KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Algoritmų sudarymas ir analizė (P170B400)

Laboratorinių darbų ataskaita

Atliko:
IFF-1/6 gr. studentas
Lukas Kuzmickas
2023 m. vasario 22 d.
Priėmė:
Doc. Pilkauskas Vytautas

KAUNAS 2023

TURINYS

1. 1 LD laboratorinis darbas

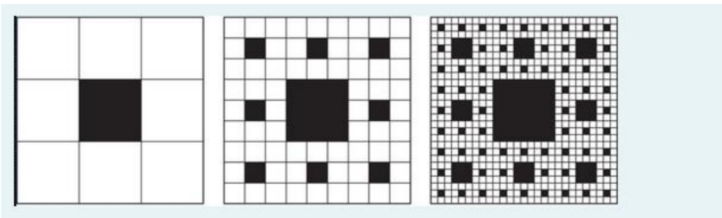
1.1. Pradinė užduotis

Kiekvienai rekurentinei lygčiai (gautai atlikus užduoties pasirinkimo testą):

- Realizuoti metodą, kuris atitiktų pateiktos rekurentinės lygties sudėtingumą, t. y. programinio kodo rekursinių iškvietimų ir kiekvieno iškvietimo metu atliekamų veiksmų priklausomybę nuo duomenų. Metodas per parametrus turi priimti masyvą, kurio duomenų kiekis yra rekurentinės lygties kintamasis **n** (arba masyvą ir indeksų rėžius, kurie atitinkamai nurodo masyvo nagrinėjamų elementų indeksus atitinkamame iškvietime) (2 balai).
- Kiekvienam realizuotam metodui atlikti programinio kodo analizę, parodant jog jis atitinka pateiktą rekurentinę lygtį (1 balas).
- Išspręskite rekurentinę lygtį ir apskaičiuokite jos asimptotinį sudėtingumą (taikoma pagrindinė teorema, medžių ar kitas sprendimo metodas) (1 balas)
- Atlikti eksperimentinį tyrimą (našumo testus) ir patikrinkite ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus (1 balas).

Naudojant rekursiją ir nenaudojant grafinių bibliotekų sudaryti nurodytos struktūros BMP formato (gautą atlikus užduoties pasirinkimo testą):

- Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas. (3 balai)
- Eksperimentiškai nustatykite darbo laiko ir veiksmų skaičiaus priklausomybę nuo generuojamo paveikslėlio dydžio (taškų skaičiaus). Gautus rezultatus atvaizduokite grafikais. Grafiką turi sudaryti nemažiau kaip 5 taškai ir paveikslėlio taškų skaičius turi didėti proporcingai (kartais). (1 balas)
- Analitiškai įvertinkite procedūros, kuri generuoja paveikslėlį, veiksmų skaičių sudarydami rekurentinę lygtį ir ją išspręskite. Gautas rezultatas turi patvirtinti eksperimentinius rezultatus. (1 balas)



$$T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + n$$

$$T(n) = T(n-9) + T(n-1) + 1$$

$$T(n) = 2 * T\left(\frac{n}{8}\right) + n^4$$

1 pav. Individualios užduoties variantas.

1.2. Užduoties sprendimas ir rezultatai

1.2.1 Pirmosios rekurentinės lygties sprendimas

Realizuojame metodą, kuris atitinka rekurentinės lygties $T(n) = 2 * T\left(\frac{n}{8}\right) + n^4$ sudėtingumą (2 paveikslėlis).

```
3 references
static ulong firstRecurrent(ulong[] arr)
{
    //T(n)=2*T(n/8)+n^4
    ulong result = 0;                    //c1 | 1
    if (arr.Length >= 8)                 //c2 | 1
    {
        result += firstRecurrent(new ulong[arr.Length / 8]);       //T(n/8) | 1
        result += firstRecurrent(new ulong[arr.Length / 8]);       //T(n/8) | 1
        for (int i = 0; i < arr.Length; i++)            //c5 |  (n+1)
            for (int j = 0; j < arr.Length; j++)        //c6 |  (n+1)
                for (int k = 0; k < arr.Length; k++)    //c7 |  (n+1)
                    for (int l = 0; l < arr.Length; l++) //c8 |  (n+1)
                        result += 1;                     //c9 | (n)^4
    }
    return result;                                       //c10 | 1
}
// T(n) = c1 + c2 + (T(n/8)) + (T(n/8)) + c5(n+1) + c6(n+1) + c7(n+1) + c8(n+1) + c9(n^4) + c10
// T(n) = T(n/8) + n+1 + n^4
```

2 pav. Pirmosios rekurentinės lygties metodas.

