Kauno technologijos universitetas

Informatikos fakultetas, programų inžinerijos katedra

P170B400 Algoritmų sudarymas ir analizė

Bucket sort algoritmas ir jo analizė

Arnas Švenčionis

Projekto autorius

IFF-8/11

Akademinė grupė

Table of Contents

Algoritmo įvertinimas literatūroje

Bucket sort algoritmo teoriniai įvertinimai, kai naudota duomenų struktūra yra masyvas. Šaltinis – Vikipedija.

Worst-case performance	$O(n^2)$
Average performance	$O(n + \frac{n^2}{k} + k)$, where k is the number of buckets. $O(n)$, when $k \approx n$.

Algoritmo sudėtingumas, remiantis programos išeities tekstu.

Rikiavimas operatyvinėj atminty naudojant masyvą

```
public int Length { get { return length; } }                                  c2   1
```

Lenght metodo sudėtingumo įvertinimas - T_L(obj_FLinesArray) = c2

```
public override Objektas this[int index]
        {
            get { return data[index]; }                                       c3   1
        }
```

Change metodo sudėtingumo įvertinimas - T_L(obj_FLinesArray) = c3

```
public override void Change(int index, Objektas naujas)
        {
            data[index] = naujas;                                             c4   1
        }
```

Indekso metodo sudėtingumo įvertinimas - = c3

```
public static void BubbleSort(List<Objektas> items)                        kaina   kiekis
        {
            Objektas prevdata, currentdata;                                  c1    1
            for (int i = items.Count - 1; i >= 0; i--)                       c2    n
            {
                currentdata = items[0];                                      c3    n-1
                for (int j = 1; j <= i; j++)                                 c4    n*(n-1)
                {
                    prevdata = currentdata;                                  c5    (n-1)*(n-1)
                    currentdata = items[j];                                c5+c3   (n-1)*(n-1)
                    if (prevdata > currentdata)                              c5    (n-1)*(n-1)
                    {
                        items[j - 1] = currentdata;                          c5    (n-1)*(n-1)
                        items[j] = prevdata;                                 c5    (n-1)*(n-1)
                        currentdata = prevdata;                              c5    (n-1)*(n-1)
                    }
                }
            }
        }
```

BubbleSort metodo sudėtingumo įvertinimas = c1 + n*c2 + c3*(n-1) + c4*(n*(n-1)

+ 6*c5*(n-1)*(n-1) = c1 + c2*n + (n-1)*c3 + (n^2-n)*c4 + 6*c5*(n^2-2n+1) =

= c1 + n*(c2 + c3 + c4 6*c5) + n^2*(c4 + 6c5) = O(n^2)

```
private static void BucketSortArray(DataArray x)
{
    int n = x.Length;
    int numOfBuckets = 10;

    List<Objektas>[] buckets = new List<Objektas>[numOfBuckets];
    for (int i = 0; i < numOfBuckets; i++)
    {
        buckets[i] = new List<Objektas>();
    }

    for (int i = 0; i < n; i++)
    {
        int bucket = (int)(x[i].flo * numOfBuckets);
        buckets[bucket].Add(x[i]);
    }


    int a = 0;
    for (int i = 0; i < numOfBuckets; i++)
    {
        BubbleSort(buckets[i]);
        for (int j = 0; j < buckets[i].Count; j++)
        {
            x.Change(a, buckets[i][j]);
            a++;
        }
    }
}
```

kaina	kiekis
C1+c2	1
C1	1
C1	1
C2	10+1
C3	10
C4	n+1
C5	n
C5+c2	n
C1	1
C4	n+1
C5	n
C7	n^2
C5	n
C5+c3	n

$T(n)=3*c1 + (c1+c2) + c2*11 + c3*10 + 2*c4*n+1 + 3*c5*n + (c5+c2)*n + (c5+c3)*n + c7*n^2 = n(3*c5 + (c5+c2)) + c + c7*n^2 = O(n^2)$

Rikiavimas operatyvinėj atminty naudojant sąrašą

```
public int Length { get { return length; } }                                  c1   1
```

Lenght metodo sudėtingumo įvertinimas = c1

```
public override Objektas Head()                                             kiekis  kaina
        {
            currentNode = headNode;                                           c1    1
            prevNode = null;                                                  c1    1
            return currentNode.data;                                          c1    1
        }
```

