

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**  
**INFORMATIKOS FAKULTETAS**

**ALGORITMŲ SUDARYMAS IR ANALIZĖ**

Laboratorinis darbas nr. 1

Atliko:

IFF – 7/2 gr. studentas Giedrius Kristinaitis


**KAUNAS, 2019**


## 1. Užduotis


Darbo užduotis – palyginti rikiavimo algoritmus (Insertion sort ir Merge sort) masyve ir susietajame sąrašas dviem atvejais: kai duomenų struktūra realizuota operatyvinėje atmintyje ir diskinėje atmintyje. Taip pat reikia atlikti paieškos analizę maišos lentelės su sąrašais atveju. Suskaičiuoti algoritmų sudėtingumą remiantis programos tekstu ir padaryti išvadas.

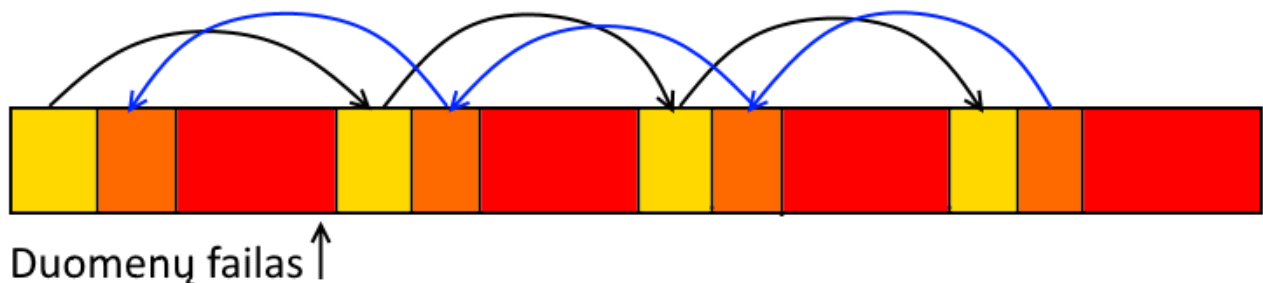
## 2. Susietasis sąrašas išorinėje atmintyje

Diskinėje atmintyje realizuotas susietasis sąrašas atrodo taip:

 - duomenų blokas (double)

 - nuoroda į kitą mazgą (int)

 - nuoroda į prieš tai esantį mazgą (int)



Elemento nuskaitymas realizuotas šiuo kodu:

```
public double next() {  
    if (current == -1) {  
        current = first;  
    } else {  
        try {  
            data.seek(current);  
  
            current = data.readInt();  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
  
    return get();  
}
```

```

public double get() {
    try {
        data.seek(current + 8);

        return data.readDouble();
    } catch (IOException ex) {
        ex.printStackTrace();
    }

    return 0;
}

```

Čia data – duomenų failas, current – esamas sąrašo mazgas.

### 3. Insertion sort analizė

Insertion sort – tai rikiavimo algoritmas, kai duomenys surikiuojami iš eilės imant kiekvieną elementą ir jį įdedant į reikiamą vietą.

Teorinis algoritmo sudėtingumas:  $O(n^2)$

Suskaiciuojame algoritmo sudėtingumą pagal programos tekstą (array.length() pavadinkime n):

	Kaina	Kartai
<pre> public void sortArray(DataArray array) {     for (int i = 1; i &lt; array.length(); i++) {         int a = i;         int b = i - 1;          while (b &gt;= 0 &amp;&amp; array.get(b) &gt; array.get(a)) {             array.swap(a, b);              a--;             b--;         }     } } </pre>	C1	n+1
	C2	n
	C3	n
	C4	$\sum_0^n \sum_0^i 1$
	C5	$\sum_0^n \sum_0^{i-1} 1$
	C6	$\sum_0^n \sum_0^{i-1} 1$
	C7	$\sum_0^n \sum_0^{i-1} 1$

$$\begin{aligned}
 T(n) &= C1n + C1 + C2n + C3n + C4n^2 + C5n(n-1) + C6n(n-1) + \\
 C7n(n-1) &= n(C1 + C2 + C3) + C1 + n(n-1)(C4 + C5 + C6 + C7) + C4 = nC + \\
 n(n-1)C1 + C2 &= \theta(n^2)
 \end{aligned}$$

Suskaiciuotas algoritmo sudėtingumas sutampa su teoriniu sudėtingumu.