Atliekame programinio kodo analizę – nustatome šio metodo (3 paveikslėlis) kainą ir kiekį, taip parodome, kad metodas atitinka rekurentinę lygtį.

Vertinamas programos fragmentas

```
static ulong T1(ulong[] arr)
{
    ulong result = 0;
    if (arr.Length >= 8)
    {
        result += T1(new ulong[arr.Length / 8]);
        result += T1(new ulong[arr.Length / 8]);
        for (int i = 0; i < arr.Length; i++)
            for (int j = 0; j < arr.Length; j++)
                for (int k = 0; k < arr.Length; k++)
                    for (int l = 0; l < arr.Length; l++)
                        result += 1;
    }
    return result;
}
```

Kaina	Kiekis
c1	1
c2	1
T(n/8)	1
T(n/8)	1
c3;c4;c5	1;n+1;n
c6;c7;c8	n;(n^2)+1;n^2
c9;c10;c11	n^2;(n^3)+1;n^3
c12;c13;c14	n^3;(n^4)+1;n^4
c15	n^4
c16	1

3 pav. Pirmojo metodo kodo analizė

c_i – konstantiniai veiksmai O(1).
n – elementų skaičius.

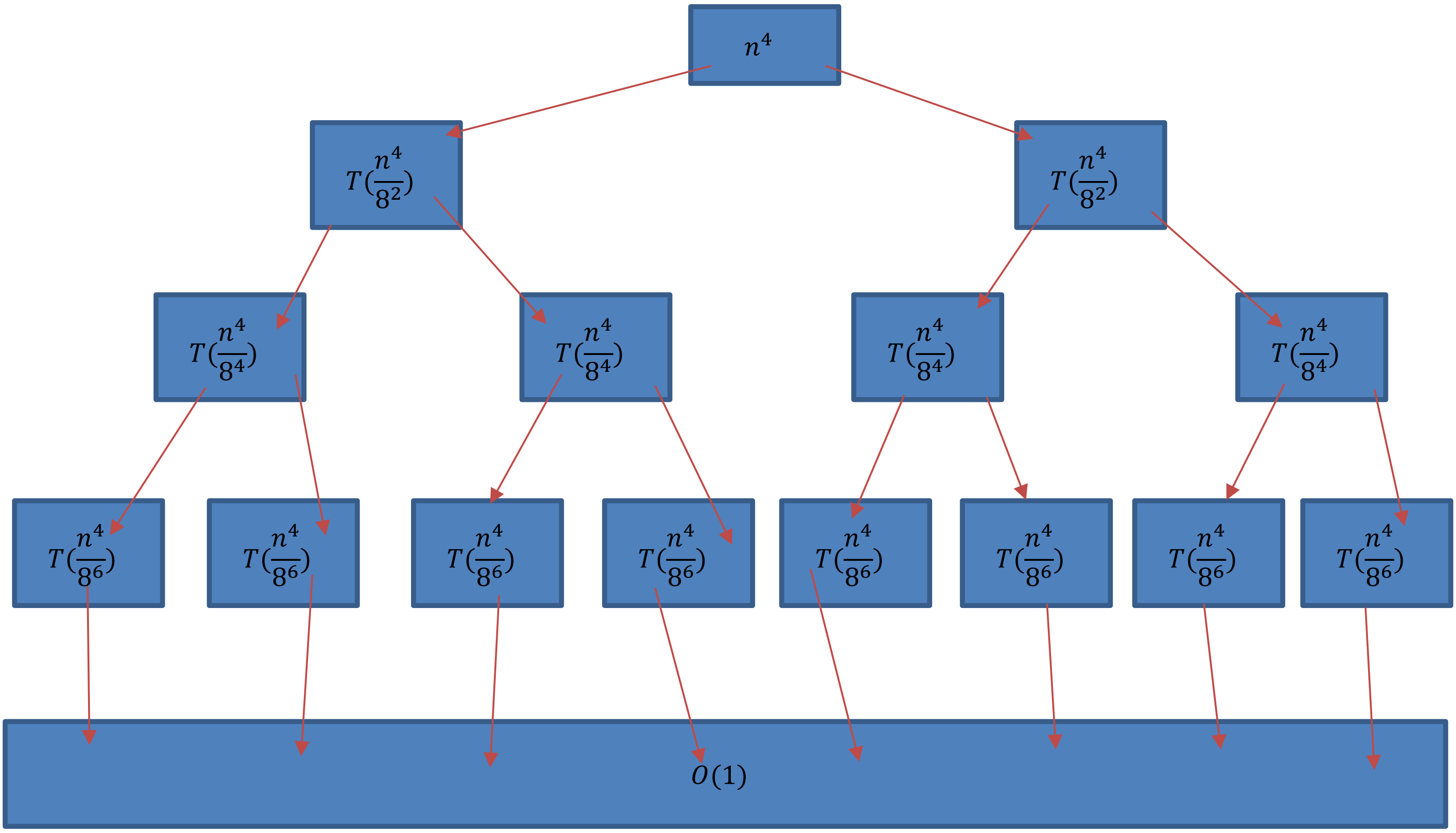
$$T(n) = c_1 + c_2 + T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + c_3 + c_4 n + c_5 n + c_6 n + c_7 n^2 + c_8 n^2 + c_9 n^2 + c_{10} n^3 + c_{11} n^3 + c_{12} n^3 + c_{13} n^4 + c_{14} n^4 + c_{15} n^4 + c_{16}$$
$$= T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + c_1 + c_2 + c_{16} + c_3 + n(c_5 + c_6) + n^2(c_7 + c_8 + c_9) + n^3(c_{10} + c_{11} + c_{12}) + n^4(c_{13} + c_{14} + c_{15});$$