Head metodo sudėtingumo įvertinimas = 3*c1

```
        public override Objektas Next()
        {
            prevNode = currentNode;                                           c1    1
            currentNode = currentNode.nextNode;                               c1    1
            if (currentNode == null) return null;                             c1    1
            return currentNode.data;                                          c1    1
        }
```

Next metodo sudėtingumo įvertinimas = 4*c1

```
public override void addAll(List<Objektas> items)                           kiekis  kaina
        {
            foreach (Objektas item in items)                                  c3    n
            {
                if (headNode == null)                                         c2    n-1
                {
                    headNode = new MyLinkedListNode(item);                    c2    n-1
                    currentNode = headNode;                                   c2    n-1
                    continue;                                                 c2    n-1
                }
                prevNode = currentNode;                                       c2    n-1
                currentNode.nextNode = new MyLinkedListNode(item);            c2    n-1
                currentNode = currentNode.nextNode;                           c2    n-1
            }
            currentNode.nextNode = null;                                      c1     1
        }
```

addAll metodo sudėtingumo įvertinimas = n*c3 + c1 + 6*c2*(n-1) = O(n);

```
public override void clear()
        {
            headNode = null;                                                  c1    1
            prevNode = null;                                                  c1    1
            currentNode = null;                                               c1    1
        }
```

clear metodo sudėtingumo įvertinimas = 3*c1

```
public static void BubbleSort(List<Objektas> items)                         kaina   kiekis
        {
            Objektas prevdata, currentdata;                                   c1     1
            for (int i = items.Count - 1; i >= 0; i--)                        c2     n
            {
                currentdata = items[0];                                       c3     n-1
                for (int j = 1; j <= i; j++)                                  c4     n*(n-1)
                {
```

```
                    prevdata = currentdata;                                  c5    (n-1)*(n-1)
                    currentdata = items[j];                                c5+c3   (n-1)*(n-1)
                    if (prevdata > currentdata)                              c5    (n-1)*(n-1)
                    {
                        items[j - 1] = currentdata;                          c5    (n-1)*(n-1)
                        items[j] = prevdata;                                 c5    (n-1)*(n-1)
                        currentdata = prevdata;                              c5    (n-1)*(n-1)
                    }
                }
            }
        }
```

BubbleSort metodo sudėtingumo įvertinimas bs= c1 + n*c2 + c3*(n-1) + c4*(n*(n-1)

+ 6*c5*(n-1)*(n-1) = c1 + c2*n + (n-1)*c3 + (n^2-n)*c4 + 6*c5*(n^2-2n+1) =

= c1 + n*(c2 + c3 + c4 6*c5) + n^2*(c4 + 6c5) = O(n^2)

```
private static void BucketSortList(DataList x)
        {
            int numOfBuckets = 10;                                           c1    1

            List<Objektas>[] buckets = new List<Objektas>[numOfBuckets];     c1    1
            for (int i = 0; i < numOfBuckets; i++)                           c2    10+1
            {
                buckets[i] = new List<Objektas>();                           c3    10
            }

            Objektas temp = x.Head();                                      c4+3*c1  1
            buckets[(int)(temp.flo * numOfBuckets)].Add(temp);               c1    1
            for (int i = 1; i < x.Length; i++)                               c4    n+1
            {
                temp = x.Next();                                           c5+4*c1  n
                int bucket = (int)(temp.flo * 10);                           c5    n
                buckets[bucket].Add(temp);                                   c5    n
            }
            int a = 0;                                                     c1+3*c1  1
            x.clear();
            for (int i = 0; i < numOfBuckets; i++)                           c2    10+1
            {
                BubbleSort(buckets[i]);                                    c3+n^2   10
                x.addAll(buckets[i]);                                      c3+n     10
            }
        }
```

T(n)= 3*c1 + 2*c2*11+ c3*10 + (c4+3+c1)1 + c4(n+1) + 2*c5 + (c5+4*c1)n + (c1+3*c1) 2*(c3+n^2)*10 = c + n(c4 + (c5+4*c1) + c3) + n^2(20*c3) = O(n^2)