Masyvo operatyvinėje atmintyje eksperimento rezultatai:

Masyvo dydis	Vykdyimo laikas (ms)
1000	11
2000	16
4000	23
8000	63
16000	223
32000	808
64000	3219

Masyvo diskinėje atmintyje eksperimento rezultatai:

Masyvo dydis	Vykdyimo laikas (ms)
10	5
20	22
40	48
80	165
160	655
320	2431
640	10659

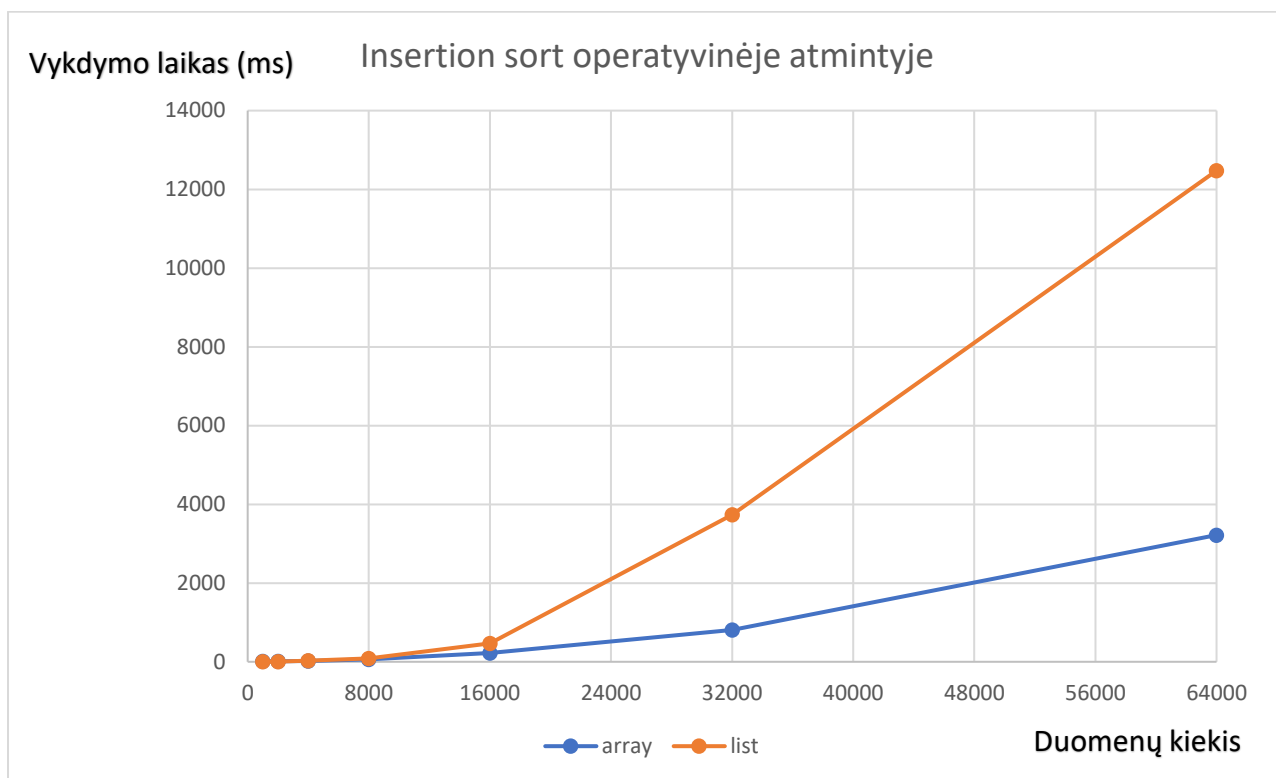
Sąrašo operatyvinėje atmintyje eksperimento rezultatai:

Sąrašo dydis	Vykdyimo laikas (ms)
1000	4
2000	8
4000	27
8000	86
16000	464
32000	3736
64000	12475

