

Proof. It suffices to note that $G^{\langle \ell \rangle}$ is the permutation group $G^{(\ell)}$ with domain restricted to $[n]^{\langle \ell \rangle}$, which is a union of orbits because $\pi \in G^{(\ell)}$ always sends a tuple with unique entries to another tuple with unique entries (since a permutation on $[n]$ is applied to each entry). The desired result then follows from Theorem 4.7 and Lemma 4.8. ∎

We now observe that the transformation $G^{\langle \ell \rangle}$ immediately allows us to show that directed graph symmetries are well-shuffling.

COROLLARY 4.10 (directed graph symmetries). *The set $\mathcal{G} = \{G_k\}_{k \in \mathbb{N}, k \geq 2}$ of all directed graph symmetries is well-shuffling with power* 6.[6] *Here the permutation group G_k acts on a domain of size $n = k(k-1)$ representing the possible arcs of a k-vertex directed graph, and G_k consists of all $k!$ permutations on these arcs that act by relabeling the vertices.*

Proof. This immediately follows by observing that $G_k = S_k^{\langle 2 \rangle}$. To see this, note that the domain of G_k is the set of all ordered pairs $(x, y) \in [k]$ with $x \neq y$, which is precisely $[k]^{\langle 2 \rangle}$, and the permutations in G_k are just those in S_k applied to both coordinates, which is precisely relabeling the vertices. Corollary 4.9 then gives $\mathrm{cost}(G_k, r) \geq \mathrm{cost}(S_k, \sqrt{r})/2$, which is at least $\Omega(r^{1/6})$ by Theorem 4.1. ∎

The collection of directed hypergraph symmetries is similarly well-shuffling.

COROLLARY 4.11 (directed hypergraph symmetries). *The set $\mathcal{G}_p = \{G_k\}_{k \in \mathbb{N}, k \geq p}$ of all directed p-uniform hypergraph symmetries is well-shuffling with power $3p$. Here the permutation group G_k acts on a domain of size $n = k(k-1)\cdots(k-p+1)$ representing the possible arcs of a k-vertex directed p-uniform hypergraph, and G_k consists of all $k!$ permutations on these arcs that act by relabeling the vertices.*

Proof. This follows from the same argument as Corollary 4.10. ∎

4.3.2. Transformation for undirected graphs. To handle undirected graphs, we introduce yet another operation on permutation groups which approximately preserves the cost.

[6] We do not know whether this power 6 is tight; 6 is only an upper bound on the power of well-shuffling. We similarly do not know the tight well-shuffling powers for the other (hyper)graph symmetries considered later in this section, because we only prove upper bounds.

THEOREM 4.12. *Let G be a permutation group, and let $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$ be a block system for G, where the blocks have equal size. Let H be the permutation group on $[k]$ induced by the action of G on the blocks. Then* $\mathrm{cost}(H, r) \geq \mathrm{cost}(G, r)$.

Proof. Let Q be a quantum algorithm distinguishing H from $D_{k,r}$ using $\mathrm{cost}(H, r)$ queries. We construct a quantum algorithm Q' for distinguishing G from $D_{n,r}$. The algorithm Q' fixes a unique $i_t \in B_t$ for each $t = 1, 2, \ldots, k$. On input α from $G \cup D_{n,r}$, the algorithm Q' runs Q as follows: each query $t \in [k]$ that Q makes is turned into the query $i_t \in [n]$ for α, and the output $\alpha(i_t)$ is converted into the symbol t' such that $\alpha(i_t) \in B_{t'}$ and returned to Q. In this way, the algorithm Q' effectively runs Q on the mapped string $\phi(\alpha) \in [k]^k$, where $\phi(\alpha)_t$ is the symbol t' such that $\alpha(i_t) \in B_{t'}$.

Now, if $\alpha \in D_{n,r}$, then $\phi(\alpha) \in D_{k,r}$, while if $\alpha \in G$, we have $\phi(\alpha) \in H$. Since Q distinguishes H from $D_{k,r}$, it follows that Q' distinguishes G from $D_{n,r}$ using the same number of queries, as desired. ∎

We are now finally ready to prove the formal version of Theorem 1.2, showing that the collection of (undirected) graph symmetries is well-shuffling.

DEFINITION 4.13 (graph symmetries). *The collection of* graph symmetries *is the set $\mathcal{G} = \{G_k\}_{k \in \mathbb{N}}$ of permutation groups with G_k acting on $[n]$ with $n = k(k-1)/2$, such that the domain $[n]$ represents the set of all possible edges in a k-vertex graph, and G_k acts on these edges and permutes them in a way that corresponds to relabeling the vertices of the underlying graph.*

COROLLARY 4.14. *The set of all graph symmetries is well-shuffling with power 6. Hence* $\mathrm{R}(f) = O(\mathrm{Q}(f)^6)$ *for functions f symmetric under a graph symmetry.*

Proof. Let G be a directed graph symmetry on domain size $k(k-1)$, and partition this domain into $k(k-1)/2$ sets of size 2 of the form $\{(x, y), (y, x)\}$ for $x, y \in [k]$. Then the induced permutation group H on these sets (from Theorem 4.12) is precisely the undirected graph symmetry on graphs of size k. Since $\mathrm{cost}(H, r) \geq \mathrm{cost}(G, r)$, and since the directed graph symmetries are well-shuffling with power 6, it follows that the undirected graph symmetries are also well-shuffling with power 6. ∎

Using similar arguments, we can show a similar result for hypergraphs.

COROLLARY 4.15. *For every constant $p \in \mathbb{N}$, the collection of all p-uniform hypergraph symmetries is well-shuffling with power $3p$.*

4.3.3. Transformations for bipartite graphs. We introduce yet more operations on permutation groups for the case of bipartite graph symmetries.

DEFINITION 4.16 (product of permutation groups). *Let G_1 and G_2 be two permutation groups acting on $[n_1]$ and $[n_2]$, respectively. Then the product permutation group $G_1 \times G_2$ is a permutation group acting on $[n_1 n_2]$ such that for any $(\pi_1, \pi_2) \in G_1 \times G_2$, and any $k \in [n_1]$ and $\ell \in [n_2]$, we have $(\pi_1, \pi_2)(k, \ell) = (\pi_1(k), \pi_2(\ell))$. (Here we identify $[n_1] \times [n_2]$ with $[n_1 n_2]$.)*

THEOREM 4.17. *For all G_1 and G_2 acting on $[n_1]$ and $[n_2]$, respectively, and for all r,* $\mathrm{cost}(G_1 \times G_2, r^2) \geq \min\{\mathrm{cost}(G_1, r), \mathrm{cost}(G_2, r)\}$.

Proof. Let $H = G_1 \times G_2$ and $m = n_1 n_2$. Let Q be an algorithm distinguishing H from D_{m,r^2}. Let μ_1 be the hard distribution for G_1 (on $D_{n_1,r}$), and let μ_2 be the hard distribution for G_2 (on $D_{n_2,r}$). Let μ' be the distribution on $D_{m,r}$ that we get by sampling $\alpha_1 \leftarrow \mu_1$, and $\alpha_2 \leftarrow \mu_2$ independently, and returning $\alpha' = (\alpha_1, \alpha_2)$. Note that if α_1 and α_2 have range r, then α' has range at most r^2. Now, since Q distinguishes

D_{m,r^2} from $G_1 \times G_2$, it must also distinguish μ' from the uniform distribution over $G_1 \times G_2$, which itself is the product of the uniform distribution on G_1 and the uniform distribution on G_2. Let ν_1 be the uniform distribution on G_1, and let ν_2 be the uniform distribution on G_2. Consider the behavior of Q on $\mu_1 \times \nu_2$. It must either distinguish this distribution from $\mu_1 \times \mu_2$ or else from $\nu_1 \times \nu_2$ (since it distinguishes $\mu_1 \times \mu_2$ and $\nu_1 \times \nu_2$ from each other). In the first case, we can construct Q' which artificially generates the sample from μ_1 and uses Q to distinguish μ_2 from ν_2. In the second case, we can construct Q' which artificially generates the sample from ν_2 and uses Q to distinguish μ_1 from ν_1. Hence $\mathrm{cost}(G_1 \times G_2, r^2) \geq \min\{\mathrm{cost}(G_1, r), \mathrm{cost}(G_2, r)\}$, as desired. ∎

COROLLARY 4.18. *The collection $\mathcal{G}$ of all bipartite graph symmetries with equal parts is well-shuffling with power* 6.

Proof. This immediately follows by observing that bipartite graph symmetries are the symmetries $S_k \times S_k$. Then Theorems 4.17 and 4.1 give the desired result. ∎

4.3.4. Other transformations. We introduce one final transformation, which merges two permutation groups into one. This transformation also does not decrease the cost.

LEMMA 4.19 (merger). *Let G and H be two permutation groups on $[n]$, and let $F = \langle G, H \rangle$ be the group generated by G and H (i.e., the closure of $G \cup H$ under composition of permutations). Then* $\mathrm{cost}(F, r) \geq \mathrm{cost}(G, r)$.

Proof. Since G is a subset of F, distinguishing G from $D_{n,r}$ is strictly easier than distinguishing F from $D_{n,r}$. ∎

5. Classification of well-shuffling permutation groups. In this section, we show that the results from the previous section are qualitatively optimal, in the sense that permutation groups constructed out of hypergraph symmetries are essentially the *only* groups that are not consistent with superpolynomial quantum speedups. Our starting point is an observation that superpolynomial quantum speedups can be constructed out of any permutation group with sufficiently small minimal base size.

PROPOSITION 5.1. *Let G be a permutation group on $[n]$, and let f be a (possibly partial) Boolean function with* $\mathrm{Dom}(f) \subseteq \Sigma^n$ *for some alphabet Σ. Then there exists a partial Boolean function g that is symmetric under G such that $Q(g) \leq Q(f) + b(G)$ and $R(g) \geq R(f)$.*

Proof. Define a promise problem g as follows. A string $x \in (\Sigma \times [n])^n$ is in the promise if there exists a permutation $\pi \in G$ such that for all i, $x_{\pi(i)} = (y_i, i)$, where $y \in \Sigma^n$ is in the promise of f. In other words, the promise for g contains all reorderings of strings of the form (y_i, i) by permutations that are in G. Naturally, in this case we define $g(x) = f(y)$. Then clearly $R(g) \geq R(f)$, because any randomized algorithm that solves g also solves g restricted to the promise that π is the identity permutation, in which case g is equivalent to f. Moreover, this problem is clearly symmetric under G. On the other hand, $Q(g) \leq Q(f) + b(G)$: by querying a minimal base for G, the algorithm can learn the unique permutation π such that $x_{\pi(i)} = (y_i, i)$. Now, the algorithm just permutes x according to π and runs the query algorithm for f. ∎

Put another way, if $b(G) = n^{o(1)}$, then by choosing an arbitrary function f with $Q(f) = n^{o(1)}$ and $R(f) = n^{\Omega(1)}$ (e.g., Simon's problem [52] or Forrelation [2]), then one can construct a function g symmetric under G with $Q(g) = n^{o(1)}$ and $R(g) = n^{\Omega(1)}$.

Thus, permutation groups that do not allow superpolynomial speedups must have large minimal base size.

Naturally, this raises the question of whether a sort of converse holds, i.e., whether $b(G) = n^{\Omega(1)}$ implies that G is well-shuffling. We will show that this is true for primitive permutation groups (Theorem 5.4) and moreover that primitive groups with $b(G) = n^{\Omega(1)}$ all look roughly like symmetries of p-uniform hypergraphs (Corollary 5.5). Thus, we completely classify which primitive permutation groups are consistent with superpolynomial quantum speedups, and which are not.

For imprimitive groups, we will see that there exist permutation groups G with $b(G) = n^{\Omega(1)}$ that are consistent with large quantum speedups. So, $b(G) = n^{\Omega(1)}$ is not equivalent to the well-shuffling property in general. Nevertheless, we can show that primitive permutation groups with $b(G) = n^{\Omega(1)}$ are the building blocks of well-shuffling group actions. Indeed, we show in Corollary 5.12 that if a collection of imprimitive permutation groups is inconsistent with superpolynomial speedups, then it must be built out of a constant number of well-shuffling primitive groups. Thus, in some sense, all permutation groups that do not allow large quantum speedups must involve symmetries of p-uniform hypergraphs.

5.1. Primitive groups. For primitive permutation groups, the following theorem due to Liebeck [46] shows that there is a tight connection between minimal base size and the structure of the group. Its proof relies on the classification of finite simple groups.

THEOREM 5.2. *Let G be a primitive permutation group on $[n]$. Then one of the following holds:*

(i) *G is a subgroup of $S_\ell \wr S_m$ containing $A_m^{\times \ell}$, where the action of S_m is on p-element subsets of $[m]$ and the wreath product has the product action of degree $n = \binom{m}{p}^\ell$, or*

(ii) *$b(G) < 9 \log_2 n$.*

Here, S_n denotes the symmetric group on $[n]$ and A_n denotes the alternating group on $[n]$.

Observe that in case (i), when $\ell = 1$, this corresponds precisely to either the set of p-uniform undirected hypergraph symmetries with m vertices or the subgroup of all even permutations of the vertices (the alternating group A_m, which has half the size of S_m). When p is a constant, we know that these groups are well-shuffling. More generally, as long as both ℓ and p are constant, then these groups are well-shuffling.

COROLLARY 5.3. *For all constant ℓ, p, the collection of primitive permutation groups that satisfy case* (i) *of Theorem* 5.2 *is well-shuffling with power $3\ell p$.*

Proof. Suppose G, ℓ, p, and m are as in case (i) of Theorem 5.2. Let A denote the action of the alternating group A_m on $[m]$, and let H denote the action of A_m on size-p subsets of $[m]$. Formally, we have the following chain of inequalities:

$$\begin{aligned}
\operatorname{cost}(G, r) &\geq \operatorname{cost}(H^{\times \ell}, r) && \text{Lemma 4.19} \\
&\geq \operatorname{cost}(H^{(\ell)}, r) && \text{Lemma 4.19} \\
&\geq \operatorname{cost}(H, r^{1/\ell})/\ell && \text{Theorem 4.7} \\
&\geq \operatorname{cost}(A^{\langle p \rangle}, r^{1/\ell})/\ell && \text{Theorem 4.12} \\
&\geq \operatorname{cost}(A, r^{1/\ell p})/\ell p && \text{Corollary 4.9} \\
&\geq \Omega\big(r^{1/3\ell p}\big) && \text{Corollary 4.5.}
\end{aligned}$$

In words, in the first two lines, it suffices to show that a subgroup of G is well-shuffling and thus that a subgroup of a subgroup is well-shuffling. $H^{(\ell)}$ is the so-called diagonal subgroup of $H^{\times \ell}$, consisting of the permutations $(\pi_1, \pi_2, \ldots, \pi_\ell)$ such that $\pi_i = \pi_j$ for all i, j; this is consistent with the notation from Definition 4.6. The third inequality appeals to the cost lower bound for power actions. Then, we observe that H is equivalent to the action of $A^{\langle p \rangle}$ on blocks, where $A^{\langle p \rangle}$ is the power action of A restricted to tuples with no repeats, and the blocks are sets of tuples that are reorderings of each other. The next inequality uses the cost lower bound for this restriction of the power action. Finally, we appeal to the fact that the alternating groups on m points are $(m-2)$-transitive, and thus well-shuffling. ∎

In fact, the following theorem shows that a collection of primitive permutation groups is well-shuffling if and only if the permutation groups are as in case (i) of Theorem 5.2, with ℓ and p both constant.

