KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

ALGORITMŲ SUDARIMAS IR ANALIZĖ (P170B400)

LABORATORINIS DARBAS

Užduoties nr. 5

Atliko:

IFF-7/3 gr. studentas

Ernestas Jėcka

Priėmė:

Doc. PILKAUSKAS Vytautas

KAUNAS

2019

1. Turinys

2. Užduotis (nr. 5)

Realizuoti pateiktus algoritmus bei atlikti eksperimentinius tyrimus ir analizę.

Rikiavimo algoritmai:

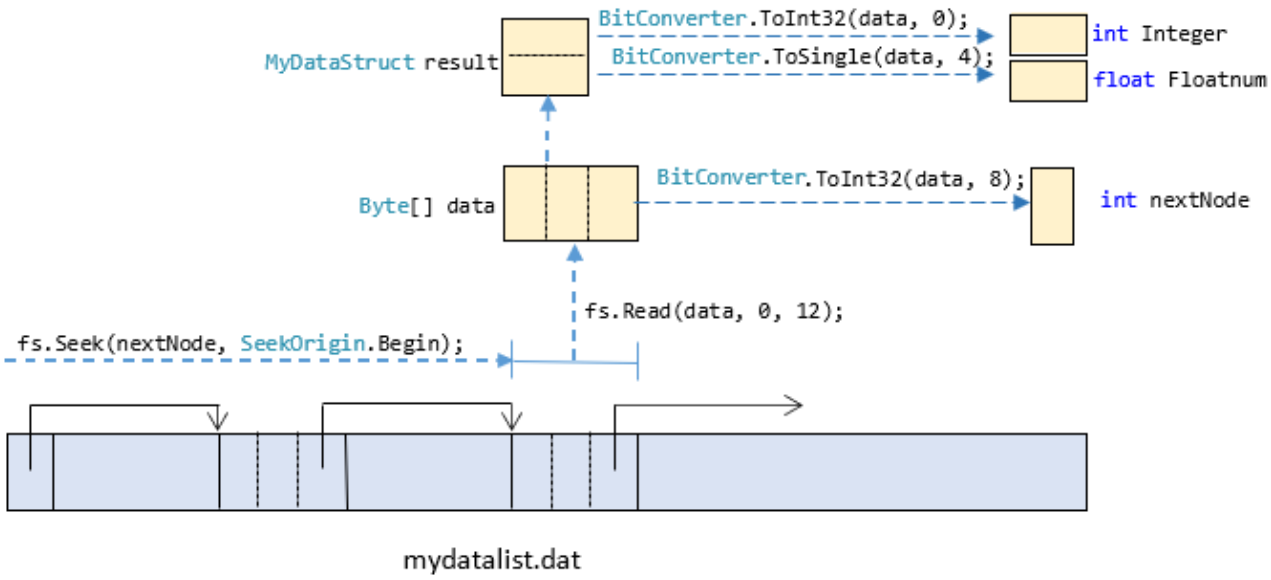
a) Rikiavimas „Merge sort“

b) Rikiavimas „Radix sort“

Paieškos algoritmas:

c) Maišos lentelės su tiesioginiu adresavimu, kai naudojama dvigubo hešavimo funkcija.

3. Susietojo sąrašo realizavimas išorinėje atmintyje




```
public override MyDataStruct Next()
{
    using (fs = new FileStream(fileDest, FileMode.Open, FileAccess.ReadWrite))
    {
        Byte[] data = new Byte[12];
        fs.Seek(nextNode, SeekOrigin.Begin);
        fs.Read(data, 0, 12);
        prevNode = currentNode;
        currentNode = nextNode;
        MyDataStruct result = new MyDataStruct(BitConverter.ToInt32(data, 0),
         BitConverter.ToSingle(data, 4));
        nextNode = BitConverter.ToInt32(data, 8);
        return result;
    }
}
```

4. Duomenų struktūrų realizacija

Vidinėje atmintyje

	kaina	Kartai

`class MyDataArray : DataArray`
`{`
`  MyDataStruct[] data;`
`  public MyDataArray(int n, int seed)`
`  {`
`  data = new MyDataStruct[n];`
`  length = n;`
`  Random rand = new Random(seed);`
`  for(int i = 0; i < length-1; i++)`
`  {`
`  data[i] = new MyDataStruct(rand);`
`  }`
`  data[length - 1] = new MyDataStruct(rand);`
`  data[length - 1].Int = data[length - 2].Int;`
`  }`		

`  public MyDataArray(int size) : base()`
`  {`
`  data = new MyDataStruct[size];`
	C_1	1

