



## Testing of sequences by simulation

**Antal Iványi**

Eötvös Loránd University  
Department of Computer Algebra  
H-1117, Budapest, Hungary  
Pázmány sétány 1/C  
email: [tony@compalg.inf.elte.hu](mailto:tony@compalg.inf.elte.hu)

**Balázs Novák**

Eötvös Loránd University  
Department of Computer Algebra  
H-1117, Budapest, Hungary  
Pázmány sétány 1/C  
email: [psziho@inf.elte.hu](mailto:psziho@inf.elte.hu)

**Abstract.** Let  $\xi$  be a random integer vector, having uniform distribution

$$\mathbf{P}\{\xi = (i_1, i_2, \dots, i_n) = 1/n^n\} \text{ for } 1 \leq i_1, i_2, \dots, i_n \leq n.$$

A realization  $(i_1, i_2, \dots, i_n)$  of  $\xi$  is called *good*, if its elements are different. We present algorithms LINEAR, BACKWARD, FORWARD, TREE, GARBAGE, BUCKET which decide whether a given realization is good. We analyse the number of comparisons and running time of these algorithms using simulation gathering data on all possible inputs for small values of  $n$  and generating random inputs for large values of  $n$ .

### 1 Introduction

Let  $\xi$  be a random integer vector, having uniform distribution

$$\mathbf{P}\{\xi = (i_1, i_2, \dots, i_n)\} = 1/n^n$$

for  $1 \leq i_1, i_2, \dots, i_n \leq n$ .

A realization  $(i_1, i_2, \dots, i_n)$  of  $\xi$  is called *good*, if its elements are different. We present six algorithms which decide whether a given realization is good.

This problem arises in connection with the design of agricultural [4, 5, 57, 72] and industrial [34] experiments, with the testing of Latin [1, 9, 22, 23, 27, 32,

---

**Computing Classification System 1998:** G.2.2

**Mathematics Subject Classification 2010:** 68M20

**Key words and phrases:** random sequences, analysis of algorithms, Latin squares, sudoku squares

53, 54, 63, 64] and sudoku [3, 4, 6, 12, 13, 14, 15, 16, 17, 20, 21, 22, 26, 29, 30, 31, 41, 42, 44, 46, 47, 51, 55, 59, 61, 64, 66, 67, 68, 69, 70, 72, 74] squares, with genetic sequences and arrays [2, 7, 8, 18, 24, 28, 35, 36, 37, 38, 45, 48, 49, 50, 56, 65, 71, 73, 75], with sociology [25], and also with the analysis of the performance of computers with interleaved memory [11, 33, 39, 40, 41, 43, 52].

Section 2 contains the pseudocodes of the investigated algorithms. In Section 3 the results of the simulation experiments and the basic theoretical results are presented. Section 4 contains the summary of the paper.

Further simulation results are contained in [62]. The proofs of the lemmas and theorems can be found in [43].

## 2 Pseudocodes of the algorithms

This section contains the pseudocodes of the investigated algorithms LINEAR, BACKWARD, FORWARD, TREE, GARBAGE, and BUCKET. The pseudocode conventions described in the book [19] written by Cormen, Leiserson, Rivest, and Stein are used.

The inputs of the following six algorithms are  $n$  (the length of the sequence  $s$ ) and  $s = (s_1, s_2, \dots, s_n)$ , a sequence of nonnegative integers with  $0 \leq s_i \leq n$  for  $1 \leq i \leq n$  in all cases. The output is always a logical variable  $g$  (its value is TRUE, if the input sequence is good, and FALSE otherwise).

The working variables are usually the cycle variables  $i$  and  $j$ .

### 2.1 Definition of algorithm LINEAR

LINEAR writes zero into the elements of an  $n$  length vector  $\mathbf{v} = (v_1, v_2, \dots, v_n)$ , then investigates the elements of the realization and if  $v[s_i] > 0$  (signalising a repetition), then stops, otherwise adds 1 to  $v[s[i]]$ .

LINEAR( $n, s$ )

```

01  $g \leftarrow \text{TRUE}$ 
02 for  $i \leftarrow 1$  to  $n$ 
03   do  $v[i] \leftarrow 0$ 
04 for  $i \leftarrow 1$  to  $n$ 
05   do if  $v[s[i]] > 0$ 
06     then  $g \leftarrow \text{FALSE}$ 
07     return  $g$ 
08     else  $v[s[i]] \leftarrow v[s[i]] + 1$ 
09 return  $g$ 

```

## 2.2 Definition of algorithm BACKWARD

BACKWARD compares the second ( $i_2$ ), third ( $i_3$ ), ..., last ( $i_n$ ) element of the realization  $s$  with the previous elements until the first collision or until the last pair of elements.

BACKWARD( $n, s$ )

```
01  $g \leftarrow \text{TRUE}$ 
02 for  $i \leftarrow 2$  to  $n$ 
03     do for  $j \leftarrow i - 1$  downto 1
04         do if  $s[i] = s[j]$ 
05             then  $g \leftarrow \text{FALSE}$ 
06         return  $g$ 
07 return  $g$ 
```

## 2.3 Definition of algorithm FORWARD

FORWARD compares the first ( $s_1$ ), second ( $s_2$ ), ..., last but one ( $s_{n-1}$ ) element of the realization with the following elements until the first collision or until the last pair of elements.