THEOREM 5.4. *Fix some constant $\epsilon > 0$, and let $\mathcal{G}$ be the collection of all primitive permutation groups G that satisfy $b(G) \geq n^\epsilon$:*

$$\mathcal{G} := \{G : n \in \mathbb{N}, G \text{ is a primitive permutation group on } [n], b(G) \geq n^\epsilon\}.$$

Then there exists $c \in \mathbb{N}$ such that $\mathcal{G}$ is well-shuffling with power c. Moreover, for all sufficiently large n, any $G \in \mathcal{G}$ satisfies case (i) *of Theorem* 5.2 *and has $\ell p \leq c/3$.*

Proof. We prove the "moreover" part first. For sufficiently large n, any $G \in \mathcal{G}$ cannot be in case (ii) of Theorem 5.2 because $n^\epsilon \leq b(G) \leq 9 \log_2 n$ is only possible for bounded values of n.

We next show that any $G \in \mathcal{G}$ in case (i) of Theorem 5.2 has $\ell p \leq c/3$ for some constant c to be chosen later. Using the notation of Theorem 5.2, we must have $\binom{m}{p} \geq 2$ because otherwise the definition of the wreath product (Definition 2.9) makes G the trivial group. In particular, this implies $m \geq 2$. The minimal base size of G is upper bounded by

$$\begin{aligned}
b(G) &\leq \log_2(|G|) && \text{Lemma 2.12} \\
&\leq \log_2(|S_\ell \wr S_m|) && G \subseteq S_\ell \wr S_m \\
&= \log_2(\ell! \cdot m!^\ell) && \text{Fact 2.10} \\
&\leq \ell \log_2 \ell + \ell m \log_2 m \\
&\leq 2\ell^2 m^2.
\end{aligned}$$

Thus we have

$$\left(\frac{m}{p}\right)^{\ell p \epsilon} \leq \binom{m}{p}^{\ell \epsilon} = n^\epsilon \leq b(G) \leq 2\ell^2 m^2,$$

so

$$\ell p \epsilon (\log_2 m - \log_2 p) \leq 1 + 2 \log_2 \ell + 2 \log_2 m.$$

Consider two cases. First, suppose $m \leq p^2$. Without loss of generality, we may always assume $p \leq m/2$, because the action of S_m on p-element subsets is equivalent to the action on $(m-p)$-element subsets, by identifying each p-element subset with its complement in $[m]$. So we have

$$\ell p \epsilon \leq \ell p \epsilon (\log_2 m - \log_2 p) \leq 1 + 2 \log_2 \ell + 2 \log_2 m \leq 1 + 2 \log_2 \ell + 4 \log_2 p.$$

But clearly $\ell p \epsilon \leq 1 + 2 \log_2 \ell + 4 \log_2 p$ is possible only for bounded values of ℓp.

Otherwise, suppose $m \geq p^2$. Then we have

$$\ell p\epsilon \leq \frac{1+2\log_2 \ell + 2\log_2 m}{\log_2 m - \log_2 p} \leq 2\frac{1+2\log_2 \ell + 2\log_2 m}{\log_2 m}$$
$$= 4 + \frac{2+4\log_2 \ell}{\log_2 m} \leq 6 + 4\log_2 \ell.$$

Likewise, $\ell p\epsilon \leq 6 + 4\log_2 \ell$ is possible only for bounded values of ℓp. So we may suppose that $\ell p \leq c/3$ for some constant c.

To summarize, we have shown that all but finitely many groups $G \in \mathcal{G}$ satisfy case (i) of Theorem 5.2 with $\ell p \leq c/3$. Consequently, the first part of the theorem now follows from Corollary 5.3, because the well-shuffling property is unaffected by adding or removing the finitely many groups G that do not satisfy case (i) of Theorem 5.2 with $\ell p \leq c/3$; we can always increase the constant b in Definition 3.3. ∎

COROLLARY 5.5. *Let $\mathcal{G}$ be a collection of primitive permutation groups. The following are equivalent:*

(i) *$\mathcal{G}$ is well-shuffling.*
(ii) *$b(G) = n^{\Omega(1)}$ for $G \in \mathcal{G}$ acting on $[n]$.*
(iii) *All but finitely many $G \in \mathcal{G}$ satisfy case* (i) *of Theorem* 5.2 *with $\ell p = O(1)$.*

Proof. First observe that (i) implies (ii) by Proposition 5.1, because well-shuffling permutation groups are not consistent with superpolynomial quantum speedups. (ii) implies (iii) by Theorem 5.4. Finally, (iii) implies (i) by Corollary 5.3, because adding or removing finitely many groups G does not affect the well-shuffling property; we can always increase the constant b in Definition 3.3. ∎

COROLLARY 5.6. *Let $\mathcal{G}$ be a collection of primitive permutation groups. Then the following hold:*

(i) *If $b(G) = n^{\Omega(1)}$ for $G \in \mathcal{G}$ acting on $[n]$, then $\mathcal{G}$ is well-shuffling, and thus $R(f) = Q(f)^{O(1)}$ for all $f \in F(\mathcal{G})$.*
(ii) *If $b(G) = n^{o(1)}$ for $G \in \mathcal{G}$ acting on $[n]$, then there exists a class of partial functions $f \in F(\mathcal{G})$ with $R(f) = Q(f)^{\omega(1)}$.*

It would be interesting to prove some version of Corollary 5.6 directly using the definition of $b(G)$, without appealing to such deep results about the structure of permutation groups.

5.2. Imprimitive groups. Starting with an arbitrary permutation group G, the first place to look for structure is in the orbits of G. We first observe that quantum speedups can be constructed out of any permutation group with many orbits.

PROPOSITION 5.7. *Let G be a permutation group on $[n]$. Let $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$ be a partition of $[n]$ into k orbits of G. Let f be a (possibly partial) Boolean function with $\mathrm{Dom}(f) \subseteq \Sigma^k$ for some alphabet Σ. Then there exists a partial Boolean function g that is symmetric under G such that $Q(g) = Q(f)$ and $R(g) = R(f)$.*

Proof. Define the promise for g to consist of the strings x such that for all $i \in [n]$, $x_i = y_j$, where $i \in O_j$ and $y \in \mathrm{Dom}(f)$. Naturally, in this case we define $g(x) = f(y)$. Then this problem is trivially symmetric under G, and moreover these two problems have the same (quantum or randomized) query complexity because g is just f with some inputs possibly repeated. ∎

By choosing f to be some query problem with $Q(f) = O(1)$ but $R(f) = \omega(1)$ (e.g., Forrelation [2]), one can construct a superpolynomial speedup out of any collection of permutation groups with $\omega(1)$ orbits.

Next, we observe that if G restricted to any orbit is consistent with superpolynomial speedups, then so is G as a whole.

PROPOSITION 5.8. *Let G be a permutation group on $[n]$. Let $\mathcal{O}=\{O_1,O_2,\ldots,O_k\}$ be a partition of $[n]$ into k orbits of G, and let $G|_{O_i}$ denote the restriction of G acting only on O_i. Let f be a (possibly partial) Boolean function that is symmetric under $G|_{O_i}$ with $\mathrm{Dom}(f) \subseteq \Sigma^{|O_i|}$ for some alphabet Σ and orbit i. Then there exists a partial Boolean function g that is symmetric under G such that $Q(g) = Q(f)$ and $R(g) = R(f)$.*

Proof. Define the promise for g to consist of the strings x that agree with some string $y \in \mathrm{Dom}(f)$ on O_i and are some constant symbol elsewhere. Naturally, in this case we define $g(x) = f(y)$. Then this problem is trivially symmetric under G, and moreover these two problems have the same (quantum or randomized) query complexity because g is just f with extra inputs that do not affect the output. ∎

Thus, a collection of permutation groups that does not allow any superpolynomial quantum speedups must remain so when restricted to any subset of its orbits. For this reason, the main task is to understand which *transitive* permutation groups (i.e., groups with a single orbit) allow superpolynomial quantum speedups. The following well-known embedding theorem shows that impritive transitive groups can be expressed as subgroups of a wreath product constructed from a nontrivial block system. See Theorem II.49 of [37] for a proof.

LEMMA 5.9 (embedding theorem). *Let G be a transitive imprimitive permutation group on domain D with $\mathcal{B}=\{B_1,B_2,\ldots,B_k\}$ a nontrivial block system for G. Let H denote the permutation group acting on $[k]$ induced by the action of G on blocks $\mathcal{B}$. Let $\mathrm{Stab}_G(\{B_1\})|_{B_1}$ denote the permutation group induced by the action of $\mathrm{Stab}_G(\{B_1\})$ restricted to the orbit B_1. Then the action of G is permutation isomorphic to a subgroup of $H \wr \mathrm{Stab}_G(\{B_1\})|_{B_1}$ in imprimitive action, meaning that G can be viewed as a subgroup under an appropriate relabeling that identifies D with $[k] \times B_1$.*

Applying Lemma 5.9 recursively, one can always decompose a transitive imprimitive permutation group as a subgroup of an iterated wreath product of primitive groups. Thus, primitive groups are sometimes described as the building blocks of permutation groups. Note that such a decomposition is not unique in general, because a permutation group can have many nontrivial block systems.

The next proposition shows that if any terms in such a wreath product decomposition are consistent with superpolynomial speedups, then so is the group as a whole.

PROPOSITION 5.10. *Let G be a permutation group on $[n]$ that is a subgroup of an iterated wreath product $G_1 \wr G_2 \wr \cdots \wr G_d$ in imprimitive action. Suppose f is a (possibly partial) function symmetric under G_i for some i. Then there exists a partial function g symmetric under G with $Q(g) = Q(f)$ and $R(g) = R(f)$.*

Proof. Suppose G_i acts on n_i points, so that $n = \prod_{i=1}^{d} n_i$. It suffices to exhibit such a function when G is the full wreath product, and each G_j is the symmetric group S_{n_j} for every $j \neq i$. Define the function $\textsc{Triv}_m$ to be a promise problem that only takes two inputs, 0^m and 1^m, outputting 0 on 0^m and 1 on 1^m. The block composition $\textsc{Triv}_{n_1 n_2 \cdots n_{i-1}} \circ f \circ \textsc{Triv}_{n_{i+1} n_{i+2} \cdots n_d}$ is symmetric under G and has quantum and randomized query complexities identical to those of f. ∎

Less trivially, if a wreath product decomposition of a group is sufficiently deep, then the group is also consistent with large speedups. The following construction is based on the so-called 1-FAULT DIRECT TREES problem of [57, 41].

THEOREM 5.11. *Let G be a permutation group on $[n]$ that is a subgroup of an iterated wreath product of nontrivial permutation groups $G_1 \wr G_2 \wr \cdots \wr G_d$ in imprimitive action. Then there exists a function f symmetric under G such that $Q(f) = O(1)$ but $R(f) = \Omega(\log d)$.*

Proof sketch. Suppose G_i acts on $n_i \geq 2$ points, so that $n = \prod_{i=1}^{d} n_i$. It suffices to show that there exists a superpolynomial speedup when G is the full wreath product and each G_i is the symmetric group S_{n_i}. In this case, G can be described as the group of symmetries of a depth-d rooted tree, where all of the vertices at distance $i-1$ from the root have exactly n_i children. G is the action of this group of symmetries on the leaves of the tree. We construct a problem symmetric under G by taking a Boolean evaluation tree and adding a promise that gives an exponential speedup.

If $n_i = 2$ for all i, such a problem is already known: the 1-FAULT DIRECT TREES problem is a promise problem symmetric under G with quantum query complexity $O(1)$ [41] and randomized query complexity $\Omega(\log d)$ [57]. One version of the problem involves evaluating a Boolean formula that is a depth-d binary tree of NAND gates on a 2^d-bit input, subject to a certain promise on the input that enables a fast quantum query algorithm.

We define a more general version of the 1-FAULT DIRECT TREES problem (a formal definition can be found in Appendix B) where the Boolean functions in the formula can have different fan-in at each depth. Specifically, we take the tree to be a block composition of functions $f_1 \circ f_2 \circ \cdots \circ f_d$. The function f_i at depth i is defined by $f_i : S_i \to \{0,1\}$, where $S_i \subseteq \{0,1\}^{n_i}$ consists of the n_i-bit strings that have Hamming weight in $\{0, \lfloor n_i/2 \rfloor, \lceil n_i/2 \rceil, n_i\}$, and $f_i(x) = 0$ if and only if $x = 1^{n_i}$. Note that f_i is essentially just a padded version of the binary NAND function.

We defer to Appendix B for our full definition of 1-FAULT DIRECT TREES and a proof that it satisfies the desired query complexity bounds. We note that the proof follows almost immediately from the proofs in [41, 57], with minor modifications. ∎

We remark that Theorem 5.11 shows the existence of superpolynomial speedups under permutation groups with large base. Indeed, if we define $G = \underbrace{S_2 \wr S_2 \wr \cdots \wr S_2}_{d \text{ times}}$ to be the depth-d iterated wreath product in imprimitive action of the symmetric group S_2, then $b(G) = 2^{d-1}$ even though G acts on 2^d points.

Putting all of these together, we can say the following about collections of permutation groups that are inconsistent with superpolynomial speedups.

COROLLARY 5.12. *Let $\mathcal{G}$ be a collection of permutation groups. Suppose that for every $f \in F(\mathcal{G})$, we have $R(f) = Q(f)^{O(1)}$. Then the following must hold:*

(i) *Permutation groups in $\mathcal{G}$ have $O(1)$ orbits.*
(ii) *The collection of restrictions of groups $G \in \mathcal{G}$ to subsets of their orbits is also inconsistent with superpolynomial quantum speedups.*
(iii) *All wreath product decompositions of groups $G \in \mathcal{G}$ restricted to an orbit have $O(1)$ primitive factors.*
(iv) *The collection of primitive factors that appear in such wreath product decompositions is well-shuffling. Equivalently, by Corollary 5.5, those primitive factors G that act on $[n]$ have $b(G) = n^{\Omega(1)}$, and all but finitely many satisfy case* (i) *of Theorem 5.2 with $\ell p = O(1)$.*

Thus, we have formally shown that collections of permutation groups that do not allow superpolynomial quantum speedups must be constructed out of a constant number of well-shuffling primitive groups. As we have shown before, such primitive

groups are essentially just extensions of permutation groups that correspond to p-uniform hypergraph symmetries.