`  length = 0;` `  }`	C_2	1

`  public override void Add(MyDataStruct nauj)`
`  {`
`  data[length] = nauj;`
	C_3	1

`  length++;` `  }`	C_4	1

`  public override MyDataStruct this [int index]`
`  {`
`  get { return data[index]; }`
	C_5	1

`  set { data[index] = value;  }` `  }`	C_4	1

`  public override MyDataStruct First()`
`  {`
`  return data[0];`
`  }`
	C_5	1

`  public override void RemoveFirst()`
`  {`
`  data = data.Skip(1).ToArray();`
	C_6	1

`  length--;`
`  }`
`}`
	C_4	1

MyDataArray(int size) įvertinimas $T_{MDA}(int\ size) = C_1 + C_2$,

Add(MyDataStruct nauj) įvertinimas $T_{add}(obj\_size) = C_3 + C_4$,

Get įvertinimas $T_{get} = C_5$,

Set įvertinimas $T_{Set} = C_4$,

First() įvertinimas $T_{First} = C_5$,

RemoveFirst() $T_{RMFirst} = C_6 + C_4$,

5. „Merge sort“ rikiavimo analizės rezultatai

Teorinis įvertinimas

Šis rikiavimo metodas remiasi „Skaldyk ir valdyk“ paradigma. Metodas – rekursinis. Algoritmo viekimo tvarka:

- Visa duomenų imtis, turinti n elementų, padalinama į dvi imtis, kuriuose yra n/2 elementų.
- Šios dvi imtys rikiuojamos rekursyviai naudojant rikiavimą suliejimu.
- Sulieti dvi surikiuotas imtis tam, kad gauti surikiuotą atsakymą.

Imtis rekursyviai dalinama, ir tada suliejant šios imtys yra rikiuojamos. Teoriškai šio algoritmo sudėtingumas yra:
$O(n * log2(n))$
n – rikiuojamų elementų kiekis.

Suskaičiuotas algoritmo sudėtingumas

Lenght = n,

```
        private static DataArray MergeSort(DataArray unsorted)
        {
            if (unsorted.Length <= 1)
                return unsorted;
            DataArray left = new MyDataArray(unsorted.Length /
2);
            DataArray right = new MyDataArray(unsorted.Length /
2);
            int middle = unsorted.Length / 2;
            for (int i = 0; i < middle; i++)
            {
                left.Add(unsorted[i]);
            }
            for (int i = middle; i < unsorted.Length; i++)
            {
                right.Add(unsorted[i]);
            }
            left = MergeSort(left);
            right = MergeSort(right);
            return Merge(left, right);
        }
        private static DataArray Merge(DataArray left, DataArray
right)
        {
```

Kaina	Kiekiai
C_7	1
C_5	1
$T_{MDA}(n/2)$	1
$T_{MDA}(n/2)$	1
C_8	1
C_9	n/2
$T_{add}(obj)$	$\sum_{i=0}^{n/2} 1$
C_9	n/2
$T_{add}(obj)$	$\sum_{i=0}^{n/2} 1$
$T_{mergesort}(n/2)$	1
$T_{mergesort}(n/2)$	1
$T_{merge}(middle, middle)$	1

```
            DataArray result = new MyFileArray(left.Length +
right.Length);
                                                                    T_MDA(length)      n + 1
            while (left.Length > 0 || right.Length > 0)
            {                                                       C10                n
                if (left.Length > 0 && right.Length > 0)
                {                                                   C12                Σ_0^n t
                    if (left.First() <= right.First())
                    {
                                                                    Per vieną iteraciją turi bet
                        result.Add(left.First());                   kokiu atveju įvykti tik vienas
                                                                    kodo fragmentas, kurių
                        left.RemoveFirst();                         vertinimai nesiskiria.
                    }
                    Else                                            T_add(obj_nauj)    n
                    {
                        result.Add(right.First());                  T_RMFirst          n

                        right.RemoveFirst();
                    }
                }
                else if (left.Length > 0)                           C12                Σ_{i=0}^n t
                {
                    result.Add(left.First());
                    left.RemoveFirst();
                }                                                                      Σ_{i=0}^n t

                else if (right.Length > 0)                          C12
                {
                    result.Add(right.First());

                    right.RemoveFirst();
                }
            }
            return result;
        }
```

DataArray metodų sudėtingumas

MyDataArray(int size) įvertinimas $T_{MDA}(int\ size) = C_1 + C_2$,

Add(MyDataStruct nauj) įvertinimas $T_{add}(DataArray) = C_3 + C_4$,

Get įvertinimas $T_{get} = C_5$,

Set įvertinimas $T_{Set} = C_4$,

First() įvertinimas $T_{First} = C_5$,

RemoveFirst() įvertinimas $T_{RMFirst} = C_6 + C_4$,

t priklauso nuo to kuriame masyve yra didesnis pirmasis elementas elementas.