Rikiavimas išorinėje atminty naudojant masyvą

```
public override Objektas this[int index]
        {
            get
            {
                Byte[] data = new Byte[8];                                          c1    1
                fs.Seek(8 * index, SeekOrigin.Begin);                               c1    1
                fs.Read(data, 0, 8);                                                c1    1
                string s = Encoding.ASCII.GetString(data.Take(4).ToArray());        c1    1
                float dataFloat = BitConverter.ToSingle(data, 4);                   c1    1
                return new Objektas(s, dataFloat);                                  c1    1
            }
        }
```

Indekso sudėtingumo įvertinimas - T_I(obj_FLinesArray, j) = 6c1

```
public int Length { get { return length; } }                                        c2    1
```

Lenght metodo sudėtingumo įvertinimas - T_L(obj_FLinesArray) = c2

```
public override void SetValue(int i, Objektas v)
        {
            Byte[] data = new Byte[8];                                              c6    1
            Encoding.ASCII.GetBytes(v.str).CopyTo(data, 0);                         c6    1
            BitConverter.GetBytes(v.flo).CopyTo(data, 4);                           c6    1
            fs.Seek(8 * i, SeekOrigin.Begin);                                       c6    1
            fs.Write(data, 0, 8);                                                   c6    1
        }
```

SetValue metodo sudėtingumo įvertinimas - = 5c6

```
public static int CompareV1(Objektas a, Objektas b)
            {
                if (a.flo == b.flo)                                                 c7    1
                    return a.str.CompareTo(b.str);                                  c7    1
                else
                    return a.flo.CompareTo(b.flo);
            }
```

CompareV1 metodo sudėtingumo įvertinimas - = 2c7

```
public static void InsertionSort(DataArray myList)
            {
                int n = myList.Length;                                                          c3     1
                for (int i = 1; i < n; ++i)                                                     c4     n+1
                {
                    Objektas key = myArray[i];                                                  c5+6c1 n
                    int j = i - 1;                                                              c5     n

                    while (j >= 0 && CompareV1(myArray[i], key) > 0)          c8+TI(arr,j)+Tcmp(key) n(n+1)/2-1
                    {
                        myList.SetValue(j + 1, myArray[i]);           c9+TI(arr, j)+Trw(arr,j+1)   n(n-1)/2
                        j = j - 1;                                                              c9     n(n-1)/2
                    }
                    myList.SetValue(j + 1, key);                                                c7     n-1
                }
            }
```

Laikome, kad T_{rw}(obj_FLinesArr, j) = 6c_4, T_I(obj_FLinesArray, j) = 6C_3, T_{cmp}(obj_Line)= 7c1, T_L=c2

$T_{sortArray}() = c_5 + T_L(arr) + nc_6 + (T_I(arr,i))(n-1) + 3c_7(n-1) + (c_8 + T_I(arr,j) + T_{cmp}(key))(n(n+1)/2 - 1) + (T_I(arr, j) + T_{rw}(arr,j+1))(n(n-1)/2) + 2c_9(n(n-1)/2) = c_5 + T_L(arr) + nc_6 + n\,T_I(arr,i) - T_I(arr,i) + 3c_7n - 3c_7 + c_8n^2/2 + c_8n/2 - c_8 + T_I(arr, j)n^2/2 + T_I(arr, j)n/2 -$

$T_l(arr, j)+ T_{cmp}(key)n^2/2 + T_{cmp}(key)n/2 – T_{cmp}(key) + T_l(arr, j)n^2/2 - T_l(arr, j)n/2 + T_{rw}(arr, j+1)n^2/2 - T_{rw}(arr, j+1)n/2 + c_9n^2 – c_9n =$
$\mathbf{n^2}(c_8/2 + 6c_3 + 7c1/2 +6c_4/2 + c9) + n\ (c_6 + 6c_3 + 7c1/2 + 3c_7 + c8/_2 – 6c_4/2 – c_9) + c_5 + c_2 – 26c_3 - 7c1 – 3c_7 – c_8 = O(\mathbf{n^2})$