6. Exponential speedup for adjacency list graph property testing. Having shown that graph properties in the adjacency *matrix* model cannot have exponential quantum speedup, we now turn to the adjacency *list* model. Specifically, we describe a graph property testing problem in the adjacency list model for which there is an exponential quantum speedup. We work with graphs that are undirected and have degree at most 5, while for convenience[7] we allow the graphs to have self-loops and parallel edges (we count these with multiplicity toward the degree bound).

Recall that in property testing, the goal is to distinguish between some set of yes instances and the set of no instances consisting of all graphs that are ε-far from the yes instances. More precisely we say that a graph $G = (V, E)$ (with maximum degree at most 5) is ε-far from the property $\mathcal{P}$ if for any yes instance $G' = (V, E') \in \mathcal{P}$ with n vertices, the symmetric difference of the edge sets has size $|E \triangle E'| \geq \varepsilon n$, where we count edges with multiplicity. Since we work with graph properties, we also require that $\mathcal{P}$ is invariant under permuting the vertices.

The graph property $\mathcal{P}_k$ that we want to test is the following. A graph has the property $\mathcal{P}_k$ if its single-edges form a graph that contains $2j(2^k - 1)$ disconnected instances of "candy" graphs for some $j \in \mathbb{N}$, as shown in Figure 1. A candy graph is obtained from four binary trees of depth k: two "antenna" trees A_1, A_2 and two

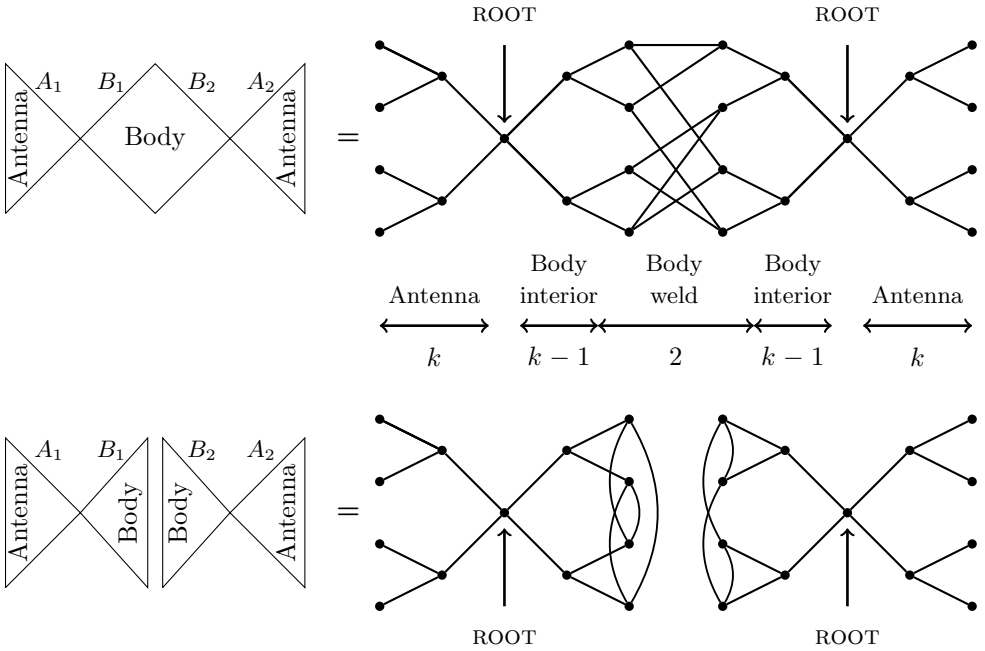

FIG. 1. *An illustration of a candy graph in the yes instance (top right) and our shorthand symbol for it (top left), and a double-bow-tie graph which appears in the particular no instances considered for our classical lower bound proof (bottom right) and our shorthand symbol for it (bottom left).*

[7]This is simply for clarity of presentation; we use self-loops and double edges as markers, for nodes and edges, respectively. However, one could also work with simple graphs only and get the same exponential separation. This can be achieved by using slightly more complicated marker structures, e.g., one could replace self-loops with an edge connected to a triangle, etc.

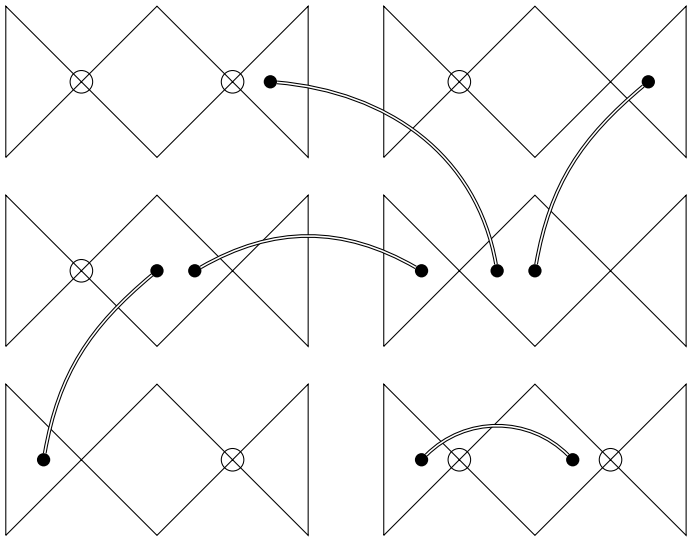

FIG. 2. *Multiple candy (sub)graphs which together form a yes-instance graph with property $\mathcal{P}_k$. In the particular no instances we consider, the candy graphs are replaced by double-bow-tie graphs. Here, we only show six of them and only five of the many "advice edges" (indicated by double lines) that connect each body vertex to a distinct antenna vertex. The circles in the figure represent self-loops at the roots of the candy graphs, which provide advice about whether a body vertex is in the interior or the weld. An even parity of circles indicates interior, while an odd parity indicates weld.*

"body" trees B_1, B_2. The candy graph is formed by connecting the leaves of B_1 and B_2 by a collection[8] of alternating cycles containing all leaves of the body trees, then merging the root of A_1 with the root of B_1 and the root of A_2 with the root of B_2. Additionally, in the full graph, $j2^k$ of the candy (sub)graphs get a self-loop on exactly one of the roots, $j(2^k-2)/2$ of the candy graphs get no self-loops, and $j(2^k-2)/2$ of the candy graphs get self-loops on both of the roots.

There is also a double edge attached to every nonroot vertex of the candy graphs, namely, every body vertex is connected to a distinct antenna vertex by a double edge in the following way: if the body vertex is a "weld" vertex (i.e., a leaf of a body tree), then it gets attached to an antenna vertex inside a candy graph that has exactly one self-loop, while every body interior vertex (i.e., nonweld and nonroot) gets attached to an antenna vertex in a candy graph where the number of self-loops is either zero or two.[9]

While we do not claim that testing the above graph property is particularly useful, we can prove that a quantum computer has an exponential advantage in testing this property. The intuition is the following: this graph property is exponentially difficult to test on a classical computer because the welded trees locally look like exponentially deep trees, unless the weld vertices can be distinguished. This is where quantum computers get their advantage: a quantum computer can "read" the advice hidden in

[8] In the lower bound proof we assume that there is a single long cycle, but this is hard to test, so for the definition of the property we allow smaller cycles as well.

[9] The arrangement of double edges is valid because (i) the total number of body weld vertices in all candy graphs and the total number of antenna vertices in candy graphs with exactly one self-loop are the same—both equal $2 \cdot 2^k \cdot 2j(2^k-1)$; (ii) the total number of body interior vertices in all candy graphs and the total number of antenna vertices in candy graphs with zero or two self-loops are the same—both equal $2 \cdot (2^k-2) \cdot 2j(2^k-1)$.

the structure of the double edges very efficiently using the quantum walk algorithm of Childs et al. [25]. Once we can tell apart the weld vertices from the other vertices, testing the structure could be performed even on a classical computer.

We prove the following theorem in this section.

THEOREM 6.1. *In the adjacency list model, there exists a constant ϵ such that testing whether a bounded-degree graph has property $\mathcal{P}_k$ or is ϵ-far from having it can be done by a quantum algorithm using $Q(\mathcal{P}_k) = \mathrm{poly}(k)$ queries, with perfect completeness and constant soundness. On the other hand, any classical randomized algorithm that can test the property with bounded two-sided error needs $R(\mathcal{P}_k) = \exp(\Omega(k))$ queries.*

Proof outline. First we prove in subsection 6.1 that the property $\mathcal{P}_k$ can be tested with perfect completeness and constant success probability with $\mathrm{poly}(k)$ queries, given the proper advice. Then in subsection 6.2 we show that for graphs having property $\mathcal{P}_k$ the advice can be computed on a quantum computer with probability 1. This completes the proof of our efficient quantum tester.

Finally, in subsection 6.3 we show that there are two particular distributions of yes and no instances that are exponentially difficult to distinguish on a classical computer, where the considered no instances are constant-far from having property $\mathcal{P}_k$ with high probability. This rules out any subexponential-time classical property tester of $\mathcal{P}_k$ that succeeds with constant probability. ∎

6.1. Efficient classical testability with advice. In this subsection, we describe how to test the property $\mathcal{P}_k$ efficiently with advice. First, we rigorously define this notion, which has a similar flavor to the complexity classes NP and MA. Then, we proceed by showing an efficient tester with advice.

The required advice is precisely what our quantum algorithm can provide as per the above subsection: marking the weld vertices. Formally, the advice is a bit associated with every vertex, and for yes instances the advice is supposed to mark weld vertices.

There are two major parts of our analysis. First, we show that the test always accepts yes instances if the advice is correctly provided. Then we show that if the test accepts a graph with high probability, then the graph must be close to a yes instance. The latter implies that no instances that are far from complying with the property must be rejected with high probability.

6.1.1. Property testing with advice. We say that a property $\mathcal{P}$ can be tested with advice[10] in Q queries if there is a tester $\mathcal{T}$ that is

Complete. For every instance $x \in \mathcal{P}$ there is an advice string $w \in \{0,1\}^*$, such that $\mathcal{T}$ accepts (x, w) with probability at least $2/3$ and makes at most Q queries to x and w.

Sound. For every instance x that is at least ε-far from $\mathcal{P}$, and for any advice string $w \in \{0,1\}^*$, $\mathcal{T}$ accepts (x, w) with probability at most $1/3$.

Additionally, if for every yes instance there is an advice string that makes the tester accept with certainty, we say that the tester has perfect completeness.

6.1.2. Testing the binary trees, the weld, and the advice. In this subsection, we describe our classical tester using the advice. We build increasingly more complex tests, such that passing them enforces more and more structure on the graph.

[10] Note that this terminology differs from the standard use of "advice" in complexity theory, which typically refers to a string that can only depend on the input length.

Our final tester ensures that any graph passing the test with high probability must be ε-close to a yes instance. The query and time complexity of our final tester is $\mathrm{poly}(k/\varepsilon)$.

As a first step we test the binary tree structures that are supposed to remain after removing every double edge, self-loop, and edge between marked (therefore supposedly weld) vertices. Therefore, while testing the binary tree structures—i.e., in Algorithms 6.1 and 6.2 and the consistency check below—we ignore those edges (whenever we say that we ignore some edges we also exclude them from degree counts). Also, if we encounter any vertex that has more than 4 single-edge neighbors (excluding self-loops), then we immediately reject the graph, so we can also assume without loss of generality that the modified graph has degree at most 4 in Algorithms 6.1 and 6.2. Similarly, if Algorithm 6.1 or 6.2 returns REJECT, we immediately reject the instance.

The basic primitive of our tester is a binary tree tester described in Algorithm 6.1, based on nonbacktracking walks. The main idea of the tester is that for any nonroot vertex v at level ℓ (i.e., distance ℓ from the root) in a binary tree of depth k, any nonbacktracking walk starting toward the parent of v can continue for at least $k-\ell+2$ steps, while any other nonbacktracking walk must terminate after $k-\ell$ steps upon reaching a leaf, as illustrated in Figure 3. Based on this observation we can find the parent of v efficiently. Algorithm 6.1 summarizes this procedure for finding the parent vertex and adds some consistency checks that are helpful for catching potential corruptions of the structure.

Algorithm 6.2 recursively applies Algorithm 6.1 to find a path to the root. Although this recursive structure may not be the most efficient way of finding such a root-path, it allows us to argue that a root-path found for a vertex should match the root-path of its parent, even in slightly corrupted trees.

Consistency test. During this test we ignore every double edge and self-loop and also any edge between marked vertices. Given vertex v, reject if v is marked and has degree greater than 1. Run Algorithm 6.2 to find a path to the root (degree-4 vertex). If the root-path is longer than k, or any vertex (other than v) is marked on the root-path, then reject. Also reject if v is marked and the path is shorter than k. Repeat this procedure 100 times and reject unless always the same path is found. If this consistency check or any individual run fails, then reject; otherwise accept.

DEFINITION 6.2. *A vertex v is* consistent *if it passes the consistency test with probability at least* $1/2$. *For a consistent vertex, there is a* consistent root *and a* consistent root-path *to the consistent root that Algorithm* 6.2 *finds with probability at least* 0.99.