FORWARD( $n, s$ )

```
01  $g \leftarrow \text{TRUE}$ 
02 for  $i \leftarrow 1$  to  $n - 1$ 
03     do for  $j \leftarrow i + 1$  to  $n$ 
04         do if  $s[i] = s[j]$ 
05             then  $g \leftarrow \text{FALSE}$ 
06         return  $g$ 
07 return  $g$ 
```

## 2.4 Definition of algorithm TREE

TREE builds a random search tree from the elements of the realization and finishes the construction of the tree if it finds the following element of the realization in the tree (then the realization is not good) or it tested the last element too without a collision (then the realization is good).

TREE( $n, s$ )

```
01  $g \leftarrow \text{TRUE}$ 
02 let  $s[1]$  be the root of a tree
03 for  $i \leftarrow 2$  to  $n$ 
```

```

04   if [s[i] is in the tree
05       then g ← FALSE
06           return
07       else insert s[i] in the tree
08 return g

```

## 2.5 Definition of algorithm GARBAGE

This algorithm is similar to LINEAR, but it works without the setting zeros into the elements of a vector requiring linear amount of time.

Beside the cycle variable  $i$  GARBAGE uses as working variable also a vector  $\mathbf{v} = (v_1, v_2, \dots, v_n)$ . Interesting is that  $\mathbf{v}$  is used without initialisation, that is its initial values can be arbitrary integer numbers.

The algorithm GARBAGE was proposed by Gábor Monostori [58].

GARBAGE( $n, s$ )

```

01 g ← TRUE
02 for i ← 1 to n
03     do if v[s[i]] < i and s[v[s[i]]] = s[i]
04         then g ← FALSE
05             return g
06         else v[s[i]] ← i
07 return g

```

## 2.6 Definition of algorithm BUCKET

BUCKET handles the array  $Q[1 : m, 1 : m]$  (where  $m = \lceil \sqrt{n} \rceil$ ) and puts the element  $s_i$  into the  $r$ th row of  $Q$ , where  $r = \lceil s_i/m \rceil$  and it tests using linear search whether  $s_j$  appeared earlier in the corresponding row. The elements of the vector  $\mathbf{c} = (c_1, c_2, \dots, c_m)$  are counters, where  $c_j$  ( $1 \leq j \leq m$ ) shows the number of elements of the  $i$ th row.

For the simplicity we suppose that  $n$  is a square.

BUCKET( $n, s$ )

```

01 g ← TRUE
02 m ←  $\sqrt{n}$ 
03 for j ← 1 to m
04     do c[j] ← 1
05 for i ← 1 to n
06     do r ←  $\lceil s[i]/m \rceil m$ 

```

```

07     for j ← 1 to c[r] - 1
08         do if s[i] = Q[r, j]
09             then g ← FALSE
10                 return g
11             else Q[r, c[r]] ← s[i]
12                 c[r] ← c[r] + 1
13 return g

```

### 3 Analysis of the algorithms

#### 3.1 Analysis of algorithm LINEAR

The first algorithm is LINEAR. It writes zero into the elements of an  $n$  length vector  $\mathbf{v} = (v_1, v_2, \dots, v_n)$ , then investigates the elements of the realization sequentially and if  $i_j = k$ , then adds 1 to  $v_k$  and tests whether  $v_k > 0$  signaling a repetition.

In best case LINEAR executes only two comparisons, but the initialization of the vector  $\mathbf{v}$  requires  $\Theta(n)$  assignments. It is called LINEAR, since its running time is  $\Theta(n)$  in best, worst and so also in expected case.

**Theorem 1** *The expected number  $C_{\text{exp}}(n, \text{LINEAR}) = C_L$  of comparisons of LINEAR is*

$$\begin{aligned}
 C_L &= 1 - \frac{n!}{n^n} + \sum_{k=1}^n \frac{n!k^2}{(n-k)!n^{k+1}} \\
 &= \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \kappa(n) - \frac{n!}{n^n},
 \end{aligned}$$

where

$$\kappa(n) = \frac{1}{3} - \sqrt{\frac{\pi n}{2}} + \sum_{k=1}^n \frac{n!k}{(n-k)!n^{k+1}}$$

tends monotonically decreasing to zero when  $n$  tends to infinity.  $n!/n^n$  also tends monotonically decreasing to zero, but their difference  $\delta(n) = \kappa(n) - n!/n^n$  is increasing for  $1 \leq n \leq 8$  and is decreasing for  $n \geq 8$ .