	svoris	kaina
`private static MyFileArray BucketSortArray(DataArray x)`		
`{`		
`DirectoryInfo di = new DirectoryInfo(@"..\..\data\");`	c1	1
`foreach (FileInfo file in di.GetFiles())`	c8	10+1
`if (file.Name.Contains("ABucket"))`	c3	10
`file.Delete();`	c3	10
`int[] lengths = new int[10];`	c1	1
`for (int i = 0; i < x.Length; i++)`	c6+6c1	n+1
`{`		
`Objektas key = x[i];`	c7+c2	n
`int bucket = (int)(key.flo * 10);`	c7	n
`string fileName = "ABucket" + bucket + ".dat";`	c7	n
`string path = @"..\..\data\" + fileName;`	c7	n
`if (!File.Exists(@"..\..\data\" + fileName))`	c7	n
`using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.Create)))`	c7	n
`{`		
`Byte[] str = Encoding.ASCII.GetBytes(key.str);`	c7	n
`writer.Write(str);`	c7	n
`writer.Write(key.flo);`	c7	n
`lengths[bucket] = 1;`	c7	n
`}`		
`else`		
`using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.Append)))`		
`{`		
`Byte[] str = Encoding.ASCII.GetBytes(key.str);`		
`writer.Write(str);`		
`writer.Write(key.flo);`		
`lengths[bucket] += 1;`		
`}`		
`}`		
`MyFileArray ats = new MyFileArray(@"ats.dat", x.Length);`	c5+c2	1
`using (BinaryWriter writer = new BinaryWriter(File.Open(@"ats.dat", FileMode.Create)))`	c1	1
`{`		
`foreach (FileInfo file in di.GetFiles())`	c6	n+1
`{`		
`int length = lengths[int.Parse(file.Name.Substring(7, 1))];`	c7	n
`MyFileArray buck = new MyFileArray(@"..\..\data\" + file.Name, length);`	c7	n
`using (buck.fs = new FileStream(@"..\..\data\" + file.Name, FileMode.Open, FileAccess.ReadWrite))`	c7	n
`{`		
`InsertionSort(buck);`	c9	n^2
`for (int j = 0; j < length; j++)`	c8	n+1
`{`		
`Byte[] str = Encoding.ASCII.GetBytes(buck[j].str);`	c7	n
`writer.Write(str);`	c7	n
`writer.Write(buck[j].flo);`	c7	n
`}`		
`}`		
`}`		
`}`		
`return ats;`	c1	1
`}`		

$T(N)=4*c1 + 2*10*c3 + (n+1)*(c6+6c1) + n*(c7+c2) + 15*c7*n + c5+c2 + (n+1)*c* + c9*n^2 = n((c7+c2) + 15*c7) + c9*n^2 + c = O(n^2)$

Rikiavimas išorinėje atminty naudojant sąrašą

```
public override Objektas Head()                                                    svoris  kaina
        {
            Byte[] data = new Byte[12];                                              c1    1
            fs.Seek(0, SeekOrigin.Begin);                                            c1    1
            fs.Read(data, 0, 4);                                                     c1    1
            currentNode = BitConverter.ToInt32(data, 0);                             c1    1
            prevNode = -1;                                                           c1    1
            fs.Seek(currentNode, SeekOrigin.Begin);                                  c1    1
            fs.Read(data, 0, 12);                                                    c1    1
            string str = Encoding.ASCII.GetString(data.Take(4).ToArray());           c1    1
            float flo = BitConverter.ToSingle(data, 4);                              c1    1
            nextNode = BitConverter.ToInt32(data, 8);                                c1    1
            return new Objektas(str, flo);                                           c1    1
        }
```

Head metodo sudėtingumo įvertinimas - = 11c1

```
        public override Objektas Next()                                            svoris  kaina
        {
            Byte[] data = new Byte[12];                                              c1    1
            fs.Seek(nextNode, SeekOrigin.Begin); fs.Read(data, 0, 12);               c1    1
            prevNode = currentNode;                                                  c1    1
            currentNode = nextNode;                                                  c1    1
            string str = Encoding.ASCII.GetString(data.Take(4).ToArray());           c1    1
            float flo = BitConverter.ToSingle(data, 4);                              c1    1
            nextNode = BitConverter.ToInt32(data, 8);                                c1    1
            return new Objektas(str, flo);                                           c1    1
        }
```

Next metodo sudėtingumo įvertinimas - = 8c1

```
public override Objektas ElementAt(int n)                                          svoris  kaina
        {
            Objektas temp = Head();                                                  c1    1
            for (int i = 0; i < Length; i++)                                         c6    n+1
            {
                if (i == n)                                                          c7    n
                    return temp;
                temp = Next();
            }
            return temp;
        }
```