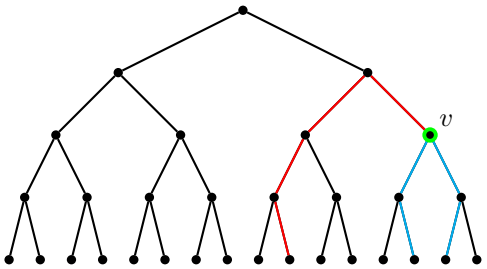

FIG. 3. *A binary tree of height* 4 *with a vertex v at level* 2 *highlighted in green. The two blue paths setting out downward reach a leaf already after* 2 *steps, while the red path setting out in the direction of the root needs to traverse* 4 *edges before reaching a leaf.*

Algorithm 6.1. FindParent(v).

input: vertex v (in a graph with maximum degree at most 4)
output: parent vertex u (if the graph is a binary tree of depth k)

1: **if** $\deg(v) = 4$ **then return** ROOT ($\star$ only the root should have degree 4 $\star$)
2: **if** $\deg(v) \notin \{1,3,4\}$ **then** REJECT ($\star$ nonroots can only have 1 or 3 neighbors $\star$)
3: $r \leftarrow \emptyset$ ($\star$ stores a root candidate that is any degree-4 vertex encountered $\star$)
4: **for each** $w \in \mathrm{adj}(v)$ **do** ($\star$ if we get here we know that the degree of v is 1 or 3 $\star$)
5: $p_0^{(w)} \leftarrow v$ ($\star$ the starting vertex of the nonbacktracking walk $\star$)
6: $p_1^{(w)} \leftarrow w$ ($\star$ the first vertex on the path in the direction toward w $\star$)
7: $\ell^{(w)} \leftarrow 2k$ ($\star$ upper bound on the length of a nonbacktracking walk $\star$)
8: **for each** $t = 1, \ldots, 2k-1$ **do** ($\star$ the steps of the nonbacktracking walk $\star$)
9: **if** $\deg(p_t^{(w)}) = 1$ **then** ($\star$ a leaf is found $\star$)
10: $\ell^{(w)} \leftarrow t$ ($\star$ store the length of the path $\star$)
11: **quit loop and goto line** 17 ($\star$ quit the loop and end the walk $\star$)
12: **if** $\deg(p_t^{(w)}) = 4$ **then** ($\star$ a root candidate is found $\star$)
13: **if** $r = \emptyset$ **then** $r \leftarrow p_t^{(w)}$ ($\star$ update root candidate $\star$)
14: **else** REJECT ($\star$ there is a unique root $\star$)
15: $p_{t+1}^{(w)} \leftarrow$ i.i.d. uniformly random vertex in $\mathrm{adj}(p_t^{(w)}) \setminus \{p_{t-1}^{(w)}\}$ ($\star$ random step $\star$)
16: **end for**
17: **if** $\deg(p_{\ell^{(w)}}^{(w)}) > 1$ **then** REJECT ($\star$ verify that a leaf is found $\star$)
18: **end for**
19: $u \leftarrow \mathrm{argmax}_w(\ell^{(w)})$ ($\star$ the longest path must go through the parent $\star$)
20: **if** $\exists w_1, w_2 \in \mathrm{adj}(v) \setminus \{u\}$ such that $\ell^{(w_1)} \neq \ell^{(w_2)}$ **then** REJECT ($\star$ consistency- $\star$)
21: **if** $r \neq \emptyset \wedge r \notin \{p_1^{(u)}, p_2^{(u)}, \ldots, p_{\ell^u}^{(u)}\}$ **then** REJECT ($\star$ checks $\star$)
22: **return** u

Algorithm 6.2. FindRootPath(v).

input: vertex v (in a graph with maximum degree at most 4)
output: a path to the root (if the graph is a binary tree of depth k)

1: $p_0 \leftarrow v$ ($\star$ v is the first vertex on the path to the root $\star$)
2: $\ell \leftarrow k$ ($\star$ the path to the root cannot be longer than k $\star$)
3: **for each** $t = 1, \ldots, k$ **do** ($\star$ find a path to the root step-by-step $\star$)
4: $p_t \leftarrow \mathrm{FindParent}(p_{t-1})$ ($\star$ find the parent of the current vertex $\star$)
5: **if** $p_t =$ ROOT **then** ($\star$ the root is found $\star$)
6: $\ell \leftarrow t$ ($\star$ store the length of the path $\star$)
7: **quit loop and goto** 9 ($\star$ store the length of the path and stop $\star$)
8: **end for**
9: **if** $\deg(p_\ell) \neq 4$ **then** REJECT ($\star$ verify that a root is found $\star$)
10: **return** $(p_0, p_1, \ldots, p_\ell)$

LEMMA 6.3. *For any edge e between two neighboring consistent vertices u and v, one of the consistent root-paths, say, $(u, p_1, \ldots, p_\ell)$, must be a concatenation of the edge e and the other consistent root-path, i.e., $p_1 = v$.*

Proof. First note that due to line 2 of Algorithm 6.1, any consistent vertex u, v must have degree 1, 3, or 4. If v has degree 4, then for any neighbor u the consistent

root-path must be (u, v) due to line 21. Also note that the statement is trivial if either u or v has degree 1.

Finally, we treat the case when both u and v have degree 3. Toward a contradiction let us assume that the statement of the lemma does not hold. Let $v_c \neq u$ denote the neighbor of v which is not its immediate parent on the consistent root-path, and similarly let $u_c \neq v$ denote the neighbor of u which is not its immediate parent on the consistent root-path. Due to line 20, a nonbacktracking walk from u in the direction u_c has typically (with probability at least 0.99) the same length as a nonbacktracking walk in the direction v. This also means that there is a fixed length that has high probability (> 0.98), which we denote by ℓ_u. We denote by ℓ_v the analogous typical length starting from v toward either v_c or u. When we start a nonbacktracking walk in the direction of u we continue with a nonbacktracking walk toward u_c with probability at least $1/3$. This implies that $\ell_v > \ell_u$. Since the situation is symmetric we can also deduce that $\ell_v < \ell_u$, which is a contradiction. ∎

It is easy to see that for any consistent vertex v all vertices on its consistent root-path are also consistent vertices. This is due to the recursive structure of Algorithm 6.2.

Consequently we can conclude that the graph G_C induced by consistent vertices is a (sub)binary forest. More precisely, every connected component can be injectively mapped into a graph obtained by merging the roots of two binary trees of depth k. The mapping is defined via the root-paths and also ensures that marked consistent vertices get mapped to leaves. Moreover, for any $\delta > 0$, if the consistency test is passed with probability $1 - \delta$ for a uniformly random vertex $v \in V$, then there are at least $(1 - 2\delta)|V|$ consistent vertices, since by definition inconsistent vertices fail to pass the consistency check with probability at least $1/2$.

Weld-consistency test. During this test we ignore every double edge and self-loop (additionally, when we call the consistency test as a subroutine, we also ignore edges between marked vertices). Run the consistency test on v and denote by r the root found. If the length of the root-path is less than k (i.e., v is not a "leaf"), then accept. Let e denote the last edge of the root-path from v to r.

- If v is not marked, then go to the root r and perform 100 nonbacktracking random walks of length k along e. If any walk terminates before k steps or any marked vertex is found, then reject; otherwise accept.
- If v is marked, then reject unless exactly 2 of its neighbors are also marked. Run the consistency test on both marked neighbors and denote the roots r' and r''. Reject if $r' \neq r''$ or $r = r'$. Perform a nonbacktracking random walk from r along all 4 edges of length k; if any walk stops before k steps or not exactly 2 out of the 4 end-points are marked, then reject. Also reject if the walk along e does not end in a marked vertex. Reject unless both marked end-points have exactly two marked neighbors. For each marked end-point, randomly choose a marked neighbor, run a consistency test on the marked neighbor, and report its root. If either consistency test fails or either of the found roots does not match r', then reject. Repeat the process of this paragraph 100 times.

DEFINITION 6.4. *A vertex v is* weld-consistent *if it passes the above test with probability at least* $1/2$.

We already established that consistent vertices form a (sub)binary forest, where some leaves at level k may be marked. The weld-consistency test might remove

some of the marked vertices but ensures that the (sub)binary trees are paired up via marked-marked vertex edges.

LEMMA 6.5. *All weld-consistent marked vertices in a (sub)binary tree t_1 are connected to at most 2 consistent marked vertices in another (sub-)binary tree t_2 of consistent vertices.*

Proof. Indeed, for any weld-consistent vertex v, its consistent root r has the property that a nonbacktracking walk of length k along the 4 outgoing edges after traversing the 2 marked leaves finds the same root r' with high probability. We call r' the *consistent-root-pair* of r. It is easy to see that all consistent marked neighbors of v must have the same consistent root as the consistent-root-pair of r. ∎

By Lemma 6.5 we have established that weld-consistent vertices form pairs of (sub)binary trees that are connected by edges between marked vertices at level k, with each marked vertex having at most 2 marked neighbors. Moreover, these (sub)binary candy graphs are isomorphic to an induced subgraph of an actual candy graph. Also at any root, at most two of the branches contain marked leaves. We call such a graph a (sub)binary candy ensemble.

Similarly as before, for any $\delta > 0$, if the weld-consistency test is passed with probability $1-\delta$ for a uniformly random vertex $v \in V$, then there are at least $(1-2\delta)|V|$ weld-consistent vertices. This way we can ensure that there are many weld-consistent vertices and that they form a (sub)binary candy ensemble. By the following test we also ensure that a typical connected component is close to being a complete candy graph.

Completeness test. During this test we still ignore every double edge and self-loop. Given vertex v, run the weld-consistency test and then go to the root r. From r, along each edge perform a nonbacktracking walk. Reject if any walk fails to find a leaf (degree-1 vertex) or a marked vertex (with 2 marked neighbors) after exactly k steps or any non-degree-3 vertex is encountered other than a leaf. Also reject if there are not exactly 2 marked vertices among the 4 reached end-points. Traverse to a random marked neighbor of a randomly chosen marked end-point, find its root-path, and let r' denote the root that is found. Also run the weld-consistency check for every vertex on every sampled path and reject if any of the vertices fail the weld-consistency check. Go to r' and run the same test, except for traversing a marked-marked edge at the end. Repeat the whole process described in this paragraph $\Theta(k/\varepsilon)$ times.

LEMMA 6.6. *If v is a weld-consistent vertex within a (sub)binary candy graph induced by fewer than $(1-\varepsilon)(2^{k+3}-6)$ weld-consistent vertices, then the completeness test rejects with high probability.*

Proof. Let us consider the connected component of v formed by weld-consistent vertices. We already established that this connected component can be embedded into a candy graph. If at least an ε-fraction of the vertices of the candy graph are missing from the connected component of v, then we must encounter a nonweld-consistent vertex or a degree-deficient vertex (i.e., less than degree-3 interior or weld vertex) with probability $\Omega(\varepsilon/k)$ in each of the $\Theta(k/\varepsilon)$ runs (since by the pigeonhole principle there is a level set of the candy graph, where at least an $\Omega(\varepsilon/k)$-fraction of the vertices are missing). This implies the statement of the lemma. ∎

Passing the weld-consistency test verifies that most vertices are weld-consistent. However, a large deviation form the property could still occur if the weld-consistent vertices did not form (near) complete candy graphs. The completeness test is designed to rule out this possibility. Indeed, if there are more than $\varepsilon|V|$ weld-consistent vertices

whose connected components have size at most $(1-\varepsilon)(2^{k+1}-1)$, then the completeness check fails with probability at least ε as shown by the above lemma. Therefore, one can see that if the graph passes the completeness check with probability at least $1-\varepsilon$, then the induced subgraph of weld-consistent vertices is $O(\varepsilon)$-close to the desired structure of having a disjoint union of candy graphs, when disregarding self-loops and double edges. (One can actually form the desired structure by removing inconsistent vertices and then completing the potentially incomplete trees missing $O(\varepsilon)$-fraction edges.)

Advice test. Given vertex v, run the completeness test and reject unless it has precisely one double edge or has 4 single-edge neighbors, with the potential addition of a self-loop (root vertex). If v has a double-edge neighbor u, then run the completeness test on u. Reject unless precisely one of u and v appear to be in a branch with nonmarked leaves in the completeness test. Let a denote the one in the nonmarked branch, and b the other. Check the parity of the number of loops of a's root r and the root-pair r' of r found during the completeness test. Reject if the parity does not match the marking of b.

DEFINITION 6.7. *A vertex v is* advice-consistent *if it passes the advice test with probability at least* $1/2$.

We can see that if the graph passes the advice test with probability at least $1-\varepsilon$, then at most an $O(\varepsilon)$-fraction of the weld-consistent vertices have an incorrect advice edge. Thus, if the number of vertices is a multiple of $2\cdot(2^k-1)\cdot(2^{k+3}-6)$, then we can just take the induced subgraph of the advice-consistent vertices and complete it using $O(\varepsilon)$ edges to get to a yes instance with property $\mathcal{P}_k$.[11]

The final test. Verify that the number of vertices is a multiple of $2\cdot(2^k-1)\cdot(2^{k+3}-6)$, and then run the advice test $O(1/\varepsilon)$ times. If we have a yes instance with the correct advice, then the test always accepts, so the tester has perfect completeness. On the other hand, if the test accepts with probability at least $1/3$, then we know that the graph must be ε-close to a yes instance. Therefore, any ε-far no instance will be rejected with probability at least $2/3$.

6.2. Marking the weld vertices using a quantum computer. In this subsection, we describe how to efficiently mark weld vertices in a graph $G\in\mathcal{P}_k$. More precisely, given a vertex v in G the task is to tell whether it is a weld vertex or not, in query and time complexity $\mathrm{poly}(k)$. If the graph G does not have property $\mathcal{P}_k$ we do not place any restriction on the output, so we will always assume that $G\in\mathcal{P}_k$. The procedure that we describe below has success probability 1.

Let us denote by W the set of weld vertices in a graph G with property $\mathcal{P}_k$. Given a vertex v, we decide whether or not $v\in W$ as follows:

1. Along every nondouble edge adjacent to v, start a nonbacktracking (random) walk, which only traverses single-edges. If a degree-3 vertex (i.e., a leaf with a double edge) is encountered within the first k steps of any of these walks (including v itself), then conclude $v\notin W$.
2. Now we know that v is a nonroot body vertex. We traverse the double edge to the antenna vertex a in antenna A. From now on we completely ignore double edges (and even exclude them from degree counts) and use Algorithm 6.2 to find the root r_a of the antenna A.
3. From the root r_a we run the quantum walk algorithm of Childs et al. [25] to find the other root r_b in the candy graph. We conclude $v\in W$ if exactly one of r_a, r_b has a self-loop; otherwise we conclude $v\notin W$.

[11] Note that one might also need to remove an $O(\varepsilon)$-fraction of the candy-components if there is a surplus of candies with even/odd number of marker loops.

Now we briefly prove the correctness of the above procedure. For the first step, note that the degree-3 vertices are precisely the leaves of the antennas. Moreover, if a nonbacktracking walk is started from an antenna vertex toward the leaves, then it will always find a leaf within k steps. On the other hand, any body vertex is at least $k+1$ steps from an antenna leaf, if only single-edges can be traversed.

The second step relies on the correctness of Algorithm 6.2, which we already analyzed in the previous section.

The third step deserves a bit more explanation. We would like to use the quantum algorithm of Childs et al. [25] on the body part of the candy graph. As we discuss in the next paragraph, for this it suffices to construct an adjacency list "oracle" for a graph, where all vertices have bounded degree (say, 3) and the body welded tree structure containing r_a is one of the connected components. We construct this oracle starting from the input oracle and removing double edges. Then we disconnect the antennae by treating the roots (vertices with at least 4 nondouble edges) in a special way. From a root we start a nonbacktracking (random) walk of length k along each edge. If we find a leaf after k steps, then we know that the initial edge leads to the antenna vertex and therefore we remove the corresponding edge adjacent to the root.[12]

Once we have the adjacency list oracle for the modified graph, we can construct a block-encoding of the modified adjacency matrix A divided by the maximum degree ($= 3$) using standard techniques (see, for example, [31]). Once we have a block-encoding we can also perform Hamiltonian simulation [17, 47] efficiently (or implement polynomials of A [31]), which in turn enables implementing the algorithm of [25]. Since the quantum walk algorithm of [25] finds the other root in the tree with probability $\Omega(1/\operatorname{poly}(k))$, and the success probability can also be computed, we can make the algorithm succeed with probability 1 using amplitude amplification.