**Theorem 2** *The expected running time  $T_{\text{exp}}(n, \text{LINEAR}) = T_L$  of LINEAR is*

$$T_L = n + \sqrt{2\pi n} + \frac{7}{3} + 2\delta(n),$$

n	$C_L$	$\sqrt{\pi n/2} + 2/3$	$n!/n^n$	$\kappa(n)$	$\delta(n)$
1	1.000000	1.919981	1.000000	0.080019	-0.919981
2	2.000000	2.439121	0.500000	0.060879	-0.439121
3	2.666667	2.837470	0.222222	0.051418	-0.170804
4	3.125000	3.173295	0.093750	0.045455	-0.048295
5	3.472000	3.469162	0.038400	0.041238	+0.002838
6	3.759259	3.736647	0.015432	0.038045	+0.022612
7	4.012019	3.982624	0.006120	0.035515	+0.029395
8	4.242615	4.211574	0.002403	0.033444	+0.031040
9	4.457379	4.426609	0.000937	0.031707	+0.030770
10	4.659853	4.629994	0.000363	0.030222	+0.029859

Table 1: Values of  $C_L$ ,  $\sqrt{\pi n/2} + 2/3$ ,  $n!/n^n$ ,  $\kappa(n)$ , and  $\delta(n) = \kappa(n) - n!/n^n$  for  $n = 1, 2, \dots, 10$

where

$$\delta(n) = \kappa(n) - \frac{n!}{n^n}$$

tends to zero when  $n$  tends to infinity, further

$$\delta(n+1) > \delta(n) \text{ for } 1 \leq n \leq 7 \text{ and } \delta(n+1) < \delta(n) \text{ for } n \geq 8.$$

Table 1 shows some concrete values connected with algorithm LINEAR.

### 3.2 Analysis of algorithm BACKWARD

The second algorithm is BACKWARD. This algorithm is a naive comparison-based one. BACKWARD compares the second ( $i_2$ ), third ( $i_3$ ), ..., last ( $i_n$ ) element of the realization with the previous elements until the first repetition or until the last pair of elements.

The running time of BACKWARD is constant in the best case, but it is quadratic in the worst case.

**Theorem 3** *The expected number  $C_{\text{exp}}(n, \text{BACKWARD}) = C_B$  of comparisons of the algorithm BACKWARD is*

$$C_B = n + \sqrt{\frac{\pi n}{8}} + \frac{2}{3} - \alpha(n),$$

where  $\alpha(n) = \kappa(n)/2 + (n!/n^n)((n+1)/2)$  monotonically decreasing tends to zero when  $n$  tends to  $\infty$ .

Table 2 shows some concrete values characterizing algorithm BACKWARD.

$n$	$C_B$	$n - \sqrt{\pi n/8} + 2/3$	$(n!/n^n)((n+1)/2)$	$\kappa(n)$	$\alpha(n)$
1	0.000000	1.040010	1.000000	0.080019	1.040010
2	1.000000	1.780440	0.750000	0.060879	0.780440
3	2.111111	2.581265	0.444444	0.051418	0.470154
4	3.156250	3.413353	0.234375	0.045455	0.257103
5	4.129600	4.265419	0.115200	0.041238	0.135819
6	5.058642	5.131677	0.054012	0.038045	0.073035
7	5.966451	6.008688	0.024480	0.035515	0.042237
8	6.866676	6.894213	0.010815	0.033444	0.027536
9	7.766159	7.786695	0.004683	0.031707	0.020537
10	8.667896	8.685003	0.001996	0.030222	0.017107

Table 2: Values of  $C_B$ ,  $n - \sqrt{\pi n/8} + 2/3$ ,  $(n!/n^n)((n+1)/2)$ ,  $\kappa(n)$ , and  $\alpha(n) = \kappa(n)/2 + (n!/n^n)((n+1)/2)$  for  $n = 1, 2, \dots, 10$

The next assertion gives the expected running time of algorithm BACKWARD.

**Theorem 4** *The expected running time  $T_{\text{exp}}(n, \text{BACKWARD}) = T_B$  of the algorithm BACKWARD is*

$$T_B = n + \sqrt{\frac{\pi n}{8}} + \frac{4}{3} - \alpha(n),$$

where  $\alpha(n) = \kappa(n)/2 + (n!/n^n)((n+1)/2)$  monotonically decreasing tends to zero when  $n$  tends to  $\infty$ .

### 3.3 Analysis of algorithm FORWARD

FORWARD compares the first ( $s_1$ ), second ( $s_2$ ),  $\dots$ , last but one ( $s_{n-1}$ ) element of the realization with the next elements until the first collision or until the last pair of elements.

Taking into account the number of the necessary comparisons in line 04 of FORWARD, we get  $C_{\text{best}}(n, \text{FORWARD}) = 1 = \Theta(1)$ , and  $C_{\text{worst}}(n, \text{FORWARD}) = B(n, 2) = \Theta(n^2)$ .

The next assertion gives the expected running time.

**Theorem 5** *The expected running time  $T_{\text{exp}}(\mathbf{n}, \text{FORWARD}) = T_{\text{F}}$  of the algorithm FORWARD is*

$$T_{\text{F}} = \mathbf{n} + \Theta(\sqrt{\mathbf{n}}). \quad (1)$$

Although the basic characteristics of FORWARD and BACKWARD are identical, as Table 3 shows, there is a small difference in the expected behaviour.

$\mathbf{n}$	number of sequences	number of good sequences	$C_{\text{F}}$	$C_{\text{W}}$
2	4	2	1.000000	1.000000
3	27	6	2.111111	2.111111
4	256	24	3.203125	3.156250
5	3 125	120	4.264000	4.126960
6	46 656	720	5.342341	5.058642
7	823 543	5 040	6.326760	5.966451
8	16 777 216	40 320	7.342926	6.866676
9	387 420 489	362 880	8.354165	7.766159

Table 3: Values of  $\mathbf{n}$ , the number of possible input sequences, number of good sequences, expected number of comparisons of FORWARD ( $C_{\text{F}}$ ) and expected number of comparisons of BACKWARD ( $C_{\text{W}}$ ) for  $\mathbf{n} = 2, 3, \dots, 9$

### 3.4 Analysis of algorithm TREE

TREE builds a random search tree from the elements of the realization and finishes the construction of the tree if it finds the following element of the realization in the tree (then the realization is not good) or it tested the last element too without a collision (then the realization is good).