ElementAt metodo sudėtingumo įvertinimas - = n

```
public static void InsertionSort(DataList myList)                                  svoris  kaina
            {
                int n = myList.Length;                                               c1    1
                for (int i = 1; i < n; ++i)                                          c6    n+1
                {
                    Objektas key = myList.ElementAt(i);                           c7+Tel(k)  n
                    int j = i - 1;                                                   c7    n

                    while (j >= 0 && CompareV1(myList.ElementAt(j), key) > 0) c12+TGE(list,j)+Tcmp(key) n(n+1)/2-1
                    {
                    myList.SetValue(j + 1, myList.ElementAt(j));   c13 + TRW(list, j+1) + TGE(list, j) n(n-1)/2
                        j = j - 1;                                                   c13   n(n-1)/2
                    }
                    myList.SetValue(j + 1, key);                          c11 + TRW(list, j+1)    n-1
                }
            }
```

Laikome, kad $T_{GE}(obj\_FLineList, j) = n$, $rw_E(obj\_FLinesList, j) = n$, $T_{RW}(obj\_FLinesList, j) = O(n)$

$T_{sortList}(obj\_FLinesList) = c_9 + nc_{10} + (3c_{11} + T_{GE}(list, i))(n-1) + (c_{12}+T_{GE}(list,j)+T_{cmp}(key))* (n(n+1)/2-1) + (2c_{13} + T_{RW}(list, j+1) + T_{GE}(list, j))(n(n-1)/2) + T_{RW}(list, j+1)(n-1) = c_9 + nc_{10} + 3nc_{11} - 3c_{11} + n*n - n + n^2C_{12}/2 + nc_{12}/2 - c_{12} + n^2*n/2 + n*n/2 - n + n^27c_1/2 + n7c_1/2 - 7c_1 + n^22c_{13}/2 - 2c_{13}n/2 + n*n^2/2 - n*n/2 + n*n^2/2 - n*n/2 + n*n - n = \mathbf{n^3}3/2 + n^2(1+c_{12}/2 + 7c1/2 + c_{13}) + n\ (c_{10} + 3c_{11} + c_{12}/2 - 2n + 7c_1/2 - c_{13}) + c_9 - 3c_{11} - n - c_{12} - 7c_1 = O(\mathbf{n^3})$