Atmetus konstantas, suprastinus šią išraišką, gauname panašią rekurentinę lygtį į mūsų:
$T(n) = 2 * T\left(\frac{n}{8}\right) + n^4;$

Rekurentinės lygties sprendimas:

Sprendžiame rekurentinę lygtį: $T(n) = 2 * T\left(\frac{n}{8}\right) + n^4;$

Naudosime medžio metodą, sudarome sprendimo medį šiai rekurentinei lygčiai.



Rekurentinis sprendimo medis

Toliau suskaičiuojame, kiekvieno iteracijos lygio svorių sumą.

0 lygis: n^4
1 lygis: $2\frac{n^4}{8^2}$
2 lygis: $4\frac{n^4}{8^4}$
3 lygis: $8\frac{n^4}{8^6}$

Pagal šiuos iteracijų dėsningumus, sudarome lygtį jų sumavimui:

$$T(n) = \sum_{i=0}^{k} (\frac{2}{64})^i n^4 = n^4 \sum_{i=0}^{k} (\frac{2}{64})^i$$

Nagrinėjame pati blogiausią atvejį.

$$T(n) = n^4 \sum_{i=0}^{k} (\frac{2}{64})^i < n^4 \sum_{i=0}^{\infty} (\frac{2}{64})^i = \frac{n^4}{1 - \frac{2}{64}} = \frac{32}{31} n^4;$$

$T(n) = O(n^4);$

Toliau nagrinėjame geriausią atvejį.

Pastebime, kad su kiekviena iteracija, uždavinys pamažėja aštuonis kartus, todėl medžio aukštis: $h = log_8 n$;

Sprendžiame uždavinį:

$$T(n) = n^4 \sum_{i=0}^{h} (\frac{2}{64})^i = n^4 \frac{(\frac{2}{64})^{[log_8 n]+1} - 1}{\frac{2}{64} - 1} = \frac{32}{31} n^4 \left(1 - (\frac{2}{64})^{[log_8 n]+1}\right);$$

Kadangi, $(\frac{2}{64})^{[log_8 n]+1}$ yra mažėjanti funkcija nuo tada, kai n>8

$1 - \left(\frac{2}{64}\right)^{[log_8 n]+1} > 1 - \left(\frac{2}{64}\right)^{[log_8 8]+1} = \frac{1023}{1024};$