$$\mathrm{T}_{merge}(DataArray, DataArray)$$

$$= \ (n+1)T_{MDA}(int\ size) + \ nC_{10} + \ 3C_{12}\sum_{0}^{n} t + \mathrm{nT}_{add}(DataArray)$$

$$= (n+1)(C_1 + \ C_2) + nC_{10} + \ 3C_{12}\sum_{0}^{n} t + \ \ n(C_3 + \ C_4)$$

$$= \ n(C_1 + \ C_2 + \ C_{10} + \ C_3 + \ C_4) + C_1 + \ C_2 + \ 3C_{12}\sum_{0}^{n} t = \ nD_1 + D_2 + \ 3C_{12}\sum_{0}^{n} t$$

$$\mathrm{T}_{mergesort}(n) = \ \ 2T_{MDA}(n/2) + 2T_{mergesort}(n/2) \ + \ C_7 + C_5 + (n/2)C_9 + (n/2)C_9$$
$$+ \mathrm{T}_{merge}(DataArray, DataArray)$$
$$= \ 2T_{mergesort}(n/2) + n(C_1 + \ C_2 + \ C_{10} + \ C_3 + \ C_4 + C_9) + \ C_7 + C_5 + 3C_1 + \ 3C_2$$
$$+ \ 3C_{12}\sum_{0}^{n} t = 2T_{mergesort}(n/2) + nD_1 + \ 3C_{12}\sum_{0}^{n} t + D_3$$

Ignoruojame konstantas ir nereikšminius dėmesnis $\sum_0^n t\,, nes\ visada\ n \geq \ \sum_0^n t\ /$

$$\mathrm{T}_{mergesort}(n) = 2T_{mergesort}(n/2) + n$$

$$\mathrm{T}_{mergesort}(n) = 2^i T_{mergesort}(n/2^i) + in$$

Lengvai galima nuspėti, kad su vienu elementu algoritmo sudėtingumas lygus vienam.

$$\mathrm{T}_{mergesort}(1) = 1$$

Iš to galime lengvai išsireikšti medžio aukščio priklausomybę nuo elementų skaičiaus.

$$\frac{n}{2^i} = 1 => n = 2^i$$

$$\log_2 n = \ \log_2 2^i = \ \mathrm{i}\log_2 2 \quad = i$$

$$\mathrm{T}_{mergesort}(n) = nT_{mergesort}(1) + n\log_2 n$$

Įsistatę reikšmes gauname, kad sąlajos algoritmo sudėtingumas priklauso nuo elementų skaičiaus ir medžio aukščio.

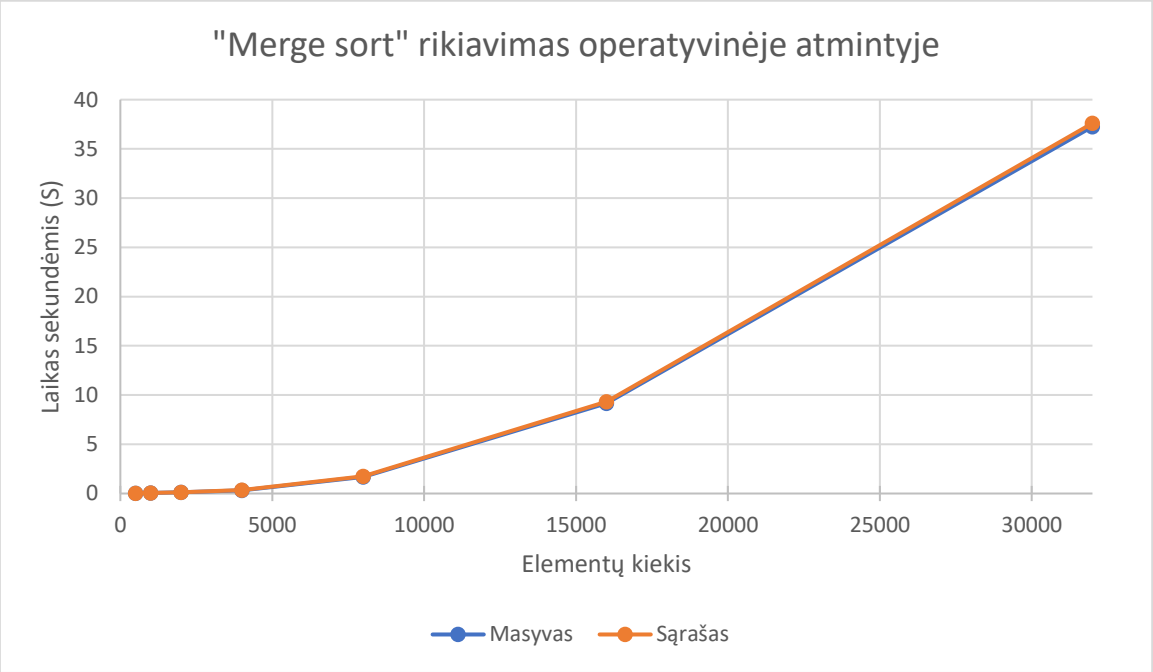
$$\mathrm{T}_{mergesort}(n) = n + n\log_2 n \quad = O(n\log_2 n\,)$$