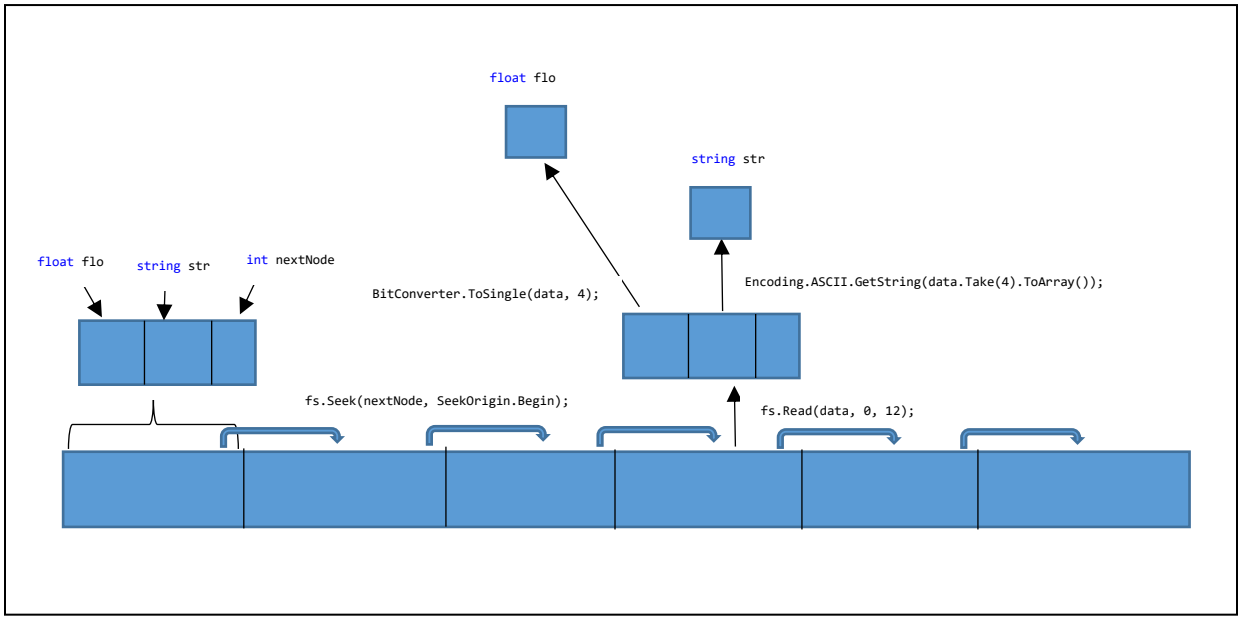
	svoris	kaina
`private static MyFileList BucketSortList(DataList x)`		
`{`		
`DirectoryInfo di = new DirectoryInfo(@"..\..\data\");`	c1	1
`foreach (FileInfo file in di.GetFiles())`	c8	10+1
`if (file.Name.Contains("ABucket"))`	c3	10
`file.Delete();`	c3	10
`int[] lengths = new int[10];`	c1	1
`for (int i = 0; i < 10; i++)`	c8	10+1
`lengths[i] = 0;`	c3	10
`for (int i = 0; i < x.Length; i++)`	c6+c2	n+1
`{`		
`Objektas temp;`	c7	n
`int bucket;`	c7	n
`if (i == 0)`	c7	n
`{`		
`temp = x.Head();`	c7+11c1	n
`bucket = (int)(temp.flo * 10);`	c7	n
`}`		
`else`		
`{`		
`temp = x.Next();`		
`bucket = (int)(temp.flo * 10);`		
`}`		
`string fileName = "LBucket" + bucket + ".dat";`	c7	n
`string path = @"..\..\data2\" + fileName;`	c7	n
`if (!File.Exists(path))`	c7	n
`using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.Create)))`	c7	n
`{`		
`writer.Write(4);`	c7	n
`Byte[] str = Encoding.ASCII.GetBytes(temp.str);`	c7	n
`writer.Write(str);`	c7	n
`writer.Write(temp.flo);`	c7	n
`writer.Write((lengths[bucket] + 1) * 12 + 4);`	c7	n
`lengths[bucket] += 1;`	c7	n
`}`		
`else`		
`using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.Append)))`		
`{`		
`Byte[] str = Encoding.ASCII.GetBytes(temp.str);`		
`writer.Write(str);`		
`writer.Write(temp.flo);`		
`writer.Write((lengths[bucket] + 1) * 12 + 4);`		
`lengths[bucket] += 1;`		
`}`		
`}`		
`MyFileList ats = new MyFileList(@"Lats.dat", x.Length);`	c1+c2	1
`using (BinaryWriter writer = new BinaryWriter(File.Open(@"Lats.dat", FileMode.Create)))`	c1	1
`{`		
`writer.Write(4);`	c1	1
`int ind = 0;`	c1	1
`foreach (FileInfo file in di.GetFiles())`	c8	10+1
`{`		
`int length = lengths[int.Parse(file.Name.Substring(7, 1))];`	c3	10
`MyFileList buck = new MyFileList(@"..\..\data2\" + file.Name, length);`	c3	10
`using (buck.fs = new FileStream(@"..\..\data2\" + file.Name, FileMode.Open, FileAccess.ReadWrite))`	c3	10
`{`		
`//buck.Print(buck.Length);`		
`InsertionSort(buck);`	$c3+n^3$	10
`for (int j = 0; j < length; j++)`	c10	n/10+1
`{`		

```
                        Objektas t;                                                    c11   n/10
                        if (j == 0)                                                    c11   n/10
                            t = buck.Head();                                           c11   n/10
                        else
                            t = buck.Next();
                        Byte[] str = Encoding.ASCII.GetBytes(t.str);                   c11   n/10
                        writer.Write(str);                                             c11   n/10
                        writer.Write(t.flo);                                           c11   n/10
                        writer.Write((ind + 1) * 12 + 4);                              c11   n/10
                        ind++;                                                         c11   n/10
                    }
                }
            }
        }
        return ats;                                                                    c1    1
    }
```

Compare, setValue ir Length metodai tokie patys kaip ir masyvo rikiavime

T(n)= 7*c1 + 6*c3*10 + 3*c8*11 n+1*(c6+c2) + 14*c7 + n*(c7+11c1) + (c1+c2) + 10*(c3_n^3) + c10*(n/10 + 1) + 8*c11*(n/10) = O(n^3)