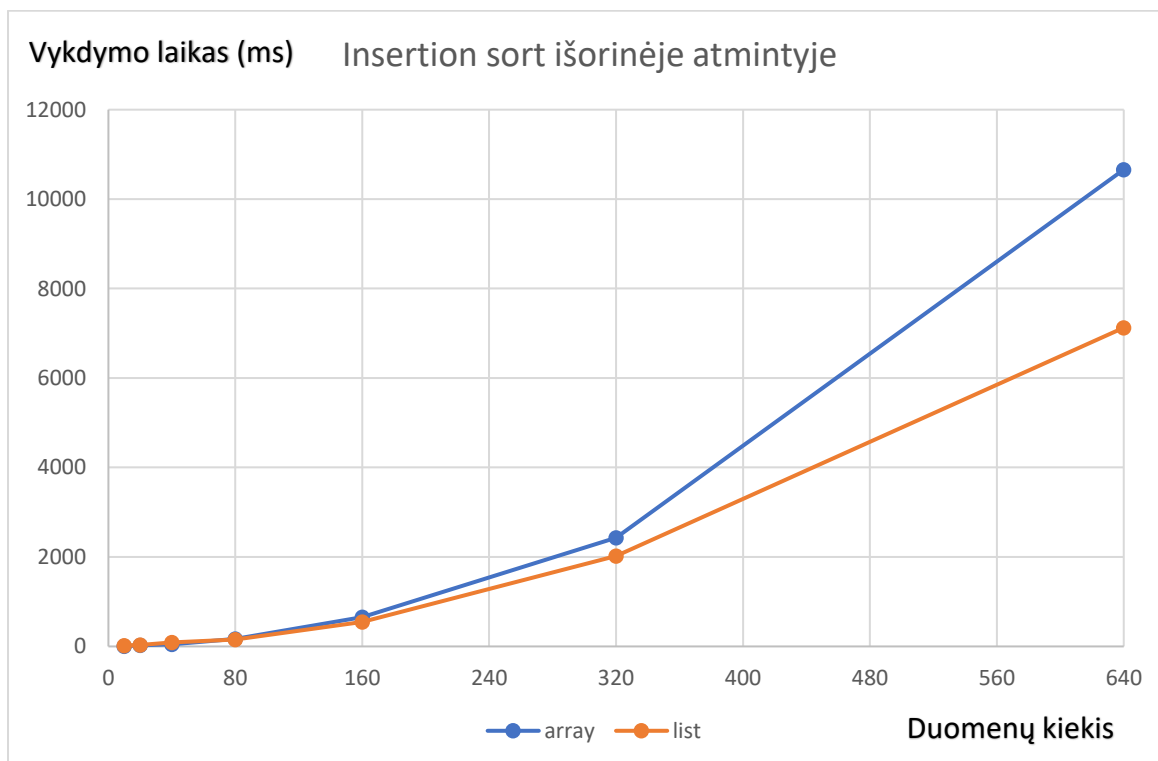
Sąrašo diskinėje atmintyje eksperimento rezultatai:

Sąrašo dydis	Vykdyimo laikas (ms)
10	19
20	32
40	86
80	151
160	546
320	2019
640	7125

## Insertion sort operatyvinėje atmintyje masyvo ir sąrašo palyginimas:



## Insertion sort diskinėje atmintyje masyvo ir sąrašo palyginimas:



### 4. Merge sort analizė

Merge sort – tai skaldyk ir valdyk rikiavimo algoritmas, kai rikiuojami duomenys rekursiškai išskaidomi į mažesnius masyvus ir po to sujungiami reikiama tvarka.

Teorinis algoritmo sudėtingumas:  $O(n \cdot \log n)$

Suskaiciuojame algoritmo sudėtingumą pagal programos tekstą (argumentą right pavadinkime n, nes right yra masyvo ilgis, merge metodą pažymėkime M(n)):

	Kaina	Kartai
<pre>private void sort(DataArray array, int left, int right) {     if (right &gt; left) {         int middle = (left + right) / 2;          sort(array, left, middle);         sort(array, middle + 1, right);          merge(array, left, right, middle);     } }</pre>	C1	1
	C2	1
	T(n/2)	1
	T(n/2)	1
<pre>private void merge(DataArray array, int left, int right, int middle) {     int firstLength = middle - left + 1;     int secondLength = right - middle;      double[] tempLeft = new double[firstLength];     double[] tempRight = new double[secondLength];      for (int i = 0; i &lt; firstLength; i++) {         tempLeft[i] = array.get(left + i);     }      for (int i = 0; i &lt; secondLength; i++) {         tempRight[i] = array.get(middle + 1 + i);     }      int arrayIndex = left;      int i = 0;     int j = 0;      while (i &lt; firstLength &amp;&amp; j &lt; secondLength) {         if (tempLeft[i] &lt;= tempRight[j]) {             array.set(arrayIndex, tempLeft[i++]);         } else {             array.set(arrayIndex, tempRight[j++]);         }          arrayIndex++;     }      while (i &lt; firstLength) {         array.set(arrayIndex++, tempLeft[i++]);     } }</pre>	C3	1
	C4	1
	C5	1
	C6	1
	C7	firstLength+1
	C8	firstLength
	C9	secLength+1
	C10	secLength
	C11	1
	C12	1
	C13	1
	C14	fLen+sLen+1
	C15	fLen+sLen
	C16	firstLength
	C17	secLength
	C18	fLen+sLen
	C19	2
	C20	1