Once we have found the other root in the candy graph we can easily compute the parity of the number of self-loops, which in turn correctly identifies the weld vertices, since $G \in \mathcal{P}_k$. As all steps of the above procedure succeed with certainty, we get the following.

LEMMA 6.8. *The above quantum algorithm given any vertex v in a graph $G \in \mathcal{P}_k$ outputs a bit which is* 1 *if and only if v is a weld vertex. Moreover, the algorithm has query and time complexity* $\operatorname{poly}(k)$.

6.3. Classical lower bound. We show that our problem is classically hard by showing that it is hard to distinguish between a graph chosen randomly from $\mathcal{G}_1$—a particular "difficult" subset of yes instances in $\mathcal{P}_k$—and a graph chosen randomly from $\mathcal{G}_2$—a particular "difficult" subset of no instances. $\mathcal{G}_1$ is the set of yes instances containing $2^k - 1$ candy subgraphs, all of which are welded along a single cycle, such that exactly half of the candy subgraphs with an even number of self-loops have no self-loops. $\mathcal{G}_2$ contains the same graphs as $\mathcal{G}_1$ except that each candy graph is modified to take on the shape of a double-bow-tie: the weld vertices within a bow-tie are connected by an arbitrary (single) cycle, and exactly half of the roots have a self-loop.

[12] Strictly speaking this results in a corrupted adjacency list oracle, because we did not delete the edge from the neighbor list of the vertex in the antenna connected to the root. This could be easily fixed with a bit of caution, but we note that the edge also gets automatically deleted if we use the block-encoding framework [31], where each matrix element of a sparse matrix is encoded by a product of two amplitudes, one from the "left" and one from the "right" adjacency lists. Indeed, if either of the two amplitudes is zero (as when an edge is deleted from one list), then the product is also zero.

We prove that classically at least $2^{\Omega(k)}$ queries are needed to distinguish between random instances from $\mathcal{G}_1$ and $\mathcal{G}_2$. With a slight abuse of notation, we also use $\mathcal{G}_i$ to mean a random graph sampled from the set $\mathcal{G}_i$. More precisely, we define the sampling procedures for $\mathcal{G}_i$ as follows:

a. Call a graph a *depth-k root-joined binary tree* if it is obtained by identifying the root vertices of two binary trees of depth k, one of which is called antenna and the other body. Create $4(2^k-1)$ depth-k root-joined binary trees, and arrange them in $2(2^k-1)$ pairs.

b. For the first 2^k pairs mark one of the roots with a self-loop, and for the next $2^{k-1}-1$ pairs mark both roots with a self-loop. (The roots of the remaining $2^{k-1}-1$ pairs are unmarked.)

c. Pick a random bijection between body interior vertices and antenna vertices that are in a pair with zero or two self-loops, and pick another random bijection between body weld vertices and antenna vertices that are in a pair with exactly one self-loop. Connect vertices along the bijections with a double edge.

d. G_1: In each pair, join the body weld vertices in the two root-joined binary trees of the pair by a single random cycle that alternates between vertices in the two trees.

   G_2: In each pair, join the body weld vertices within both root-joined binary trees by a single random cycle.

When an adjacency list oracle is sampled for the above graphs, then vertex labels are randomly permuted, as well as the adjacency lists of every vertex.[13]

First, in subsection 6.3.1, we verify that a random no instance in $\mathcal{G}_2$ is indeed far from the yes instances in expectation. By a *reduced* graph in $\mathcal{G}_i$, we refer to a graph in $\mathcal{G}_i$ such that all of its advice edges (i.e., double edges) and self-loops have been removed (i.e., in the above procedure we ignore steps b and c). Observe that reduced graphs in $\mathcal{G}_1$ are bipartite (simply alternately color each layer in each welded tree). On the other hand, we prove that reduced graphs in $\mathcal{G}_2$ are typically far from even being bipartite.

Second, we argue in subsection 6.3.2 that welded and self-welded trees are hard to distinguish by a classical randomized algorithm that uses only subexponentially queries. More specifically, we argue that such a classical randomized algorithm cannot even find a cycle in either of these two graphs; therefore, the algorithm sees both as an infinite-depth binary tree.

Now, graphs from $\mathcal{G}_1$ and $\mathcal{G}_2$ are formed by welded and self-welded trees, respectively, together with the double edges introduced in step c above. But traversing these double edges most likely leads to completely unqueried welded or self-welded trees, so it is intuitively clear that these double edges will not make distinguishing between $\mathcal{G}_1$ and $\mathcal{G}_2$ any easier than distinguishing between welded and self-welded trees. Therefore, it follows that graphs from $\mathcal{G}_1$ and $\mathcal{G}_2$ are themselves hard to distinguish. We argue this formally in subsection 6.3.3 by defining a fixed simulator $\mathcal{G}_s$ which we show appears like both $\mathcal{G}_1$ and $\mathcal{G}_2$ to a classical randomized algorithm that uses only subexponentially many queries.

6.3.1. A random graph in $\mathcal{G}_2$ is typically far from any yes instance. We show that reduced graphs in $\mathcal{G}_2$ are typically constant-far from reduced graphs in $\mathcal{G}_1$.

[13]Note that the above procedure samples from graph isomorphism classes nonuniformly; indeed, classes with richer automorphism groups tend to be sampled more often.

It is easy to see that this also implies that (nonreduced) $\mathcal{G}_2$ is typically constant-far from $\mathcal{G}_1$.

LEMMA 6.9. *With probability at least* $1-\exp(-\Omega(2^k))$*, we need to remove at least* $2^k \cdot (2^k-1)/16$ *edges from a random reduced graph in* $\mathcal{G}_2$ *to make it bipartite.*

Proof. Let us consider a single bow-tie in a reduced graph in $\mathcal{G}_2$, and in particular its induced subgraph B, spanned by the vertices in the weld layer and the layer adjacent to it, so that the total number of vertices in B is $3 \cdot 2^{k-1}$. We first show the following.

CLAIM 6.10. *With probability at least* $1-\exp(-\Omega(2^k))$*, we need to remove at least* $2^k/64$ *edges from* B *to make it bipartite.*

Proof. This can be argued following the strategy of [33, Lemma 7.4]. The sampling procedure in terms of B can be described as follows: take 2^{k-1} disjoint paths of length 2 (think about each of these as a leaf and its parent), enumerate all paths, and give subsequent labels $1-2, 3-4, \ldots, (2^k-1)-2^k$ to the end-points of the paths, then sample a random permutation π of $[2^k]$, and connect $\pi^{-1}(1)$ to $\pi^{-1}(2)$, $\pi^{-1}(2)$ to $\pi^{-1}(3)$, ..., and $\pi^{-1}(2^k)$ to $\pi^{-1}(1)$. Observe that B can be alternatively sampled by fixing a cycle of length 2^k, and randomly matching its vertices using π as follows: connect with a length-2 path $\pi(1)$ to $\pi(2)$, $\pi(3)$ to $\pi(4)$, ..., and $\pi(2^k-1)$ to $\pi(2^k)$. It is easy to see that for a given π the graphs sampled in the two different ways are isomorphic.

It is more convenient to work with the second sampling procedure. Consider a particular partition (U_1, U_2) of the cycle vertices in B and any partition (V_1, V_2) of all vertices in B that extends (U_1, U_2) in the sense that $U_i \subseteq V_i$. We say an edge violates (V_1, V_2) if it connects two vertices from the same V_i. Now we bound the probability that the partition (U_1, U_2) can be extended such that there are fewer than $2^k/64$ violating edges in B. Let $c := 2^k$ denote the number of cycle vertices, and let $C_{\text{inner}}(U_1, U_2)$ denote the number of cycle edges that either connect two vertices in U_1 or two vertices in U_2. It is easy to see that if $C_{\text{inner}}(U_1, U_2) \geq c/64$, then there can be no extension (V_1, V_2) of (U_1, U_2) featuring fewer than $2^k/64$ violating edges.

Now let us assume without loss of generality that $|U_1| \leq |U_2|$ and observe that $C_{\text{inner}}(U_1, U_2) < c/64$ implies that $|U_1| \geq c/2 - c/128 \geq 3c/8$. Now, every vertex of U_1 that is connected to a vertex in U_2 by a length-2 path results in a violating edge in every extension (V_1, V_2), so we examine the probability that fewer than $c/64$ such length-2 paths occur. Without loss of generality we can assume that we first match vertices of U_1 during the sampling of B. At each step, the probability that a vertex $w \in U_1$ is matched to a vertex in U_2 is at least $1/2$; so the probability that fewer than $c/64$ violating edges are created in total during the first $3c/16$ steps is at most

$$\sum_{i=0}^{c/64-1} \binom{3c/16}{i} \cdot 2^{i-3c/16} \leq 2^{-11c/64} \sum_{i=0}^{c/64} \binom{3c/16}{i} \overset{6.1}{\leq} 2^{\frac{3c}{16} H(16/3/64) - \frac{11c}{64}} \leq 2^{-0.11c},$$

where $H(p) := -p\log(p) - (1-p)\log(1-p)$ is the binary entropy function, and the second inequality follows from [38, Corollary 22.9] stating

$$\forall\, 0 < h \leq n/2 : \sum_{j=0}^{h} \binom{n}{j} \leq 2^{n \cdot H(h/n)}. \tag{6.1}$$

Note that this bounds the probability that *any* partition (V_1, V_2) extending (U_1, U_2) has fewer than $c/64$ violating edges.

We call a partition (V_1, V_2) of all vertices in B "bad" if it has fewer than $c/64$ violating edges. Then by the union bound

$$\begin{aligned}\Pr(\exists \text{ bad partition } (V_1,V_2)) &\le \sum_{(U_1,U_2)} \Pr(\exists \text{ bad partition extending } (U_1,U_2)) \\ &\le \#\{(U_1,U_2) : C_{\text{inner}}(U_1,U_2) < c/64\} \times 2^{-0.11c},\end{aligned} \tag{6.2}$$

where (U_1, U_2) denotes a partition of the cycle edges with $|U_1| \le |U_2|$. For each $i < c/64$, each partition which has i violating cycle edges is determined by the choice of those edges. Therefore,

$$\#\{(U_1,U_2) : C_{\text{inner}}(U_1,U_2) < c/64\} \le \sum_{i=0}^{c/64-1} \binom{c}{i} \overset{6.1}{\le} 2^{0.09c}, \tag{6.3}$$

and thus $\Pr(\exists \text{ bad partition } (V_1,V_2)) \le 2^{-0.02c} = \exp(-\Omega(c)) = \exp(-\Omega(2^k))$ as desired. ∎

This implies the statement of the lemma because there are $4 \cdot (2^k - 1)$ copies of B in any graph in $\mathcal{G}_2$. If all of them are far from bipartite, in the sense of needing to remove at least $2^k/64$ edges, then the total number of edges that need to be removed is at least $2^k \cdot (2^k - 1)/16$. On the other hand, the probability of any of them not being far from bipartite is at most $4 \cdot (2^k - 1) \cdot \exp(-\Omega(2^k)) = \exp(-\Omega(2^k))$. ∎

6.3.2. Hardness of distinguishing welded and self-welded trees. Let $\mathcal{B}$ be a classical randomized algorithm. We consider the difficulty of $\mathcal{B}$ winning four different games by querying an oracle that provides black-box access to either a random welded or self-welded tree. To be clear, in this subsection, by a random "self-welded tree" we mean two binary trees of depth k where the weld vertices in each are connected to themselves by a single uniformly random cycle, and by a random "welded tree" we mean two binary trees of depth k where the weld vertices in one are connected to weld vertices in the other by a uniformly random single alternating cycle; in both cases vertex labels are assigned uniformly at random. A self-welded tree looks like a double-bow-tie graph without the antenna, and a welded tree looks like a candy graph without the antenna (cf. Figure 1).

The difficulty of winning can be quantified by bounding the winning probability by a function of t, the number of queries that $\mathcal{B}$ makes. Throughout, the winning probability is over three or four sources of randomness: the internal randomness of $\mathcal{B}$, the cycle forming the (self-)weld, the vertex labelings, and also the random choice of the starting vertex in two of the games. These four games are described in the four lemmas below. They relate to the difficulty of distinguishing welded from self-welded trees because, unless they are won, the welded and self-welded trees both look just like a large (effectively infinite) binary tree to $\mathcal{B}$.

We can, without loss of generality, assume that $\mathcal{B}$ does not query labels which have already been queried and that each query to a vertex label returns the labels of *all* neighbors of that vertex. For notational convenience, we shall use the concept of the "knowledge graph" of $\mathcal{B}$, as introduced in [33, section 7]. The knowledge graph of $\mathcal{B}$ is the graph that it sees, which consists of all vertices whose labels it has either queried or has been returned by the oracle in answer to those queries, as well as the edges between them. The knowledge graph can change every time $\mathcal{B}$ makes a query. In the following four lemmas, we make the further assumption that $\mathcal{B}$ can only query labels in its knowledge graph. We will only use these lemmas under this

assumption. (Later arguments, in particular, Lemma 6.15, show that querying outside the knowledge graph is essentially pointless.)

LEMMA 6.11. *Let Game* A *be that of finding a cycle in a random self-welded tree given the label of* ROOT*. If $\mathcal{B}$ uses t queries, then its probability of winning Game* A *is at most $O(t^2 \cdot 2^{-k})$.*

Proof. As explained in [25], the winning probability can be expressed as the probability that a random embedding π of a random rooted binary tree T, with t vertices[14] and root at ROOT, into a random self-welded tree G contains a cycle, where the randomness in the embedding corresponds to the random vertex labelings, and the randomness in the rooted binary tree corresponds to the internal randomness of $\mathcal{B}$. The terms "random embedding" and "rooted binary tree" are defined in [25, section IV].

Therefore, it suffices to show that for an arbitrary fixed T, the probability that its random embedding $\pi(T)$ in a random G contains a cycle is at most $O(t^2 \cdot 2^{-k})$. We denote this probability by $\mathrm{E}_G[P^G(T)]$ to match the notation of [25]. Our proof is similar to that of [25, Lemma 10],[15] but gives a tighter bound.