Panaudotos duomenų struktūros realizacijos išorinėje atmintyje struktūrinę diagrama




```
public override Objektas Next()
 {
     Byte[] data = new Byte[12];
     fs.Seek(nextNode, SeekOrigin.Begin);
     fs.Read(data, 0, 12);
     prevNode = currentNode;
     currentNode = nextNode;
     string str = Encoding.ASCII.GetString(data.Take(4).ToArray());
     float flo = BitConverter.ToSingle(data, 4);
     nextNode = BitConverter.ToInt32(data, 8);
     return new Objektas(str, flo);
 }
```

Atlikti eksperimentai, greitaveika

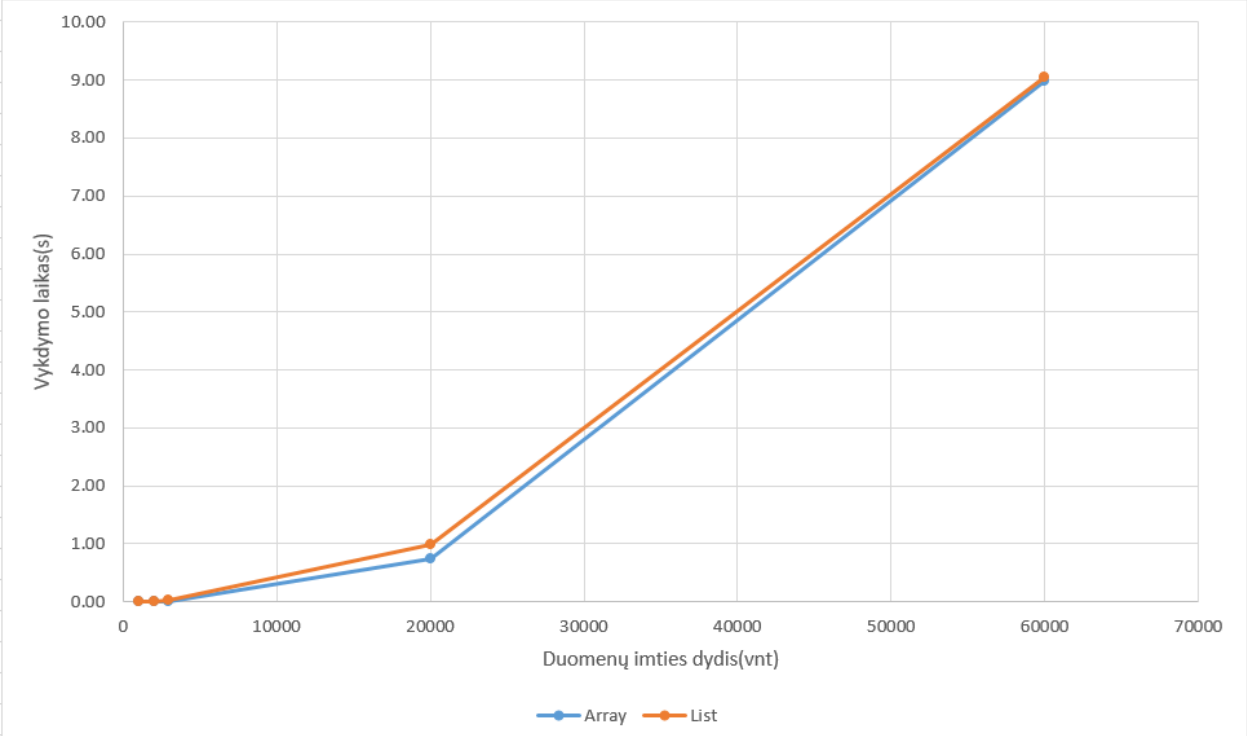
Operatyviosios atminties algoritmo analizė:

Apskaičiuoti algoritmų sudėtingumai:

Masyvas : n^2

Sąrašas : n^2

```
Array
1000 - 00:00:00.0026678
2000 - 00:00:00.0072528
3000 - 00:00:00.0164186
20000 - 00:00:00.7373715
60000 - 00:00:08.9720422
LinkedList
1000 - 00:00:00.0034246
2000 - 00:00:00.0095418
3000 - 00:00:00.0214299
20000 - 00:00:00.9801791
60000 - 00:00:09.0454277
Press any key to continue . . .
```