The worst case running time of TREE appears when the input contains different elements in increasing or decreasing order. Then the result is  $\Theta(\mathbf{n}^2)$ . The best case is when the first two elements of  $\mathbf{s}$  are equal, so  $C_{\text{best}}(\mathbf{n}, \text{TREE}) = 1 = \Theta(1)$ .

Using the known fact that the expected height of a random search tree is  $\Theta(\lg \mathbf{n})$  we can get that the order of the expected running time is  $\sqrt{\mathbf{n}} \lg \mathbf{n}$ .

**Theorem 6** *The expected running time  $T_{\text{T}}$  of TREE is*

$$T_{\text{T}} = \Theta(\sqrt{\mathbf{n}} \lg \mathbf{n}). \quad (2)$$



n	number of good inputs	number of comparisons	number of assignments
1	100 000.000000	0.000000	1.000000
2	49 946.000000	1.000000	1.499460
3	22 243.000000	2.038960	1.889900
4	9 396.000000	2.921710	2.219390
5	3 723.000000	3.682710	2.511409
6	1 569.000000	4.352690	2.773160
7	620.000000	4.985280	3.021820
8	251.000000	5.590900	3.252989
9	104	6.148550	3.459510
10	33	6.704350	3.663749
11	17	7.271570	3.860450
12	3	7.779950	4.039530
13	3	8.314370	4.214370
14	0	8.824660	4.384480
15	2	9.302720	4.537880
16	0	9.840690	4.716760
17	0	10.287560	4.853530
18	0	10.719770	4.989370
19	0	11.242740	5.147560
20	0	11.689660	5.279180

Table 4: Values of  $n$ , number of good inputs, number of comparisons, number of assignments of TREE for  $n = 1, 2, \dots, 10$

Table 4 shows some results of the simulation experiments (the number of random input sequences is 100 000 in all cases).

Using the method of the smallest squares to find the parameters of the formula  $\alpha\sqrt{n}\log_2 n$  we received the following approximation formula for the expected number of comparisons:

$$C_{\text{exp}}(n, \text{TREE}) = 1.245754\sqrt{n}\log_2 n - 0.273588.$$

### 3.5 Analysis of algorithm GARBAGE

This algorithm is similar to LINEAR, but it works without the setting zeros into the elements of a vector requiring linear amount of time.

Beside the cycle variable  $i$  GARBAGE uses as working variable also a vector

$\mathbf{v} = (v_1, v_2, \dots, v_n)$ . Interesting is that  $\mathbf{v}$  is used without initialisation, that is its initial values can be arbitrary integer numbers.

The worst case running time of GARBAGE appears when the input contains different elements and the garbage in the memory does not help, but even in this case  $C_{\text{worst}}(\mathfrak{n}, \text{GARBAGE}) = \Theta(\mathfrak{n})$ . The best case is when the first element is repeated in the input and the garbage helps to find a repetition of the first element of the input. Taking into account this case we get  $C_{\text{best}}(\mathfrak{n}, \text{GARBAGE}) = \Theta(1)$ .

According to the next assertion the expected running time is  $\Theta(\sqrt{\mathfrak{n}})$ .

**Lemma 7** *The expected running time of GARBAGE is*

$$T_{\text{exp}}(\mathfrak{n}, \text{GARBAGE}) = \Theta(\sqrt{\mathfrak{n}}). \quad (3)$$

### 3.6 Analysis of algorithm BUCKET

Algorithm BUCKET divides the interval  $[1, \mathfrak{n}]$  into  $\mathfrak{m} = \lceil \sqrt{\mathfrak{n}} \rceil$  subintervals  $I_1, I_2, \dots, I_{\mathfrak{m}}$ , where  $I_k = [(k-1)\mathfrak{m} + 1, k\mathfrak{m}]$ , and assigns a bucket  $B_k$  to interval  $I_k$ . BUCKET sequentially puts the input elements  $i_j$  into the corresponding bucket: if  $i_j$  belongs to the interval  $I_k$  then it checks whether  $i_j$  is contained in  $B_k$  or not. BUCKET works up to the first repetition. (For the simplicity we suppose that  $\mathfrak{n} = \mathfrak{m}^2$ .)

In best case BUCKET executes only 1 comparison, but the initialization of the buckets requires  $\Theta(\sqrt{\mathfrak{n}})$  assignments, therefore the best running time is also  $\sqrt{\mathfrak{n}}$ . The worst case appears when the input is a permutation. Then each bucket requires  $\Theta(\mathfrak{n})$  comparisons, so the worst running time is  $\Theta(\mathfrak{n}\sqrt{\mathfrak{n}})$ .