Realizuojant išorinėje atmintyje įvertis nekinta, nes pasikeičia tik konstantinės laiko kainos.

Vykdymo laiko priklausomybės nuo elementų skaičiaus tyrimas

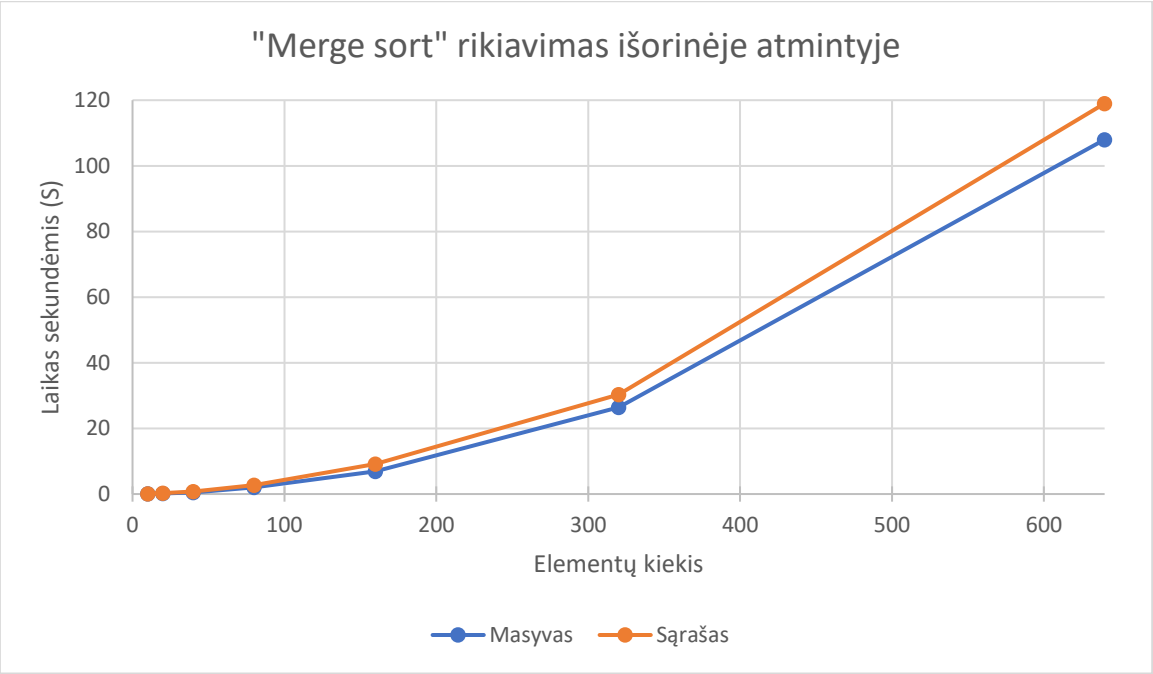
Operatyvinėje atmintyje

```
Merge rikiavimo testas operatyvineje atmintyje
N           Masyvas                 Sarasas
500         00:00:00.0296708        00:00:00.0354653
1000        00:00:00.0486234        00:00:00.0563202
2000        00:00:00.1140137        00:00:00.1227444
4000        00:00:00.3229912        00:00:00.3379304
8000        00:00:01.6750098        00:00:01.7414807
16000       00:00:09.1771346        00:00:09.3250178
32000       00:00:37.2613283        00:00:37.5990913
```



Išorinėje atmintyje

```
Merge rikiavimo testas isorineje atmintyje
N           Masyvas                 Sarasas
10          00:00:00.0779934        00:00:00.1338600
20          00:00:00.1733486        00:00:00.2969852
40          00:00:00.5208308        00:00:00.8091661
80          00:00:02.0311475        00:00:02.6932205
160         00:00:06.9750544        00:00:09.1149109
320         00:00:26.3999625        00:00:30.3087237
640         00:01:48.1971644        00:01:59.2470581
```