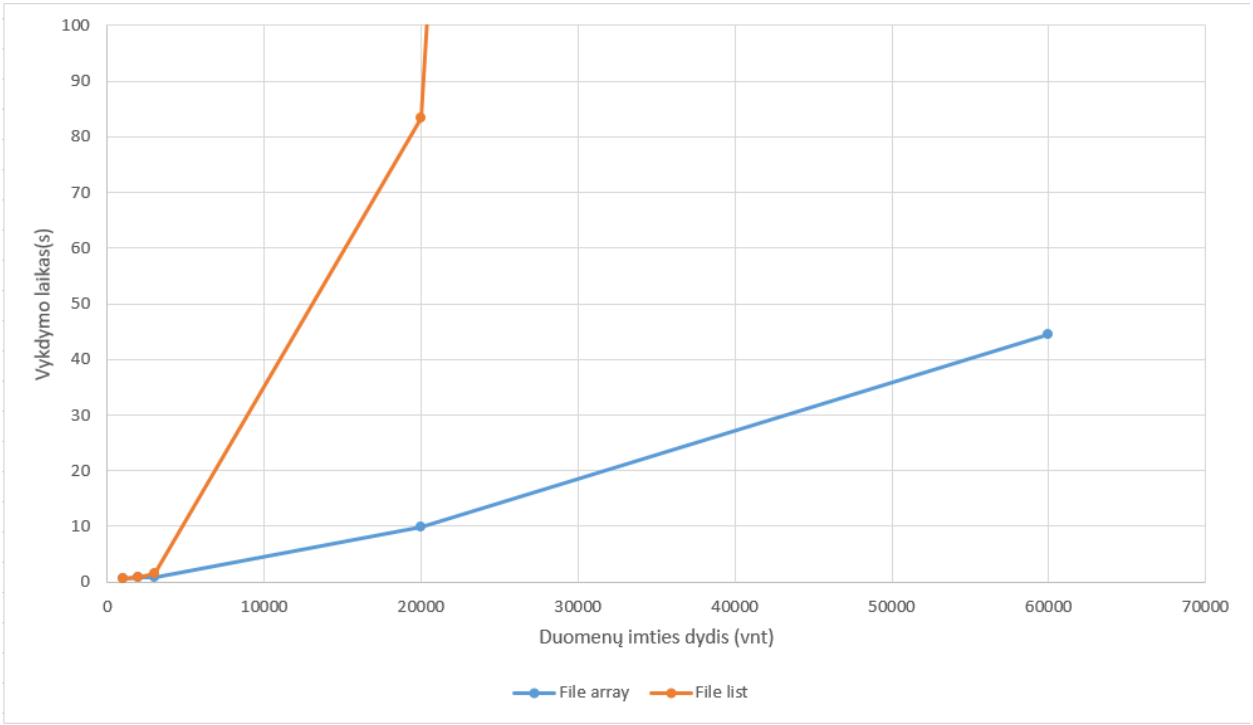
Išorinės atminties algoritmo analizė:

Apskaičiuoti algoritmų sudėtingumai:

Masyvas : n^2

Sąrašas : n^3

```
Array
100 - 00:00:00.4967801
200 - 00:00:00.7359592
300 - 00:00:00.9254607
2000 - 00:00:09.8553651
6000 - 00:00:44.4681253
LinkedList
100 - 00:00:00.5209103
200 - 00:00:00.8165526
300 - 00:00:01.5694866
2000 - 00:01:23.4318119
6000 - 00:31:59.8491978
Press any key to continue . . .
```



Išvados

BucketSort algoritmas nėra sudėtingas, veikia gana greitai. Tačiau nenaudojant kito rušiavimo algoritmo sudarytiems kibirams rūšiuoti, jis tik dalinai išrūšiuoja duomenis. Nuo kito rūšiavimo algoritmo priklauso ir galutinio algoritmo sudėtingumas. Algoritmas veikia gana greitai, nes duomenys prieš galutinį rūšiavimą yra arti vienas kito ir beveik išrūšiuoti. Operatyviojoje atmintyje algoritmas su masyvais dirba greičiau, nei su linkedlist. Masyvas greičiau veikia ir išorinėje atmintyje.

Laboratorinio darbo eiga buvo kėbli. Dirant su operatyviąja atmintimi sunkumų nekilo, tačiau prie antros darbo dalies užtrukau labai ilgai, laboratirinį darbą teko daryti kelis kartus iš naujo.