<pre> while (j &lt; secondLength) {     array.set(arrayIndex++, tempRight[j++]); } </pre>	C21 C22	2 1
---	------------	--------

Skačiuojame merge metodo sudėtingumą (kadangi firstLength + secondLength yra sujungto masyvo ilgis, pavadinkime jį m):

$$\begin{aligned}
M(m) &= C3 + C4 + C5 + C6 + C7 \left( \left( \frac{m}{2} \right) + 1 \right) + C8 \left( \frac{m}{2} \right) + C9 \left( \frac{m}{2} + 1 \right) + C10 \left( \frac{m}{2} \right) + C11 \\
&\quad + C12 + C13 + C14(m + 1) + C15m + C16 \left( \frac{m}{2} \right) + C17 \left( \frac{m}{2} \right) + C18m \\
&\quad + 2C19 + C20 + 2C21 + C22 \\
&= \left( \frac{m}{2} \right) (C7 + C8 + C9 + C10 + C16 + C17) + m(C14 + C15 + C18) + C3 \\
&\quad + C4 + C5 + C6 + C7 + C9 + C14 + C11 + C12 + C13 + 2C19 + C20 \\
&\quad + 2C21 + C22 = \left( \frac{m}{2} \right) C + mC1 + C2
\end{aligned}$$

Skačiuojame T(n):

$$\begin{aligned}
T(n) &= C1 + C2 + 2T \left( \frac{n}{2} \right) + \left( \frac{m}{2} \right) C3 + mC4 + C5 = 2T \left( \frac{n}{2} \right) + \left( \frac{m}{2} \right) C + mC1 + C2 \\
&= \theta(n * \log n)
\end{aligned}$$

Suskaičiuotas algoritmo sudėtingumas sutapo su teoriniu sudėtingumu.

Masyvo operatyvinėje atmintyje eksperimento rezultatai:

Masyvo dydis	Vykdymo laikas (ms)
10000	8
30000	9
90000	27
270000	74
810000	255
2430000	793
7290000	2536



Masyvo diskinėje atmintyje eksperimento rezultatai:

Masyvo dydis	Vykdyimo laikas (ms)
100	810
200	1742
400	2950
800	5924
1600	12337
3200	24691
6400	52760

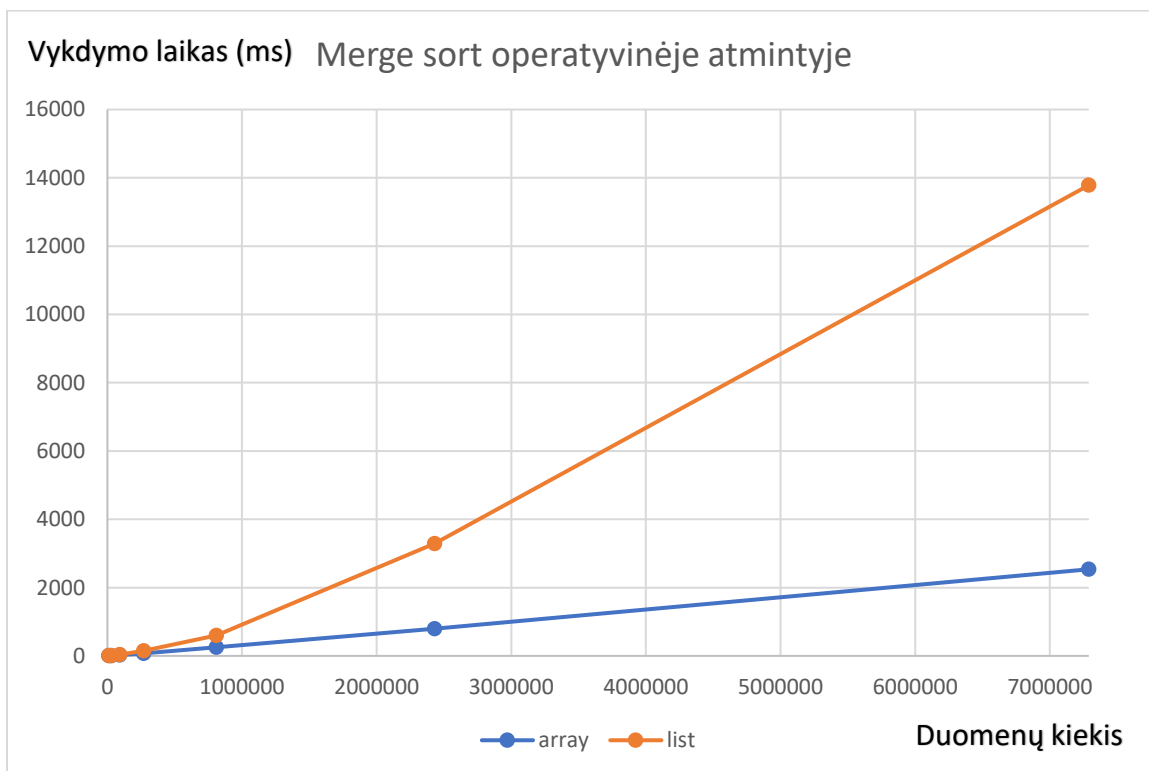
Sąrašo operatyvinėje atmintyje eksperimento rezultatai:

Sąrašo dydis	Vykdyimo laikas (ms)
10000	5
30000	7
90000	40
270000	153
810000	601
2430000	3292
7290000	13784

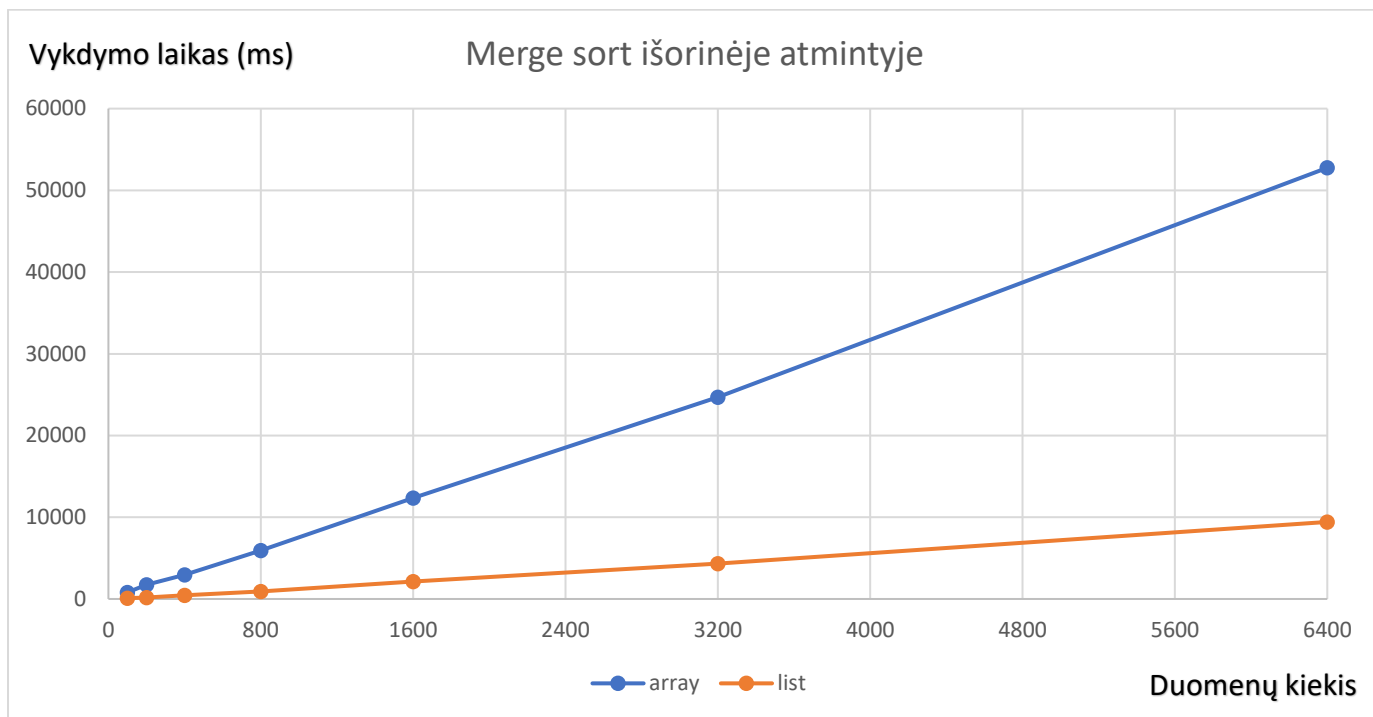
### Sąrašo diskinėje atmintyje eksperimento rezultatai:

Sąrašo dydis	Vykdyimo laikas (ms)
100	97
200	174
400	445
800	926
1600	2125
3200	4313
6400	9425

### Merge sort operatyvinėje atmintyje masyvo ir sąrašo palyginimas:



## Merge sort išorinėje atmintyje masyvo ir sąrašo palyginimas:



### 5. Paieškos analizė

Paieška buvo atlikta maišos lentelėje su sąrašais operatyvinėje ir diskinėje atmintyje.

Buvo ieškomas kiekvienas maišos lentelės elementas.

Paieškos maišos lentelėje vidutinis teorinis sudėtingumas:  $\Theta(1)$

Paieškos maišos lentelėje blogiausio atvejo teorinis sudėtingumas:  $O(n)$

Skaiciuojame paieškos sudėtingumą pagal programos kodą (getNodeWithKey metoda pažymėkime  $G$ , taip pat kadangi getNodeWithKey gali praeiti tarp 1 ir visų grandinės mazgų, grandinės ilgį pažymėkime  $n$ ):

	Kaina	Kartai
<pre> public String get(String key) {     int index = hash(key);      if (nodes[index] != null) {         Node existing = getNodeWithKey(nodes[index], key);          if (existing != null) {             return existing.value;         }     }      return null; } </pre>	C1	1
	C2	1
	G(n)	1
	C3	1
	C4	1
	C5	1

<pre>private Node getNodeWithKey(Node node, String key) {     if (node != null) {         do {              if (node.key.equals(key)) {                 return node;             }          } while ((node = node.next) != null);     }      return null; }</pre>	C6	1
	C7	n
	C8	1
	C9	n
	C10	1

Skačiuojame  $G(n)$  sudėtingumą:

$$G(n) = C6 + nC7 + C8 + nC9 + C10 = nC + C2 = O(n)$$

Skačiuojame  $T(n)$ :

$$T(n) = C1 + C2 + C3 + C4 + C5 + n = n + C = O(n)$$

Kadangi paieškos laikas priklauso nuo grandinės ilgio, kuris yra  $n$ , geriausiu atveju paieškos sudėtingumas yra  $\Theta(1)$ , o blogiausiu –  $O(n)$ .

Paskaičiuotas įvertinimas sutampa su teoriniu įvertinimu.

Paieškos operatyvinėje atmintyje rezultatai:

Duomenų kiekis	Paieškos laikas (ms)
50000	17
100000	21
200000	22
400000	48
800000	98
1600000	234
3200000	518

Paieškos išorinėje atmintyje rezultatai:

Duomenų kiekis	Paieškos laikas (ms)
300	27
600	48
1200	82
2400	135
4800	346
9600	671
19200	1649

## 6. Išvados

Merge sort algoritmo sudėtingumas yra  $O(n \cdot \log n)$  ir jis yra daug greitesnis dideliems duomenų kiekams nei insertion sort, kurio sudėtingumas yra  $O(n^2)$ . Merge sort iki 7 290 000 elementų rikiavimo laikas išaugo tik iki 2,5 sekundės, kai tuo tarpu insertion sort iki 64 000 elementų rikiavimo laikas išaugo iki 3,2 sekundės. Taip yra todėl, nes funkcija  $n^2$  auga daug greičiau nei funkcija  $n \cdot \log n$ .

Rikiavimo algoritmai buvo greitesni masyvo atveju, nes masyve galima tiesiogiai manipuluoti reikšmėmis, kai, tuo tarpu, sąraše reikia sukeitinėti mazgų rodykles, kas kainuoja daugiau laiko.

Visais algoritmų atvejais operatyvinė atmintis buvo greitesnė už išorinę atmintį, nes paimti duomenis iš operatyvinės atminties yra daug greičiau nei nuskaityti iš failo.

Paieškos vidutinis sudėtingumas maišos lentelėje yra  $\Theta(1)$ , todėl paieška buvo labai greita ir atlikti paiešką visiems 3 200 000 lentelės elementams tereikėjo tik 0,5 sekundės.