6. „Radix sort“ rikiavimo analizės rezultatai

Teorinis įvertinimas

Skaitmeninio rikiavimo algoritmuose duomenų reikšmės interpretuojamos kaip skaičiai M-tainėje (dažniausiai – dvejetainėje) skaičiavimo sistemoje. Algoritmas stabilus ir labai greitas, sudėtingumas – O(N·k) (*k* – skaičiaus ilgis)

Sudetingumo skaičiavimas

k = `data.LongestDigit,`

`n = data.Length;`

`bl = bucket.Length`

```
    public static DataIntArray RadixSort(DataIntArray data)
    {

        DataIntArray[] buckets = new DataIntArray[10];

            for (int i = 0; i < 10; i++)
            {
                buckets[i] = new MyIntArray(data.Length);
            }
        for (int i = 0; i < data.LongestDigit; i++)
        {
            for (int j = 0; j < data.Length; j++)
            {

                int digit = (int)((data[j] % Math.Pow(10, i +
1)) / Math.Pow(10, i));

                buckets[digit].Add(data[j]);
            }

            int index = 0;

            for (int k = 0; k < 10; k++)
```

Kaina	Kiekis
C_7	1
C_8	11
$T_{MDA}(int\ size)$	10
C_9	k + 1
C_9	(k + 1)*n
C_{10}	k *n
$T_{add}(size)$	k *n
C_{11}	k
C_9	k*11

```
            {
                DataIntArray bucket = buckets[k];

                for (int l = 0; l < bucket.Length; l++)
                {
                    data[index++] = bucket[l];
                }
                buckets[k].Clear();
            }
        }

        return data;
    }
```

C_{11}	k*10
C_9	k*10*(bl+1)
Tset	k*10*bl*2
C_{12}	k*10*bl
C_{11}	k*10*bl

MyIntArray(int size) įvertinimas $T_{MDA}(int\ size) = C_1 + C_2$,

Add(MyIntStruct nauj) įvertinimas $T_{add}(obj\_DataIntArray) = C_3 + C_4$,

Get įvertinimas $T_{get} = C_5$,

Set įvertinimas $T_{Set} = C_4$,

First() įvertinimas $T_{First} = C_5$,

RemoveFirst() $T_{RMFirst} = C_6 + C_4$,

T_{radix}(obj_DataIntArray) = $C_7 + 11C_8 + C_9(k+1) + 10T_{MDA}(obj_{size}) + C_9(n+1) + C_{10}kn + T_{add}(obj)kn + C_{11}k + 11kC_9 + 10kC_{11} + 10k(bl+1)C_9 + 20k*blC_4 + C_{12}k*10*bl + k*10*blC_2 = C_7 + 11C_8 + C_9k + C_9 + 10C_1 + 10C_2 + nC_9 n + nC_9 + C_{10}kn + C_3kn + C_4kn + C_{11}k + 11kC_9 + 10kC_{11} + 10k*bl + C_9 + 20k*blC_4 + C_{12}k*10*bl + k*10*blC_2 = kn(C_9 + C_{10} + C_3 + C_4) + 11k(C_9 + C_{11}) + k*bl(10 + C_9 + 20C_4 + 10C_{12} + 10C_2) + C_9k + C_9 + C_7 + 11C_8 + C_9 + 10C_1 + nC_9 + 10C_2 = knD_1 + kD_2 + k*blD_3 + kC_9 + nC_9 + D_4$,