The probability that $\pi(T)$ contains a cycle is the same as the probability that there exist vertices $a \neq b$ in T such that $\pi(a) = \pi(b)$. If there is a cycle in $\pi(T)$, then there is also a minimal rooted tree $T' \subseteq T$ such that $\pi(T')$ contains a cycle. If a and b are in such a minimal T', then it follows that no two other vertices in $T' = P_a \cup P_b$ get mapped to the same vertex in G, where P_a and P_b are the paths in T from the root to a and b, respectively. Also, if $T' = P_a \cup P_b$ is minimal, then $\pi((P_a \setminus P_b) \cup (P_b \setminus P_a))$ is a cycle in G and therefore it must contain a weld vertex. Let $C_{(a,b)}$ be the event that there is an edge $e \in P_a \setminus P_b$ that is mapped to a weld edge in G, and moreover, $\pi(a) = \pi(b)$ and no other two vertices lying on $P_a \cup P_b$ get mapped to the same vertex in G. Due to the above considerations, the existence of a cycle in $\pi(T)$ coincides with the event $\cup_{a,b \in T} C_{(a,b)}$.

Now it suffices to bound the probability of each event $C_{(a,b)}$ and take the union bound. Consider sampling the graph G and the embedding π in sync with the exploration of T. We start with only a random label assigned to ROOT, and no predetermined weld. When we explore the neighbors of a vertex, we sample random neighbors in G along any yet unexplored weld edges, then assign a uniformly random label for all yet unseen neighbors, and finally choose a random neighbor for the embedding π. It is easy to see that the resulting embedding and vertex labels are uniformly random, but we should also ensure that we faithfully sample from uniformly random weld cycles in G. During the process we will have certain weld path segments already explored and should sample a neighbor according to the uniform weld cycle distribution conditioned on the already explored weld path segments. To do so we sample a uniformly random weld path segment (i.e., a connected component of explored weld edges) and connect a random end-point of it to the vertex where we wish to continue the exploration of the weld cycle.

We start the exploration following the path P_b, then we continue with the path $P_a \setminus P_b$. Let $C^{\ell}_{(a,b)}$ be the event that the closest edge $e \in P_a \setminus P_b$ to a such that $\pi(e)$ is in the weld has distance ℓ from a. We now give a bound on the probability of $C^{\ell}_{(a,b)}$. During the exploration of $P_a \setminus P_b$, once we get to the edge e there are still

[14] Strictly speaking, T has $O(t)$ vertices as we are assuming the oracle answers each query to a vertex label with all labels of its neighbors. But, because the lemma is only up to big-O, this is unimportant.

[15] That is, [25, Lemma 8] in the `arXiv` version.

at least $2^k - 2t$ weld path segments (many of which are isolated leaf vertices) from which we pick one uniformly at random. Since e shall be the last weld edge, the rest of the path should be entirely within the binary tree part of G excluding the weld edges. Suppose that $\pi(b)$ is at depth $k-d$ of the binary tree. Then it is easy to see that the number of leaves in G reachable by a length ℓ (nonbacktracking) path that avoids weld edges is 0 if $\ell - d$ is odd or $\ell < d$, 2^d if $\ell = d$, and otherwise $2^{(d+\ell-2)/2}$ (cf. Figure 3). Note that any such $\pi(b)$ to leaf weld avoiding path is unique and the probability that we follow that unique path in the latter part of the exploration in the random embedding is $2^{-\ell}$, since at every vertex the embedding is chosen uniformly at random from two neighbors. Overall, we can upper bound the probability of $C^{\ell}_{(a,b)}$ by $\frac{2^{(d+\ell)/2}}{2^k-2t} \cdot 2^{-\ell} = \frac{2^{(d-\ell)/2}}{2^k-2t}$ (and by 0 if $\ell - d$ is odd or $\ell < d$), which yields the upper bound $\frac{2}{2^k-2t}$ on the probability of $C_{(a,b)}$, so we therefore ultimately get that

$$\mathrm{E}_G[P^G(T)] \le O(t^2/(2^k - t)). \tag{6.4}$$

∎

It is worth noting that the above bound is essentially tight. Indeed, making $2t+k$ queries we can find t leaves connected to the ROOT, and then the probability that there is a weld edge between two explored leaves is about $\binom{t}{2} \cdot 2^{-k}$.

LEMMA 6.12. *Let Game* B *be that of finding* ROOT *or a cycle in a random self-welded tree given the label of a uniformly random starting vertex. If $\mathcal{B}$ uses t queries, then its probability of winning is at most $O(t^2 \cdot 2^{-k})$.*

Proof. Again, we can bound this probability by bounding the probability that a random embedding π of an arbitrary fixed rooted binary tree T contains ROOT or a cycle. The only difference is that the root is now at a random vertex in the self-welded tree. The probability of finding a cycle is again $O(t^2/(2^k - t))$ by essentially the same argument as in the proof of Lemma 6.11.

To bound the probability of $\mathcal{B}$ finding the ROOT, observe that the probability of the random starting vertex s being at depth d is $2^d/(2^{k+1}-1) = O(2^{d-k})$. Now, consider the probability that $a \in T$ is mapped to the ROOT and let P_a denote the unique s to a path in T. Since s is at depth d, π has to embed the last d edges of P_a going upward consistently, which has probability at most 2^{-d}. Taking a union bound over all $a \in T$ and multiplying the probabilities gives the upper bound $t \cdot \sum_{d=0}^{\min(k,t)} 2^{d-k} \cdot 2^{-d} = t(\min(k,t)+1) \cdot 2^{-k} = O(t^2 \cdot 2^{-k})$, so the overall probability of $\mathcal{B}$ finding the ROOT or a cycle starting from a random vertex is $O(t^2 \cdot 2^{-k})$. ∎

LEMMA 6.13. *Given a* ROOT*, let Game* C *be that of finding the other* ROOT *or a cycle in a random welded tree. If $\mathcal{B}$ uses t queries, then its probability of winning is at most $O(t^2 \cdot 2^{-k})$.*

Proof. The proof is analogous to that of Lemma 6.11 and [25, Lemma 10]. ∎

LEMMA 6.14. *Let Game* D *be that of finding a* ROOT *or a cycle in a random welded tree given the label of a uniformly random starting vertex. If $\mathcal{B}$ uses t queries, then its probability of winning is at most $O(t^2 \cdot 2^{-k})$.*

Proof. The proof is essentially the same as that of Lemmas 6.12 and 6.13. ∎

6.3.3. Hardness of distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$. Let $\mathcal{A}$ be a classical randomized algorithm that distinguishes $\mathcal{G}_1$ from $\mathcal{G}_2$. Our strategy for proving the classical lower bound is to consider how the "knowledge graph" of $\mathcal{A}$ changes as it interacts with oracles that provide black-box descriptions of a random graph in $\mathcal{G}_1$ or $\mathcal{G}_2$. Recall that the knowledge graph of $\mathcal{A}$ is the graph that it sees. In the following, we refer

to vertices in the knowledge graph at each step as "known" and vertices outside it as "unknown."

We shall show that, regardless of whether we are given a graph from $\mathcal{G}_1$ or $\mathcal{G}_2$, the knowledge graph of $\mathcal{A}$ looks the same at each step with high probability if only a small number of queries have been made. To be more precise, we shall define a graph simulator $\mathcal{G}_s$ such that the knowledge graph of $\mathcal{A}$ when it interacts with either $\mathcal{G}_1$ or $\mathcal{G}_2$ looks the same as when it interacts with $\mathcal{G}_s$ with all but exponentially small probability, if only a subexponential number of queries are made.

To the benefit of $\mathcal{A}$, we assume that $\mathcal{A}$ has knowledge of which labels are antenna, ROOT, or body labels. We also suppose that all reached ROOT labels have been queried for free so that they and their neighbors all belong to the knowledge graph of $\mathcal{A}$ at the outset. We can assume that $\mathcal{A}$ does not query labels that have already been queried and that each query to a vertex label returns the labels of all neighbors of that vertex. Then, at each step, $\mathcal{A}$ must choose to perform one of the following actions:

I. Query an unknown body label.
II. Query an unknown antenna label.
III. Query a known body label.
IV. Query a known antenna label.

For ease of reference, we call the candy subgraphs of $\mathcal{G}_1$ and double-bow-tie subgraphs of $\mathcal{G}_2$ their "base graphs." At each step of $\mathcal{A}$, we say that a base graph has been *tapped* if any one of its vertices, other than a ROOT, is already in the knowledge graph of $\mathcal{A}$.

At each step of $\mathcal{A}$, we assume to its benefit that the following hold:

A1. $\mathcal{A}$ wins if, upon taking Action I or Action II, the label of a vertex in a tapped base graph is revealed.
A2. $\mathcal{A}$ wins if, upon taking Actions III or IV, the vertex whose label is queried is connected by an advice edge to a vertex in a tapped base graph.
A3. When $\mathcal{A}$ performs Action II, the entire half-antenna in which the queried antenna label resides is revealed (i.e., all vertex labels within the antenna as well as edges between them become known).
A4. When $\mathcal{A}$ performs Action IV, the entire half-antenna in which the queried antenna label resides is revealed.

The point of making these assumptions is simply to make it easier to analyze how the knowledge graph of $\mathcal{A}$ changes. We consider A1 and A2 because when they do not occur, $\mathcal{A}$ is essentially restricted to local traversal in each base graph. We consider A3 and A4 so that we do not need to unduly worry about how $\mathcal{A}$ traverses the antenna, which is irrelevant for distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$.

LEMMA 6.15. *If $\mathcal{A}$ uses t queries, then its probability of winning via either* A1 *or* A2 *above is at most $O(t^2 \cdot 2^{-k})$.*

Proof. In t queries, at most $2t$ base graphs can be tapped. However, there are $2(2^k - 1)$ base graphs in $\mathcal{G}_1$ and $\mathcal{G}_2$. Therefore, the probability of winning via A1 at each step is at most $O(t/2^k)$, so the probability of winning via A1 at any step is $O(t^2/2^k)$. The same bound holds for winning via A2 due to the randomness in the connection of advice edges. ∎

Let us consider a graph simulator $\mathcal{G}_s$ that can respond to the different actions of $\mathcal{A}$. As mentioned previously, we shall argue that its responses are similar to the responses of a random graph from either $\mathcal{G}_1$ and $\mathcal{G}_2$ when the number of queries is not too large. The responses of $\mathcal{G}_s$ depend on the actions of $\mathcal{A}$ as follows:

I. $\mathcal{G}_s$ returns a label chosen uniformly at random from antenna labels not currently in the knowledge graph. $\mathcal{G}_s$ also returns three labels chosen uniformly at random from body labels not currently in the knowledge graph.
II. $\mathcal{G}_s$ returns an entire half-antenna with labels chosen uniformly at random from antenna labels not currently in the knowledge graph. $\mathcal{G}_s$ also returns one body label chosen uniformly at random from body labels not currently in the knowledge graph.
III. $\mathcal{G}_s$ does one of the following two case-dependent actions.
   (a) If the queried body label is known because it is connected to a known antenna vertex, $\mathcal{G}_s$ returns three labels chosen uniformly at random from body labels not currently in the knowledge graph.
   (b) If the queried body label is known because it is connected to a known body vertex, $\mathcal{G}_s$ returns two labels chosen uniformly at random from body labels not currently in the knowledge graph as well as a label chosen uniformly at random from antenna labels not currently in the knowledge graph.
IV. $\mathcal{G}_s$ does one of the following two case-dependent actions.
   (a) If the queried antenna label is known because it is connected to a known body vertex, $\mathcal{G}_s$ returns an entire half-antenna with labels chosen uniformly at random from antenna labels not currently in the knowledge graph.
   (b) If the queried antenna label is known because it is connected to a known antenna vertex, $\mathcal{G}_s$ returns a label chosen uniformly at random from body labels not currently in the knowledge graph.

Note that in cases III and IV, $\mathcal{G}_s$ also returns vertex labels that are consistent with its knowledge graph when querying a known label, but we omitted describing this for convenience. In particular, in case IV(b), $\mathcal{G}_s$ returns antenna labels consistent with the relevant half-antenna, which must be entirely known by assumptions A3 and A4.

Now, suppose that $\mathcal{A}$ makes at most t queries to either $\mathcal{G}_1$ or $\mathcal{G}_2$. Then Lemma 6.15 says that the probability of it winning, at any step, via A1 or A2 is at most $O(t^2 \cdot 2^{-k})$. We henceforth assume that neither A1 nor A2 ever occurs. In this case, we explain why $\mathcal{G}_s$ responds like $\mathcal{G}_1$ and $\mathcal{G}_2$ when $\mathcal{A}$ performs each of the following actions:

I. When $\mathcal{A}$ queries an unknown body label, because A1 does not occur, the corresponding body vertex as well as its neighbors lie in two untapped base graphs. Therefore, the labels returned are outside the knowledge graph of $\mathcal{A}$ and so are distributed at random among those not in its knowledge graph, just like those returned by $\mathcal{G}_s$ in response to action I.
II. When $\mathcal{A}$ queries an unknown antenna label, because A1 does not occur, the corresponding entire half-antenna (cf. A3) and the neighboring body vertex lie in two untapped base graphs. Therefore, the labels returned are outside the knowledge graph of $\mathcal{A}$ and so are distributed at random among those not in its knowledge graph, just like those returned by $\mathcal{G}_s$ in response to action II.
III. When $\mathcal{A}$ queries a known body label, there are two cases, depending on how the body vertex is known.
   (a) If it is known because it is connected to a known antenna vertex by an advice edge, then it must be in an untapped base graph because A2 does not occur. Hence $\mathcal{G}_s$ behaves the same as $\mathcal{G}_i$.
   (b) If it is known because it is connected to a known body vertex, then Lemmas 6.11 and 6.12 (respectively, Lemmas 6.13 and 6.14) imply that

$\mathcal{G}_s$ behaves the same as $\mathcal{G}_2$ (respectively, $\mathcal{G}_1$), except with probability at most $O(t^2 \cdot 2^{-k})$. Lemmas 6.11 and 6.13 address the case when the queried body vertex is connected to the root of a base graph. Lemmas 6.12 and 6.14 address the case when it is connected to a random vertex in the body of a base graph.

IV. When $\mathcal{A}$ queries a known antenna label, there are two cases, depending on how the antenna vertex is known.
   (a) If it is known because it is connected to a known body vertex by an advice edge, then it must be in an untapped base graph because A2 does not occur. Hence $\mathcal{G}_s$ behaves the same as $\mathcal{G}_i$ (cf. A4).
   (b) If it is known because it is connected to a known antenna vertex, then the entire half-antenna must already be known (cf. A4), and the body vertex connected to it must be in an untapped base graph because A2 does not occur. Hence $\mathcal{G}_s$ behaves the same as $\mathcal{G}_i$.