**Lemma 8** *Let  $b_j$  ( $j = 1, 2, \dots, \mathfrak{m}$ ) be a random variable characterising the number of elements in the bucket  $B_j$  at the moment of the first repetition. Then*

$$E\{b_j\} = \sqrt{\frac{\pi}{2}} - \mu(\mathfrak{n})$$

for  $j = 1, 2, \dots, \mathfrak{m}$ , where

$$\mu(\mathfrak{n}) = \frac{1}{3\sqrt{\mathfrak{n}}} - \frac{\kappa(\mathfrak{n})}{\sqrt{\mathfrak{n}}},$$

and  $\mu(\mathfrak{n})$  tends monotonically decreasing to zero when  $\mathfrak{n}$  tends to infinity.

Table 5 contains some concrete values connected with  $E\{b_1\}$ .

$n$	$E\{b_1\}$	$\sqrt{\pi/2}$	$1/(3\sqrt{n})$	$\kappa(n)/\sqrt{n}$	$\mu(n)$
1	1.000000	1.253314	0.333333	0.080019	0.253314
2	1.060660	1.253314	0.235702	0.043048	0.192654
3	1.090055	1.253314	0.192450	0.029686	0.162764
4	1.109375	1.253314	0.166667	0.022727	0.143940
5	1.122685	1.253314	0.149071	0.018442	0.130629
6	1.132763	1.253314	0.136083	0.015532	0.120551
7	1.147287	1.253314	0.125988	0.013423	0.112565
8	1.147287	1.253314	0.117851	0.011824	0.106027
9	1.152772	1.253314	0.111111	0.010569	0.100542
10	1.157462	1.253314	0.105409	0.009557	0.095852

Table 5: Values of  $E\{b_1\}$ ,  $\sqrt{\pi/2}$ ,  $1/(3\sqrt{n})$ ,  $\kappa(n)/\sqrt{n}$ , and  $\mu(n) = 1/(3\sqrt{n}) - \kappa(n)/\sqrt{n}$  of BUCKET for  $n = 1, 2, \dots, 10$

**Lemma 9** *Let  $f_n$  be a random variable characterising the number of comparisons executed in connection with the first repeated element. Then*

$$E\{f_n\} = 1 + \sqrt{\frac{\pi}{8}} - \eta(n),$$

where

$$\eta(n) = \frac{\frac{1}{3} + \sqrt{\frac{\pi}{8}} - \frac{\kappa(n)}{2}}{\sqrt{n} + 2},$$

and  $\eta(n)$  tends monotonically decreasing to zero when  $n$  tends to infinity.

**Theorem 10** *The expected number  $C_{\text{exp}}(n, \text{BUCKET}) = C_B$  of comparisons of algorithm BUCKET in 1 bucket is*

$$C_B = \sqrt{n} + \frac{1}{3} - \sqrt{\frac{\pi}{8}} + \rho(n),$$

where

$$\rho(n) = \frac{5/6 - \sqrt{9\pi/8} - 3\kappa(n)/2}{\sqrt{n} + 1}$$

tends to zero when  $n$  and tends to infinity.

Index and Algorithm	$C_{\text{best}}(n)$	$C_{\text{worst}}(n)$	$C_{\text{exp}}(n)$
1. LINEAR	$\Theta(1)$	$\Theta(n)$	$\Theta(\sqrt{n})$
2. BACKWARD	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n)$
3. FORWARD	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n)$
4. TREE	$\Theta(1)$	$\Theta(n^2)$	$\Theta(\sqrt{n} \lg n)$
5. GARBAGE	$\Theta(1)$	$\Theta(n)$	$\Theta(\sqrt{n})$
6. BUCKET	$\Theta(\sqrt{n})$	$\Theta(n\sqrt{n})$	$\Theta(\sqrt{n})$

Table 6: The number of necessary comparisons of the investigated algorithms in best, worst and expected cases

**Theorem 11** *The expected running time  $T_B(n, \text{BUCKET}) = T_B$  of BUCKET is*

$$T_B = \left(3 + 3\sqrt{\frac{\pi}{2}}\right) \sqrt{n} + \sqrt{\frac{25\pi}{8}} + \phi(n),$$

where

$$\phi(n) = 3\kappa(n) - \rho(n) - 3\eta(n) - \frac{n!}{n^n} - \frac{3\sqrt{\pi/8} - 1/3 - 3\kappa(n)/2}{\sqrt{n} + 1}$$

and  $\phi(n)$  tends to zero when  $n$  tends to infinity.

It is worth to remark that simulation experiments of B. Novák [62] show that the expected running time of GARBAGE is a few percent better, then the expected running time of BUCKET.

## 4 Summary

Table 6 contains the number of necessary comparisons in best, worst and expected cases for all investigated algorithms.

Table 7 contains the running time in best, worst and expected cases for all investigated algorithms.

**Acknowledgements.** The authors thank Tamás F. Móri [60] for proving Lemma 8 and 9 and Péter Burcsi [10] for useful information on references, both are teachers of Eötvös Loránd University.

The European Union and the European Social Fund have provided financial support to the project under the grant agreement no. TÁMOP 4.2.1/B-09/1/KMR-2010-0003.