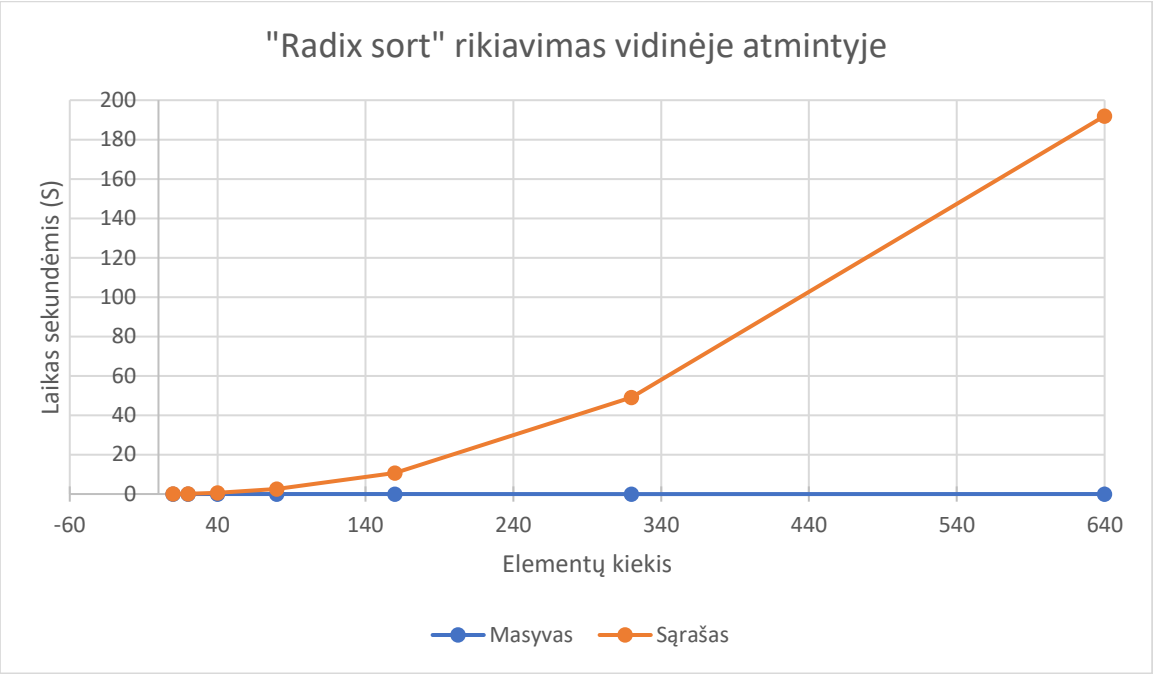
Kadangi visada n >= bl(buferio ilgis), tai svariausiu dėmeniu padaro knD_1, todėl gauname įvertį tiek iš viršaus tiek iš apačios, t.y. $T_{radix}(obj_{key}) = \theta(n*k)$ k – ilgiausio skaičiaus skaitmenų kiekis

Realizuojant išorinėje atmintyje įvertis nekinta, nes pasikeičia tik konstantinės laiko kainos.

Vykdymo laiko priklausomybės nuo elementų skaičiaus tyrimas

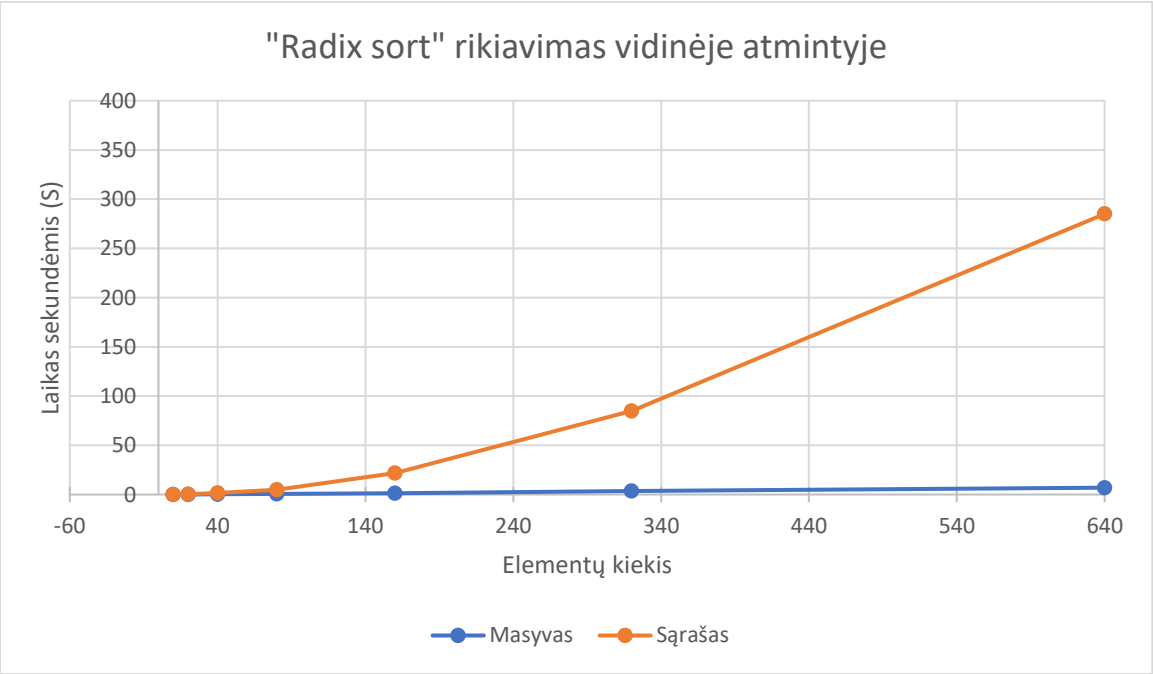
Vidinėje atmintyje

```
Radix rikiavimo testas vidineje atmintyje
N           Masyvas                 Sarasas
10          00:00:00.0002279        00:00:00.1019000
20          00:00:00.0002346        00:00:00.2006143
40          00:00:00.0002215        00:00:00.5930582
80          00:00:00.0001363        00:00:02.5596847
160         00:00:00.0002601        00:00:10.6717133
320         00:00:00.0010850        00:00:49.0707775
640         00:00:00.0025319        00:03:12.6143162
```