Therefore, the probability of $\mathcal{A}$ distinguishing $\mathcal{G}_i$ from $\mathcal{G}_s$ is at most $O(t^2 \cdot 2^{-k})$ for each $i = 1, 2$. Therefore, the probability of distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$ is also at most $O(t^2 \cdot 2^{-k})$. Now, Lemma 6.9 implies that there exists a constant ϵ such that for all k sufficiently large, the probability that $\mathcal{G}_2$ is ϵ-*close* to $\mathcal{G}_1$ is at most 0.1. Let $\mathcal{G}_2'$ be a random graph from those in $\mathcal{G}_2$ that are ϵ-far from $\mathcal{G}_1$. Then the probability of distinguishing $\mathcal{G}_2'$ from $\mathcal{G}_2$ is at most 0.1. Therefore, the probability of distinguishing $\mathcal{G}_2'$ from $\mathcal{G}_1$ is at most $0.1 + O(t^2 \cdot 2^{-k})$. Hence, to distinguish $\mathcal{G}_1$ from $\mathcal{G}_2'$ with probability at least 2/3 requires $t = \exp(\Omega(k))$ queries. This establishes the desired classical lower bound in Theorem 6.1.

7. Open problems. We conclude by discussing a few open problems.

We proved that p-uniform hypergraph properties have at most a power $3p$ separation between quantum and randomized query complexity. However, we do not know if this is tight.

OPEN PROBLEM 7.1. *What is the largest possible separation between $Q(f)$ and $R(f)$ for p-uniform hypergraph properties f? That is, what is the largest k for which there exists such an f with $R(f) = \Omega(Q(f)^k)$?*

We remark that the problem is open even for the case $p = 1$ of fully symmetric functions, where the best upper bound is $k \leq 3$ due to Chailloux [23], and the best lower bound is $k \geq 2$ for the OR function [34]. For larger p, it is at least possible to exhibit functions with a power separation of $k \geq p/2$ by appealing to Proposition 5.1, choosing the function f in Proposition 5.1 to be Forrelation [2] and using an upper bound of $b(G) = O(m)$ for G the action of S_m on $\binom{m}{p}$ points (Theorem 3.2 of [36]). Thus there must be some dependence on p.

In Corollary 5.12, we showed that well-shuffling permutation groups must be constructed out of a constant number of primitive groups with sufficiently large minimal base size. We conjecture that a converse holds, which would imply a complete dichotomy regarding which permutation groups allow superpolynomial quantum speedups and which do not.

CONJECTURE 7.2. *Let $\mathcal{G}$ be a collection of permutation groups that satisfies conditions* (i)–(iv) *of Corollary* 5.12. *Then $\mathcal{G}$ is well-shuffling, and thus $R(f) = Q(f)^{O(1)}$ for every $f \in F(\mathcal{G})$.*

Conjecture 7.2 is based on our intuition that it is challenging to find an interesting family of permutation groups that satisfies these conditions but that cannot be constructed out of well-shuffling primitive groups using the transformations that preserve

the well-shuffling property. For example, even small subgroups of a wreath product of well-shuffling groups can remain well-shuffling: the product of two well-shuffling permutation groups $G_1 \times G_2$ (as defined in Definition 4.16) is well-shuffling, as shown in Theorem 4.17. But $G_1 \times G_2$ can be viewed as a subgroup of $G_1 \wr G_2$ in imprimitive action and is potentially much smaller than the full wreath product. Based on this intuition, we expect that it is possible to prove Conjecture 7.2 essentially in the same way we classified the well-shuffling primitive groups, via reduction to the symmetric and alternating groups.

In this form, Conjecture 7.2 would also imply that the well-shuffling property completely characterizes whether a collection of permutation groups allows superpolynomial quantum speedups or not. It would be interesting to prove such a characterization directly, without appealing to the structure of such groups.

We showed how to construct superpolynomial quantum speedups out of any sufficiently small permutation group in Proposition 5.1. However, the construction requires large alphabets (indeed, even larger than the number of input symbols). Might it be possible to construct such functions over a smaller alphabet or even a Boolean alphabet? Note that some permutation groups cannot be realized as the group of symmetries of a function with Boolean inputs [42].

Finally, while we constructed a separation between classical and quantum graph property testing in the adjacency list model, it remains unclear whether such a separation exists for graph properties of practical interest. For example, it remains open whether there could be an exponential quantum speedup for testing bipartiteness [7]. Going beyond that particular example, it would be interesting to investigate the possibility of a superpolynomial quantum speedup for testing *monotone* graph properties (i.e., those that remain satisfied when adding edges to a yes instance).

Appendix A. Proof of the quantum minimax lemma. We prove Lemma 2.1, which we restate below.

LEMMA A.1 (minimax for bounded-error quantum algorithms). *Let f be a (possibly partial) Boolean function with $\mathrm{Dom}(f) \subseteq \Sigma^n$, and let $\epsilon \in (0, 1/2)$. Then there is a distribution μ supported on $\mathrm{Dom}(f)$ which is* hard *for f in the following sense: any quantum algorithm using fewer than $\mathrm{Q}_\epsilon(f)$ quantum queries for computing f must have average error more than ϵ on inputs sampled from μ.*

Proof. By [12], there is a finite bound B expressible in terms of n and $|\Sigma|$ on the necessary size of the work space register for a quantum algorithm computing f with error at most ϵ. This means the quantum query algorithms we deal with can be assumed without loss of generality to have work space size B. A quantum algorithm making T queries can be represented as a sequence of T unitary matrices of size upper bounded by B; this can be arranged as a finite vector of complex numbers. It is not hard to see that the set of all such valid quantum algorithms is a compact set.

For a quantum algorithm Q, let $\mathrm{err}(Q, x)$ denote the error Q makes when run on input $x \in \mathrm{Dom}(x)$; this is $\Pr[Q(x) \neq f(x)]$, where $Q(x)$ is the random variable for the measured output of Q when run on x. We note that $\mathrm{err}(Q, x)$ is a continuous function of Q. Let v_Q be the vector in $\mathbb{R}^{|\mathrm{Dom}(f)|}$ defined by $v_Q[x] := \mathrm{err}(Q, x)$. Then v_Q is a continuous function of Q. Further, let V be the set of all such vectors v_Q for valid quantum algorithms Q which make at most $\mathrm{Q}_\epsilon(f) - 1$ queries. Since the set of such valid quantum algorithms is compact and since v_Q is continuous in Q, we conclude that V is compact. Furthermore, we claim that V is convex: this is because for any two quantum algorithms Q and Q', there is a quantum algorithm Q'' that behaves like their mixture (in terms of its error on each input x).

Next, let $\Delta \subseteq \mathbb{R}^{|\mathrm{Dom}(f)|}$ be the set of all probability distributions over $\mathrm{Dom}(f)$. Then Δ is also convex and compact. Finally, define $\alpha : V \times \Delta \to \mathbb{R}$ by $\alpha(v,\mu) := \mathbb{E}_{x\leftarrow\mu} v[x] = \sum_{x\in\mathrm{Dom}(f)} \mu[x]v[x]$. Then α is continuous in each coordinate and is *saddle*: that is, $\alpha(\cdot,\mu)$ is convex for each $\mu \in \Delta$ (indeed, it is linear), and $\alpha(v,\cdot)$ is concave for each $v \in V$ (indeed, it is also linear). A standard minimax theorem (e.g., [53]) then gives us

$$\min_{v\in V}\max_{\mu\in\Delta}\alpha(v,\mu) = \max_{\mu\in\Delta}\min_{v\in V}\alpha(v,\mu).$$

For the left-hand side, it is clear that the maximum over μ (once the vector v has been chosen) is the same as the maximum over $x \in \mathrm{Dom}(f)$ of $v[x]$. This makes the left-hand side the minimum over $v \in V$ of $\|v\|_\infty$, or equivalently, the minimum worst-case error of quantum algorithms making at most $\mathrm{Q}_\epsilon(f) - 1$ queries. By the definition of $\mathrm{Q}_\epsilon(f)$, this minimum must be strictly greater than ϵ (or else $\mathrm{Q}_\epsilon(f)$ would be smaller). Hence the left-hand side is strictly greater than ϵ.

Looking at the right-hand side, we get a single distribution μ such that every quantum algorithm Q making at most $\mathrm{Q}_\epsilon(f) - 1$ queries must make error greater than ϵ against μ, as desired. ∎

Appendix B. Quantum speedup for deep wreath product symmetries. Here, we complete the proof of Theorem 5.11. We first define a problem called 1-Fault Direct Trees with symmetry group $G = S_{n_1} \wr S_{n_2} \wr \cdots S_{n_d}$. Then, we argue that $Q(\text{1-Fault Direct Trees}) = O(1)$ and $R(\text{1-Fault Direct Trees}) = \Omega(\log d)$.

B.1. Definitions. We begin by formally defining 1-Fault Direct Trees, generalizing (but closely following) [41, Definitions 1 and 2].

Fix $n_1, \ldots, n_d$, and let $N = \prod_{i=1}^d n_i$. Consider a depth-d Boolean formula with N inputs in which all of the gates at distance $i-1$ from the root have fan-in n_i. Label the gates at depth $i-1$ by a function f_i defined by $f_i : S_i \to \{0,1\}$, where $S_i \subseteq \{0,1\}^{n_i}$ consists of the n_i-bit strings that have Hamming weight in $\{0, \lfloor n_i/2 \rfloor, \lceil n_i/2 \rceil, n_i\}$, and $f_i(x) = 0$ if and only if $x = 1^{n_i}$. The Boolean formula is equivalent to the block composition $f_1 \circ f_2 \circ \cdots \circ f_d$.

Consider evaluating the formula on some fixed input $X \in \{0,1\}^N$. Fix a gate v labeled by f_i, and let $x \in \{0,1\}^{n_i}$ be the evaluations of its children. We say that v is *trivial* if $x = 0^{n_i}$ or $x = 1^{n_i}$; otherwise v is said to be *fault.* If v is trivial, then we say that all its children are *strong.* Otherwise, if v is fault, the child vertices that evaluate to 1 are defined to be *weak,* and the child vertices that evaluate to 0 are defined to be *strong.*

To each gate v, we assign an integer $\kappa(v)$ such that the following hold:

- $\kappa(v) = 0$ for leaf nodes.
- If v has children $v_1, \ldots, v_{n_i}$, then
  - if v is trivial, $\kappa(v) = \max_{j\in[n_i]} \kappa(v_j) = \max_{j:v_j \text{ is strong}} \kappa(v_j)$;
  - if v is fault, $\kappa(v) = 1 + \max_{j:v_j \text{ is strong}} \kappa(v_j)$.

We say that the X satisfies the 1-fault condition if $\kappa(v_r) \le 1$, where v_r is the root. The 1-Fault Direct Trees problem is to evaluate this formula $f_1 \circ f_2 \circ \cdots \circ f_d$ on X, promised that X satisfies the 1-fault condition.

Observe that 1-Fault Direct Trees is symmetric under the iterated wreath product $S_{n_1} \wr S_{n_2} \wr \cdots \wr S_{n_d}$ of symmetric groups, because for any vertex v, the quantity $\kappa(v)$ does not depend on the ordering of the children of v.

B.2. Quantum query upper bound via span programs. Using the notation from Definition 2.1 of [57], we define a span program P_i for each f_i by $v_j = \frac{1}{\sqrt{n_i}} \in \mathbb{R}^1$ and $\chi_j = \bar{x}_j$ for all $j \in [n_i]$. We now verify that P_i satisfies the key properties that are needed to prove the quantum query upper bound in [41]. We have $r_0 = \frac{1}{\sqrt{n_i}}(1, 1, \ldots, 1) \in \mathbb{C}^{n_i}$ and $C = 1$ as in Definition 2.2 of [57], which gives the following:

- $\text{WSIZE}_1(P_i, x) = 1$ if input x makes f_i trivial. If $x = 0^{n_i}$, choosing $\vec{w} = r_0$ suffices to show this. Otherwise, if $x = 1^{n_i}$, then we are forced to choose $\vec{w} = r_0$.
- $\text{WSIZE}_1(P_i, x) \leq 3$ if input x makes f_i fault. Suppose $x = 0^{\lfloor n_i/2 \rfloor} 1^{\lceil n_i/2 \rceil}$. Choosing $w_j = \frac{\sqrt{n_i}}{\lfloor n_i/2 \rfloor}$ for $j \in [\lfloor n_i/2 \rfloor]$ gives witness size $\frac{n_i}{\lfloor n_i/2 \rfloor} \leq 3$ (tight for $n_i = 3$). Similarly, for $x = 0^{\lceil n_i/2 \rceil} 1^{\lfloor n_i/2 \rfloor}$, we take $w_j = \frac{\sqrt{n_i}}{\lceil n_i/2 \rceil}$ for $j \in [\lceil n_i/2 \rceil]$ to get witness size $\frac{n_i}{\lceil n_i/2 \rceil} \leq 2$.
- For $\text{WSIZE}_s(P_i, x)$, s_j do not affect the witness size, where the jth input bit is weak. This is because the only weak input bits are those where $x_j = 1$ and f_i is fault on x. Then $f_i(x) = 1$, and we have $\chi_j = 0$, so we are forced to take $w_j = 0$.

Now that these three conditions are satisfied, the proof of Theorem 2 of [41] goes through without modifications. Thus it shows that our version of 1-Fault Direct Trees has quantum query complexity $O(1)$.

B.3. Randomized query complexity lower bound. It remains to lower bound the randomized query complexity of 1-Fault Direct Trees as we have defined it. Suppose we further promise that at every vertex in the tree corresponding to f_i, the child subtrees are partitioned into two sets, one of size $\lfloor n_i/2 \rfloor$ and one of size $\lceil n_i/2 \rceil$, such that all of the subtrees in one part have the same values at corresponding leaves. Then with this additional promise, the problem remains at least as hard as the 1-Fault Nand Trees problem defined in [41, 57]: if we always take the partition to be the $\lfloor n_i/2 \rfloor$ left vertices and the $\lceil n_i/2 \rceil$ right vertices, then it is in fact equivalent to 1-Fault Nand Trees where some inputs are repeated. [57] proved that depth-d 1-Fault Nand Trees have randomized query complexity at least $\Omega(\log d)$, which completes our proof.

Acknowledgments. We thank Scott Aaronson and Carl Miller for many helpful discussions. SP also thanks Zak Webb for many related discussions. Part of this work was done while visiting the Simons Institute for the Theory of Computing. We gratefully acknowledge the Institute's hospitality.

REFERENCES

[1] S. Aaronson and A. Ambainis, *The need for structure in quantum speedups*, Theory Comput., 10 (2014), pp. 133–166, https://doi.org/10.4086/toc.2014.v010a006; preprint available online from arXiv:0911.0996.

[2] S. Aaronson and A. Ambainis, *Forrelation: A problem that optimally separates quantum from classical computing*, in Proceedings of the 47th ACM Symposium on the Theory of Computing, 2015, pp. 307–316, https://doi.org/10.1145/2746539.2746547; preprint available online from arXiv:1411.5729.