Index and Algorithm	$T_{\text{best}}(n)$	$T_{\text{worst}}(n)$	$T_{\text{exp}}(n)$
1. LINEAR	$\Theta(n)$	$\Theta(n)$	$n + \Theta(\sqrt{n})$
2. BACKWARD	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n)$
3. FORWARD	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n)$
5. TREE	$\Theta(1)$	$\Theta(n^2)$	$\Theta(\sqrt{n} \lg n)$
6. GARBAGE	$\Theta(1)$	$\Theta(n)$	$\Theta(\sqrt{n})$
7. BUCKET	$\Theta(\sqrt{n})$	$\Theta(n\sqrt{n})$	$\Theta(\sqrt{n})$

Table 7: The running times of the investigated algorithms in best, worst and expected cases

## References

- [1] P. Adams, D. Bryant, M. Buchanan, Completing partial Latin squares with two filled rows and two filled columns, *Electron. J. Combin.* **15**, 1 (2008), Research paper 56, 26 p. [⇒ 136](#)
- [2] M.-C. Anisiu, A. Iványi, Two-dimensional arrays with maximal complexity, *Pure Math. Appl. (P.U.M.A.)* **17**, 3–4 (2006) 197–204. [⇒ 136](#)
- [3] C. Arcos, G. Brookfield, M. Krebs, Mini-sudokus and groups, *Math. Mag.* **83**, 2 (2010) 111–122. [⇒ 136](#)
- [4] R. A. Bailey, R. Cameron, P. J. Connelly, Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes, *American Math. Monthly* **115**, 5 (2008) 383–404. [⇒ 135](#), [136](#)
- [5] W. U. Behrens, Feldversuchsanordnungen mit verbessertem Ausgleich der Bodenunterschiede, *Zeitschrift für Landwirtschaftliches Versuchs- und Untersuchungswesen*, **2** (1956) 176–193. [⇒ 135](#)
- [6] D. Berthier, Unbiased statistics of a constraint satisfaction problem – a controlled-bias generator, in: S. Tarek et al. (eds.), *Innovations in computing sciences and software engineering*, Proc. Second International Conference on Systems, Computing Sciences and Software Engineering (SCSS'2009, December 4–12, 2009, Dordrecht). Springer, Berlin, 2010. pp. 91–97. [⇒ 136](#)
- [7] S. Brett, G. Hurlbert, B. Jackson, Preface [Generalisations of de Bruijn cycles and Gray codes], *Discrete Math.*, **309**, 17 (2009) 5255–5258. [⇒ 136](#)

- 
- [8] A. A. Bruen, R. A. Mollin, Cryptography and shift registers, *Open Math. J.*, **2** (2009) 16–21. [⇒ 136](#)
- [9] H. L. Buchanan, M. N. Ferencak, On completing Latin squares, *J. Combin. Math. Combin. Comput.*, **34** (2000) 129–132. [⇒ 136](#)
- [10] P. Burcsi, Personal communication. Budapest, March 2009. [⇒ 146](#)
- [11] G. J. Burnett, E. G. Coffman, Jr., Combinatorial problem related to interleaved memory systems, *J. ACM*, **20**, 1 (1973) 39–45. [⇒ 136](#)
- [12] P. J. Cameron, *Sudoku – an alternative history*, Talk to the Archimedean, Queen Mary University of London, February 2007. [⇒ 136](#)
- [13] A. Carlos, G. Brookfield, M. Krebs, Mini-sudokus and groups, *Math. Mag.*, **83**, 2 (2010) 111–122. [⇒ 136](#)
- [14] J. Carmichael, K. Schloeman, M. B. Ward, Cosets and Cayley-sudoku tables, *Math. Mag.*, **83**, 2 (2010) 130–139. [⇒ 136](#)
- [15] Ch.-Ch. Chang, P.-Y. Lin, Z.-H. Wang, M.-Ch. Li, A sudoku-based secret image sharing scheme with reversibility, *J. Commun.*, **5**, 1 (2010) 5–12. [⇒ 136](#)
- [16] Z. Chen, Heuristic reasoning on graph and game complexity of sudoku, [ARXIV.org](#), 2010. 6 p. [⇒ 136](#)
- [17] Y.-F. Chien, W.-K. Hon, Cryptographic and physical zero-knowledge proof: From sudoku to nonogram, in: P. Boldi (ed.), *Fun with Algorithms*, (5th International Conference, FUN 2010, Ischia, Italy, June 2–4, 2010.) Springer, Berlin, 2010, *Lecture Notes in Comput. Sci.*, **6099** (2010) 102–112. [⇒ 136](#)
- [18] J. Cooper, C. Heitsch, The discrepancy of the lex-least de Bruijn sequence, *Discrete Math.*, **310**, 6–7 (2010), 1152–1159. [⇒ 136](#)
- [19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Third edition. The MIT Press, Cambridge, 2009. [⇒ 136](#)
- [20] J. F. Crook, A pencil-and-paper algorithm for solving sudoku puzzles, *Notices Amer. Math. Soc.*, **56** (2009) 460–468. [⇒ 136](#)
- [21] G. Dahl, Permutation matrices related to sudoku, *Linear Algebra Appl.*, **430** (2009), 2457–2463. [⇒ 136](#)