Išorinėje atmintyje

Radix rikiavimo testas isorineje atmintyje		
N	Masyvas	Sarasas
10	00:00:00.1185475	00:00:00.0993455
20	00:00:00.1739483	00:00:00.2985394
40	00:00:00.4652093	00:00:01.6218261
80	00:00:00.8025320	00:00:04.8659272
160	00:00:01.3669511	00:00:21.7647085
320	00:00:03.6920680	00:01:25.6410524
640	00:00:06.9763070	00:04:45.3358115



7. Maišos lentelės paieškos algoritmo analizės rezultatai

Teorinis įvertinimas

Dažniausiai dvigubam dėstymui reikia mažiau palyginimų nei tiesiniam dėstymui.

Atviro adresavimo pagrindinis privalumas yra mažesnis reikalingas atminties kiekis.

Skaičiuojant efektyvumą, tariama, kad blogiausiu atveju yra užpildyta visa lentelė ir laiko atžvilgiu efektyvumas yra O (N *M), kur M lentelės dydis, o N norimų įterpti įrašų skaičius

Apskaičiuotas įvertinimas

	Kaina	Kiekis
`public V Get(K key)` `{`		
`int positionIndex = FindPosition(key);`	$T_{FP}(key)$	1
`if (positionIndex >= 0 && table[positionIndex] != null)`	C_1	1
`{`		
`return table[positionIndex].value;`	C_6	1
`}`		
`return null;`	C_2	1
`}`		
`private int FindPosition(K key)`		
`{`		
`int index = HashCode(key);`	$T_H(key)$	1
`int indexO = index;`	C_7	1
`int i = 0;`	C_7	1
`for (int j = 0; j < capacity; j++)`	C_8	1
`{`	C_9	capacity + 1
`if (table[index] == null ||` `table[index].key.Equals(key))`	C_{10}	capacity
`{`		
`return index;`	C_{11}	$\sum_{j=0}^{capasity} t$
`}`		
`index = (indexO + j * hashCode2(key)) % capacity;`	$C_{12}+ T_{H2}(key)$	$\sum_{j=0}^{capasity} \bar{t}$
`}`		
`return -1;`		
`}`	C_{11}	1
`private int hashCode2(K key)`		
`{`		
`return 7 - (Math.Abs(key.GetHashCode()) % 7);`		
`}`	C_5	1
`int HashCode(K key)`		
`{`		
`int h = key.GetHashCode();`	C_3	1
`return Math.Abs(h % capacity);`		
`}`	C_4	1

Capacity = n,

HashCode(K key) įvertinimas

$T_H(obj\_K)=C_3+C_4$,

HashCode2(K key) įvertinimas

$T_{H2}(obj\_K)=C_5$,

čia t = 1, jei elementas su priskirtu raktu neegzistuoja arba jei su tokiu raktu elementas jau egzistuoja, o kitu atveju t = 0.

FindPosition(K key) įvertinimas

$T_{FP}(obj_K) = T_H(obj_K) + 2C_7 + C_8 + C_9(n+1) + C_{10} * n + C_{11}\sum_{j=0}^{n} t + \left(C_{12} + T_{H2}(key)\right)\sum_{j=0}^{n} \bar{t} = C_3 + C_4 + 2C_7 + C_8 + n(C_9 + C_{10}) + C_9 + C_{11}\sum_{j=0}^{n} t + (C_{12} + C_5)\sum_{j=0}^{n} \bar{t}$,

Get(K key) įvertinimas

$T_{Get}(obj\_key) = T_{FP}(obj_{key}) + C_1 + C_6 + C_2 = C_3 + C_4 + 2C_7 + C_8 + n(C_9 + C_{10}) + C_9 + C_{11}\sum_{j=0}^{n} t + (C_{12} + C_5)\sum_{j=0}^{n} \bar{t} + C_1 + C_6 + C_2$,