[3] S. Aaronson and S. Ben-David, *Sculpting quantum speedups*, in Proceedings of the 31st IEEE Conference on Computational Complexity, 2016, pp. 26:1–26:28, https://doi.org/10.4230/LIPIcs.CCC.2016.26; preprint available online from https://arxiv.org/abs/1512.04016.

[4] S. Aaronson, N.-H. Chia, H.-H. Lin, C. Wang, and R. Zhang, *On the quantum complexity of closest pair and related problems*, 35th Computational Complexity Confer-

ence (CCC 2020), Leibniz International Proceedings in Informatics (LIPIcs), 169, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, pp. 16:1–16:43, https://doi.org/10.4230/LIPIcs.CCC.2020.16.

[5] S. Aaronson and Y. Shi, *Quantum lower bounds for the collision and the element distinctness problems*, J. ACM, 51 (2004), pp. 595–605, https://doi.org/10.1145/1008731.1008735.

[6] A. Ambainis, *Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range*, Theory Comput., 1 (2005), pp. 37–46, https://doi.org/10.4086/toc.2005.v001a003; preprint available online from arXiv:quant-ph/0305179.

[7] A. Ambainis, A. M. Childs, and Y.-K. Liu, *Quantum property testing for bounded-degree graphs*, in Proceedings of the 15th International Workshop on Randomization and Computation, Lecture Notes in Comput. Sci. 6845, Springer, 2011, pp. 365–376, https://doi.org/10.1007/978-3-642-22935-0_31; preprint available online from arXiv:1012.3174.

[8] A. Ambainis, K. Iwama, M. Nakanishi, H. Nishimura, R. Raymond, S. Tani, and S. Yamashita, *Quantum query complexity of Boolean functions with small on-sets*, in Proceedings of the 19th International Symposium on Algorithms and Computation, 2008, pp. 907–918, https://doi.org/10.1007/978-3-540-92182-0.

[9] A. Ambainis and R. Špalek, *Quantum algorithms for matching and network flows*, in Proceedings of the 23rd Symposium on Theoretical Aspects of Computer Science, Springer, 2006, pp. 172–183, https://doi.org/10.1007/11672142_13; preprint available online from arXiv:quant-ph/0508205.

[10] S. Apers and T. Lee, *Quantum complexity of minimum cut*, in Proceedings of the 36th IEEE Conference on Computational Complexity, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2021, https://doi.org/10.4230/LIPIcs.CCC.2021.28; preprint available online from arXiv:2011.09823.

[11] S. Balasubramanian, T. Li, and A. Harrow, *Exponential Speedups for Quantum Walks in Random Hierarchical Graphs*, arXiv:2307.15062, 2023.

[12] H. Barnum, M. Saks, and M. Szegedy, *Quantum query complexity and semi-definite programming*, in Proceedings of the 18th IEEE Conference on Computational Complexity, 2003, pp. 179–193, https://doi.org/10.1109/CCC.2003.1214419.

[13] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, J. ACM, 48 (2001), pp. 778–797, https://doi.org/10.1145/502090.502097; preprint available online from https://arxiv.org/abs/quant-ph/9802049.

[14] A. Belovs, *Span programs for functions with constant-sized* 1*-certificates*, in Proceedings of the 44th ACM Symposium on the Theory of Computing, 2012, pp. 77–84, https://doi.org/10.1145/2213977.2213985; preprint available online from arXiv:1105.4024.

[15] A. Belovs and R. Špalek, *Adversary lower bound for the k-sum problem*, in Proceedings of the 4th Innovations in Theoretical Computer Science Conference, 2013, pp. 323–328, https://doi.org/10.1145/2422436.2422474; preprint available online from arXiv:1206.6528.

[16] S. Ben-David, *The structure of promises in quantum speedups*, in Proceedings of the 11th Conference on the Theory of Quantum Computation, Communication, and Cryptography, 2016, pp. 7:1–7:14, https://doi.org/10.4230/LIPIcs.TQC.2016.7; preprint available online from https://arxiv.org/abs/1409.3323.

[17] D. W. Berry, A. M. Childs, and R. Kothari, *Hamiltonian simulation with nearly optimal dependence on all parameters*, in Proceedings of the 56th IEEE Symposium on Foundations of Computer Science, 2015, pp. 792–809, https://doi.org/10.1109/FOCS.2015.54; preprint available online from https://arxiv.org/abs/1501.01715.

[18] A. Berzina, A. Dubrovsky, R. Freivalds, L. Lace, and O. Scegulnaja, *Quantum query complexity for some graph problems*, in SOFSEM 2004: Theory and Practice of Computer Science, Springer, Berlin, 2004, pp. 140–150, https://doi.org/10.1007/978-3-540-24618-3_11.

[19] K. D. Blaha, *Minimum bases for permutation groups: The greedy approximation*, J. Algorithms, 13 (1992), pp. 297–306, https://doi.org/10.1016/0196-6774(92)90020-D.

[20] H. Buhrman and R. de Wolf, *Complexity measures and decision tree complexity: A survey*, Theory Comput., 288 (2002), pp. 21–43, https://doi.org/10.1016/S0304-3975(01)00144-X.

[21] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf, *Quantum algorithms for element distinctness*, in Proceedings of the 16th IEEE Conference on Computational Complexity, 2001, pp. 131–137; preprint also available online from arXiv:quant-ph/0007016.

[22] M. Bun, R. Kothari, and J. Thaler, *The polynomial method strikes back: Tight quantum query bounds via dual polynomials*, in Proceedings of the 50th ACM Symposium on the Theory of Computing, 2018, pp. 297–310, https://doi.org/10.1145/3188745.3188784; preprint also available online from arXiv:1710.09079.

[23] A. Chailloux, *A note on the quantum query complexity of permutation symmetric functions*, in Proceedings of the 10th Innovations in Theoretical Computer Science Conference, 2018, pp. 19:1–19:7, https://doi.org/10.4230/LIPIcs.ITCS.2019.19; preprint also available online from https://arxiv.org/abs/1810.01790.

[24] S. Chakraborty, C. Kayal, and M. Paraashar, *Separations between combinatorial measures for transitive functions*, in Proceedings of the 49th International Colloquium on Automata, Languages, and Programming, LIPIcs. Leibniz Int. Proc. Inform. 229, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022, pp. 36:1–36:20, https://doi.org/10.4230/LIPIcs.ICALP.2022.36; preprint also available online from https://arxiv.org/abs/2103.12355.

[25] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, *Exponential algorithmic speedup by quantum walk*, in Proceedings of the 35th ACM Symposium on the Theory of Computing, 2003, pp. 59–68, https://doi.org/10.1145/780542.780552; preprint also available online from https://arxiv.org/abs/quant-ph/0209131.

[26] A. M. Childs and R. Kothari, *Quantum query complexity of minor-closed graph properties*, SIAM J. Comput., 41 (2012), pp. 1426–1450, https://doi.org/10.1137/110833026.

[27] R. Cleve, *The query complexity of order-finding*, Inform. and Comput., 192 (2004), pp. 162–171, https://doi.org/10.1016/j.ic.2004.04.001; preprint also available online from arXiv:quant-ph/9911124.

[28] J. D. Dixon and B. Mortimer, *Permutation Groups*, Grad. Texts Math. 163, Springer, New York, 2012, https://doi.org/10.1007/978-1-4612-0731-3.

[29] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla, *Quantum query complexity of some graph problems*, SIAM J. Comput., 35 (2006), pp. 1310–1328, https://doi.org/10.1137/050644719.

[30] A. Gilyén, M. B. Hastings, and U. Vazirani, *(Sub)exponential advantage of adiabatic quantum computation with no sign problem*, in Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, 2021, pp. 1357–1369, https://doi.org/10.1145/3406325.3451060.

[31] A. Gilyén, Y. Su, G. H. Low, and N. Wiebe, *Quantum singular value transformation and beyond: Exponential improvements for quantum matrix arithmetics*, in Proceedings of the 51st ACM Symposium on the Theory of Computing, 2019, pp. 193–204, https://doi.org/10.1145/3313276.3316366; preprint available online from https://arxiv.org/abs/1806.01838.

[32] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750, https://doi.org/10.1145/285055.285060.

[33] O. Goldreich and D. Ron, *Property testing in bounded degree graphs*, in Proceedings of the 29th ACM Symposium on the Theory of Computing, 1997, pp. 406–415, https://doi.org/10.1145/258533.258627.

[34] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th ACM Symposium on the Theory of Computing, 1996, pp. 212–219, https://doi.org/10.1145/237814.237866; preprint available online from arXiv:quant-ph/9605043.

[35] Z. Guan, Y. Huang, P. Yao, and Z. Ye, *Quantum and classical communication complexity of permutation-invariant functions*, in Proceedings of the 41st Symposium on Theoretical Aspects of Computer Science, LIPIcs. Leibniz Int. Proc. Inform. 289, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024, pp. 39:1–39:19, https://doi.org/10.4230/LIPIcs.STACS.2024.39; preprint available online from https://arxiv.org/abs/2401.00454.

[36] Z. Halasi, *On the base size for the symmetric group acting on subsets*, Studia Sci. Math. Hungar., 49 (2012), pp. 492–500, https://doi.org/10.1556/sscmath.49.2012.4.1222.

[37] A. Hulpke, *Notes on Computational Group Theory*, https://www.math.colostate.edu/~hulpke/CGT/cgtnotes.pdf, 2010.

[38] S. Jukna, *Extremal Combinatorics*, Texts Theoret. Comput. Sci. EATCS Ser., Springer, New York, 2011, https://doi.org/10.1007/978-3-642-17364-6.

[39] D. M. Kane and S. A. Kutin, *Quantum interpolation of polynomials*, Quantum Inf. Comput., 11 (2011), pp. 95–103, https://doi.org/10.26421/QIC11.1-2-7; preprint also available from arXiv:0909.5683.

[40] A. Kerber, *Applied Finite Group Actions*, Algorithms Combin. 19, Springer, New York, 2013, https://doi.org/10.1007/978-3-662-11167-3.

[41] S. Kimmel, *Quantum adversary (upper) bound*, in Proceedings of the 12th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 7391, 2012, pp. 557–568, https://doi.org/10.1007/978-3-642-31594-7_47; preprint available online from arXiv:1101.0797.

[42] A. Kisielewicz, *Symmetry groups of Boolean functions and constructions of permutation groups*, J. Algebra, 199 (1998), pp. 379–403, https://doi.org/10.1006/jabr.1997.7198.

[43] S. Kutin, *Quantum lower bound for the collision problem with small range*, Theory Comput., 1 (2005), pp. 29–36, https://doi.org/10.4086/toc.2005.v001a002; preprint available online from arXiv:quant-ph/0304162.

[44] F. Le Gall, *Improved quantum algorithm for triangle finding via combinatorial arguments*, in Proceedings of the 55th IEEE Symposium on Foundations of Computer Science, 2014, pp. 216–225, https://doi.org/10.1109/FOCS.2014.31; preprint available online from arXiv:1407.0085.

[45] T. Lee, F. Magniez, and M. Santha, *Improved quantum query algorithms for triangle finding and associativity testing*, in Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms, 2013, pp. 1486–1502; preprint available online from https://arxiv.org/abs/1210.1014.

[46] M. W. Liebeck, *On minimal degrees and base sizes of primitive permutation groups*, Arch. Math. (Basel), 43 (1984), pp. 11–15, https://doi.org/10.1007/BF01193603.

[47] G. H. Low and I. L. Chuang, *Hamiltonian simulation by qubitization*, Quantum, 3 (2019), 163, https://doi.org/10.22331/q-2019-07-12-163; preprint available online from arXiv:1610.06546.

[48] F. Magniez, M. Santha, and M. Szegedy, *Quantum algorithms for the triangle problem*, SIAM J. Comput., 37 (2007), pp. 413–424.

[49] A. Montanaro and R. de Wolf, *A survey of quantum property testing*, Theory Comput., 7 (2016), https://doi.org/10.4086/toc.gs.2016.007; preprint available online from arXiv:1310.2035.

[50] S. Podder, P. Yao, and Z. Ye, *On the fine-grained query complexity of symmetric functions*, in Proceedings of the 34th International Symposium on Algorithms and Computation, LIPIcs. Leibniz Int. Proc. Inform 283, Schloss Dagstuhl–Leibniz-Zentrum für Informatik 289, 2023, pp. 55:1–55:18, https://doi.org/10.4230/LIPIcs.ISAAC.2023.55; preprint available online from arXiv:2309.11279.

[51] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Rev., 41 (1999), pp. 303–332, https://doi.org/10.1137/S0097539795293172; preprint available online from arXiv:quant-ph/9508027.

[52] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483, https://doi.org/10.1137/S0097539796298637.

[53] M. Sion, *On general minimax theorems*, Pacific J. Math., 8 (1958), pp. 171–176, https://doi.org/10.2140/pjm.1958.8.171.

[54] X. Sun, A. C. Yao, and S. Zhang, *Graph properties and circular functions: How low can quantum query complexity go?*, in Proceedings of the 19th IEEE Conference on Computational Complexity, 2004, pp. 286–293, https://doi.org/10.1109/CCC.2004.14.

[55] N. K. Vereshchagin, *Randomized Boolean decision trees: Several remarks*, Theoret. Comput. Sci., 207 (1998), pp. 329–342, https://doi.org/10.1016/S0304-3975(98)00071-1.

[56] A. C.-C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 222–227, https://doi.org/10.1109/SFCS.1977.24.

[57] B. Zhan, S. Kimmel, and A. Hassidim, *Super-polynomial quantum speed-ups for Boolean evaluation trees with hidden structure*, in Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, 2012, pp. 249–265, https://doi.org/10.1145/2090236.2090258; preprint available online from arXiv:1101.0796.

[58] M. Zhandry, *Secure identity-based encryption in the quantum random oracle model*, in Proceedings of Advances in Cryptology, Springer, Berlin, 2012, pp. 758–775, https://doi.org/10.1007/978-3-642-32009-5_44.

[59] M. Zhandry, *A note on the quantum collision and set equality problems*, Quantum Inf. Comput., 15 (2015), pp. 557–567, https://doi.org/10.26421/QIC15.7-8; preprint available online from arXiv:1312.1027.

[60] S. Zhang, *On the power of Ambainis lower bounds*, Theoret. Comput. Sci., 339 (2005), pp. 241–256, https://doi.org/10.1016/j.tcs.2005.01.019; preprint available online from arXiv:quant-ph/0311060.

[61] Y. Zhu, *Quantum query complexity of subgraph containment with constant-sized certificates*, Int. J. Quantum Inf., 10 (2012), 1250019, https://doi.org/10.1142/S0219749912500190; preprint available online from arXiv:1109.4165.