- 
- [22] J. Dénes, A. D. Keedwell, *Latin squares. New developments in the theory and applications*, North-Holland, Amsterdam, 1991. ⇒ 136
- [23] T. Easton, R. G. Parker, On completing Latin squares, *Discrete Appl. Math.*, **113**, 2–3 (2001) 167–181. ⇒ 136
- [24] C. H. Elzinga, S. Rahmann, H. Wung, Algorithms for subsequence combinatorics, *Theor. Comput. Sci.*, **409**, 3 (2008) 394–404. ⇒ 136
- [25] C. H. Elzinga, Complexity of categorial time series, *Sociological Methods & Research*, **38**, 3 (2010) 463–481. ⇒ 136
- [26] M. Erickson, *Pearls of discrete mathematics*, Discrete Mathematics and its Applications, CRC Press, Boca Raton, FL, 2010. ⇒ 136
- [27] R. Euler, On the completability of incomplete Latin squares, *European J. Combin.* **31** (2010) 535–552. ⇒ 136
- [28] S. Ferenczi, Z. Kása, Complexity for finite factors of infinite sequences, *Theoret. Comput. Sci.* **218**, 1 (1999) 177–195. ⇒ 136
- [29] R. Fontana, F. Rapallo, M. P. Rogantin, Indicator function and sudoku designs, in: P. Gibilisco, E. Ricco-magno, M. P. Rogantin, H. P. Wynn (eds.) *Algebraic and Geometric Methods in Statistics*, pp. 203–224. Cambridge University Press, Cambridge, 2010. ⇒ 136
- [30] R. Fontana, F. Rapallo, M. P. Rogantin, Markov bases for sudoku grids. Rapporto interno N. 4, marzo 2010, Politecnico di Torino. ⇒ 136
- [31] A. F. Gabor, G. J. Woeginger, How \*not\* to solve a Sudoku. *Operation Research Letters*, **38**, 6 (2010) 582–584. ⇒ 136
- [32] I. Hajirasouliha, H. Jowhari, R. Kumar, R. Sundaram, On completing Latin squares, *Lecture Notes in Comput. Sci.*, **4393** (2007), 524–535. Springer, Berlin, 2007. ⇒ 136
- [33] H. Hellerman, *Digital computer system principles*. Mc Graw Hill, New York, 1967. ⇒ 136
- [34] A. Heppes, P. Révész, A new generalization of the concept of latin squares and orthogonal latin squares and its application to the design of experiments (in Hungarian), *Magyar Tud. Akad. Mat. Int. Közl.*, **1** (1956) 379–390. ⇒ 135

- 
- [35] M. Horváth, A. Iványi, Growing perfect cubes, *Discrete Math.*, **308**, 19 (2008) 4378–4388. [⇒ 136](#)
- [36] A. Iványi, On the **d-complexity** of words, *Ann. Univ. Sci. Budapest. Sect. Comput.* **8** (1987) 69–90 (1988). [⇒ 136](#)
- [37] A. Iványi, Construction of infinite de Bruijn arrays, *Discrete Appl. Math.* **22**, 3 (1988/89), 289–293. [⇒ 136](#)
- [38] A. Iványi, Construction of three-dimensional perfect matrices, (Twelfth British Combinatorial Conference, Norwich, 1989). *Ars Combin.* **29C** (1990) 33–40. [⇒ 136](#)
- [39] A. Iványi, I. Kátai, Estimates for speed of computers with interleaved memory systems, *Ann. Univ. Sci. Budapest. Sect. Math.*, **19** (1976) 159–164. [⇒ 136](#)
- [40] A. Iványi, I. Kátai, Processing of random sequences with priority, *Acta Cybern.* **4**, 1 (1978/79) 85–101. [⇒ 136](#)
- [41] A. Iványi, I. Kátai, Quick testing of random variables, *Proc. ICAI'2010* (Eger, January 27–30, 2010). To appear. [⇒ 136](#)
- [42] A. Iványi, I. Kátai, Testing of uniformly distributed vectors, in: *Abstracts János Bolyai Memorial Conference*, (Budapest, August 28–30, 2010), p. 47. [⇒ 136](#)
- [43] A. Iványi, I. Kátai, Testing of random matrices, *Ann. Univ. Sci. Budapest. Sect. Comput.* (submitted). [⇒ 136](#)
- [44] A. Iványi, B. Novák, Testing of random sequences by simulation, in: *Abstracts 8th MACS* (Komárno, July 14–17, 2010). [⇒ 136](#)
- [45] A. Iványi, Z. Tóth, Existence of de Bruijn words, *Second Conference on Automata, Languages and Programming Systems* (Salgótarján, 1988), 165–172, DM, 88-4, Karl Marx Univ. Econom., Budapest, 1988. [⇒ 136](#)
- [46] I. Kanaana, B. Ravikumar, Row-filled completion problem for sudoku, *Util. Math.* **81** (2010) 65–84. [⇒ 136](#)
- [47] Z. Karimi-Dehkordi, K. Zamanifar, A. Baraani-Dastjerdi, N. Ghasem-Aghae, **Sudoku** using parallel simulated annealing, in: Y. Tan et al.