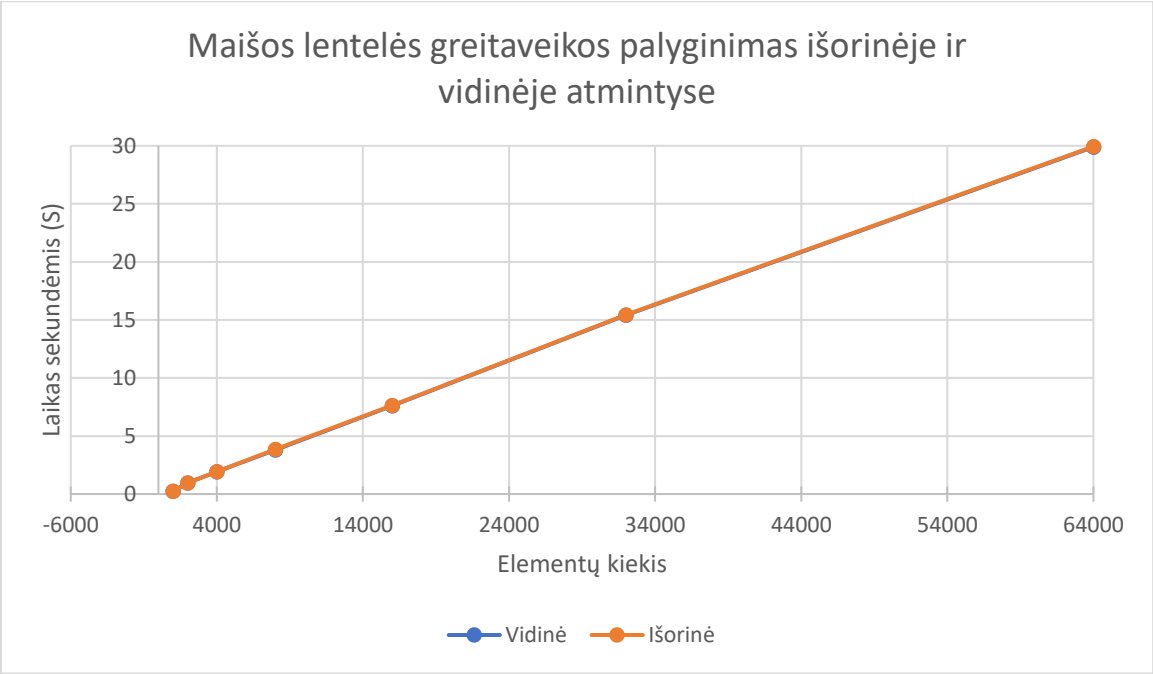
Kadangi $0 \le \sum_{j=0}^{n} t \le n$ *ir* $0 \le \sum_{j=0}^{n} \bar{t} \le n$, dižiausią reikšmingumą įgauną dėmuo $n(C_9 + C_{10})$, todėl

$T_{Get}(obj\_key) = \theta(n)$

Realizuojant išorinėje atmintyje įvertis nekinta, nes pasikeičia tik konstantinės laiko kainos.

Maišos lentelės greitaveikos palyginimas išorinėje ir vidinėje atmintyse

```
Maisos lenteles vidutinio paieskos laiko palyginimas
N          Operatyvine          Isorne
1000       240 ms               241 ms
2000       950 ms               951 ms
4000       1938 ms              1939 ms
8000       3861 ms              3862 ms
16000      7627 ms              7628 ms
32000      15434 ms             15435 ms
64000      29938 ms             29940 ms
```



8. Išvados

Visus algoritmus realizuoti pasisekė sėkmingai, sąlajos rikiavimas operatyvinėje atmintyje gana efektyvus tiek saugant duomenis sąraše, tiek masyve, tačiau išorinėje atmintyje realizuotas šis algoritmas nebuvo toks efektyvus ir atsirado didesnis greitaveikų skirtumas tarp realizacijos masyve ir sąraše, galima daryti prielaidą, kad didelę žąla greitaveikai turėjo, kad dalinant duomenų imtis buvo kuriami vis nauji duomenų failai. „Radix“ rikiavimo algoritmas gerokai lėčiau veikia realizuotas sąraše, negu masyve, išorinėje atmintyje saugant duomenis matomas šioks tok sulėtėjimas, bet greitaveikos kitimo tendencijos lieka panašios. Maišos lentelės paieškos algoritmas veikia vienodu spartumu tiek, kai duomenys saugomi išorinėje tiek vidinėje atmintyje. Visais atvejais buvo aiškiai matyti iš laiko priklausomybės grafiko, kad algoritmai realizuoti optimaliai ir atitinka teorinius įverčius.

9. Literatūros sąrašas

1. „Algoritmų sudarymas ir analizė“ modulis „Moodle“ aplinkoje
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Riverst, Clifford Stein „Introduction to Algorithms: Third Edition“