- (eds.), *Advances in Swarm Intelligence* (Proc. First International Conference, ICSI 2010, Beijing, China, June 12–15, 2010, Part II. *Lecture Notes in Comput. Sci.*, **6146** (2010) 461–467, Springer, Berlin, 2010. ⇒ 136
- [48] Z. Kása, Computing the  $d$ -complexity of words by Fibonacci-like sequences, *Studia Univ. Babeş-Bolyai Math.* **35**, 3 (1990) 49–53. ⇒ 136
- [49] Z. Kása, *Pure Math. Appl.* On the  $d$ -complexity of strings, (*P.U.M.A.*) **9**, 1–2 (1998) 119–128. ⇒ 136
- [50] Z. Kása, Super- $d$ -complexity of finite words, *Proc. 8th Joint Conference on Mathematics and Computer Science*, (Komárno, Slovakia, July 14–17), 2010, To appear. ⇒ 136
- [51] A. D. Keedwell, Constructions of complete sets of orthogonal diagonal sudoku squares, *Australas. J. Combin.* **47** (2010) 227–238. ⇒ 136
- [52] D. E. Knuth, *The art of computer programming, Vol. 1. Fundamental algorithms* (third edition). Addison–Wesley, Reading, MA, 1997. ⇒ 136
- [53] J. S. Kuhl, T. Denley, On a generalization of the Evans conjecture, *Discrete Math.* **308**, 20 (2008), 4763–4767. ⇒ 136
- [54] S. R. Kumar S., A. Russell, R. Sundaram, Approximating Latin square extensions, *Algorithmica* **24**, 2 (1999) 128–138. ⇒ 136
- [55] L. Lorch, Mutually orthogonal families of linear sudoku solutions, *J. Aust. Math. Soc.*, **87**, 3 (2009) 409–420. ⇒ 136
- [56] M. Matamala, F. Moreno, Minimum Eulerian circuits and minimum de Bruijn sequences, *Discrete Math.*, **309**, 17 (2009) 5298–5304. ⇒ 136
- [57] H.-D. Mo, R.-G. Xu, Sudoku square – a new design in field, *Acta Agronomica Sinica*, **34**, 9 (2008) 1489–1493. ⇒ 135
- [58] G. Monostori, Personal communication, Budapest, May 2010. ⇒ 138
- [59] T. K. Moon, J. H. Gunther, J. J. Kupin, Sinkhorn solves sudoku, *IEEE Trans. Inform. Theory*, **55**, 4 (2009) 1741–1746. ⇒ 136
- [60] T. Móri, Personal communication, Budapest, April 2010. ⇒ 146

- 
- [61] P. K. Newton, S. A. deSalvo, The Shannon entropy of sudoku matrices, *Proc. R. Soc. Lond. Ser. A, Math. Phys. Eng. Sci.* **466** (2010) 1957–1975. [⇒ 136](#)
- [62] B. Novák, *Analysis of sudoku algorithms* (in Hungarian), MSc thesis, Eötvös Loránd University, [Fac. of Informatics](#), Budapest, 2010. [⇒ 136, 146](#)
- [63] L.-D. Öhman, A note on completing Latin squares, *Australas. J. Combin.*, **45** (2009) 117–123. [⇒ 136](#)
- [64] R. M. Pedersen, T. L. Vis, Sets of mutually orthogonal sudoku Latin squares. *College Math. J.*, **40**, 3 (2009) 174–180. [⇒ 136](#)
- [65] R. Penne, A note on certain de Bruijn sequences with forbidden subsequences, *Discrete Math.*, **310**, 4 (2010) 966–969. [⇒ 136](#)
- [66] J. S. Provan, Sudoku: strategy versus structure, *Amer. Math. Monthly*, **116**, 8 (2009), 702–707. [⇒ 136](#)
- [67] T. Sander, Sudoku graphs are integral, *Electron. J. Combin.*, **16**, 1 (2009), Note 25, 7 p. [⇒ 136](#)
- [68] Y. Sato, H. Inoue, Genetic operations to solve sudoku puzzles, *Proc. 12th Annual Conference on Genetic and Evolutionary Computation GECCO'10*, July 7–11, 2010, Portland, OR, pp. 2111–21012. [⇒ 136](#)
- [69] M. J. Soottile, T. G. Mattson, C. E. Rasmussen, *Introduction to concurrency in programming languages*, Chapman & Hall/CRC Computational Science Series. [CRC Press](#), Boca Raton, FL, 2010. [⇒ 136](#)
- [70] D. Thom, *SUDOKU ist NP-vollständig*, PhD Dissertation, Stuttgart, 2007. [⇒ 136](#)
- [71] O. G. Troyanskaya, O. Arbell, Y. Koren, G. M. Landau, A. Bolshoy, Sequence complexity profiles of prokaryotic genomic sequences: A fast algorithm for calculating linguistic complexity, *Bioinformatics*, **18**, 5 (2002) 679–688. [⇒ 136](#)
- [72] E. R. Vaughan, The complexity of constructing gerechte designs, *Electron. J. Combin.*, **16**, 1 (2009), paper R15, 8 p. [⇒ 135, 136](#)
- [73] X. Xu, Y. Cao, J.-M. Xu, Y. Wu, Feedback numbers of de Bruijn digraphs, *Comput. Math. Appl.*, **59**, 4 (2010), 716–723. [⇒ 136](#)

- [74] C. Xu, W. Xu, The model and algorithm to estimate the difficulty levels of sudoku puzzles, *J. Math. Res.* **11**, 2 (2009), 43–46. ⇒ [136](#)
- [75] W. Zhang, S. Liu, H. Huang, An efficient implementation algorithm for generating de Bruijn sequences, *Computer Standards & Interfaces*, **31**, 6 (2009) 1190–1191. ⇒ [136](#)

*Received: August 20, 2010 • Revised: October 15, 2010*