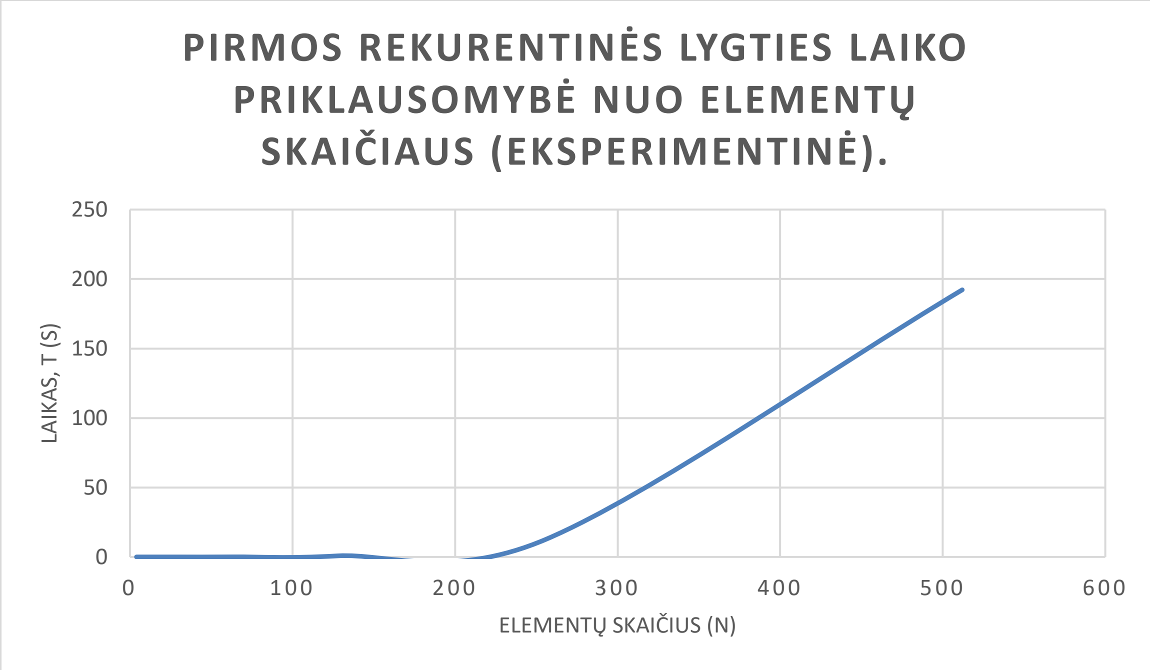
$T(n) > \frac{1023}{1024} * \frac{32}{31} * n^4 = \frac{33}{32} n^4;$

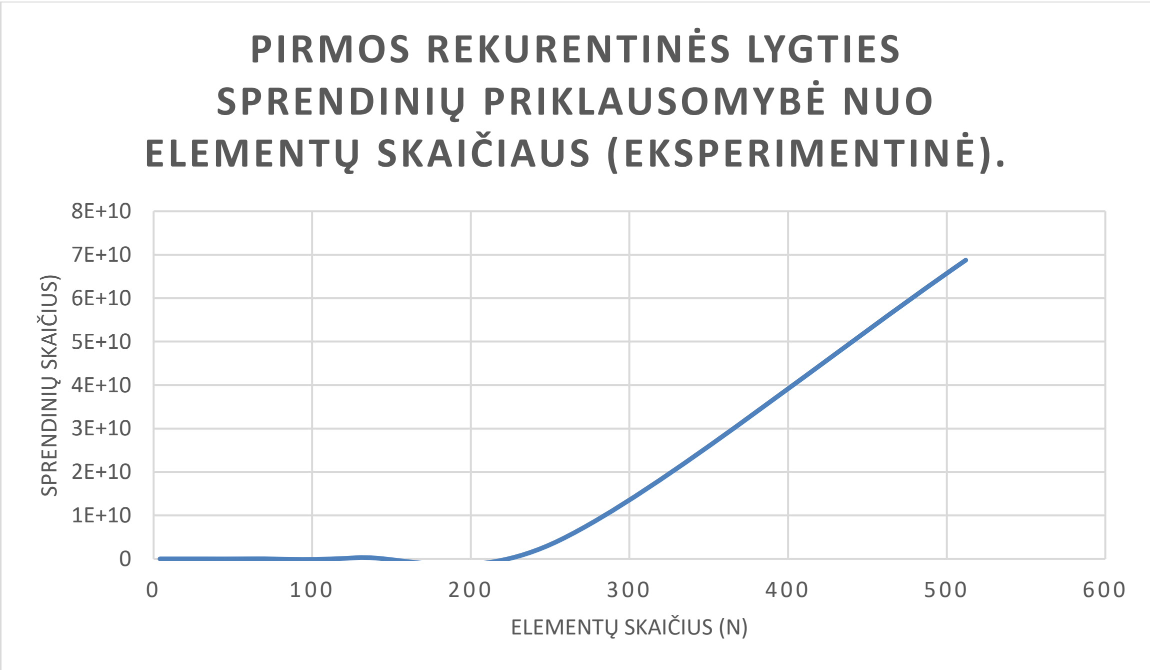
$T(n) = \lambda(n^4);$

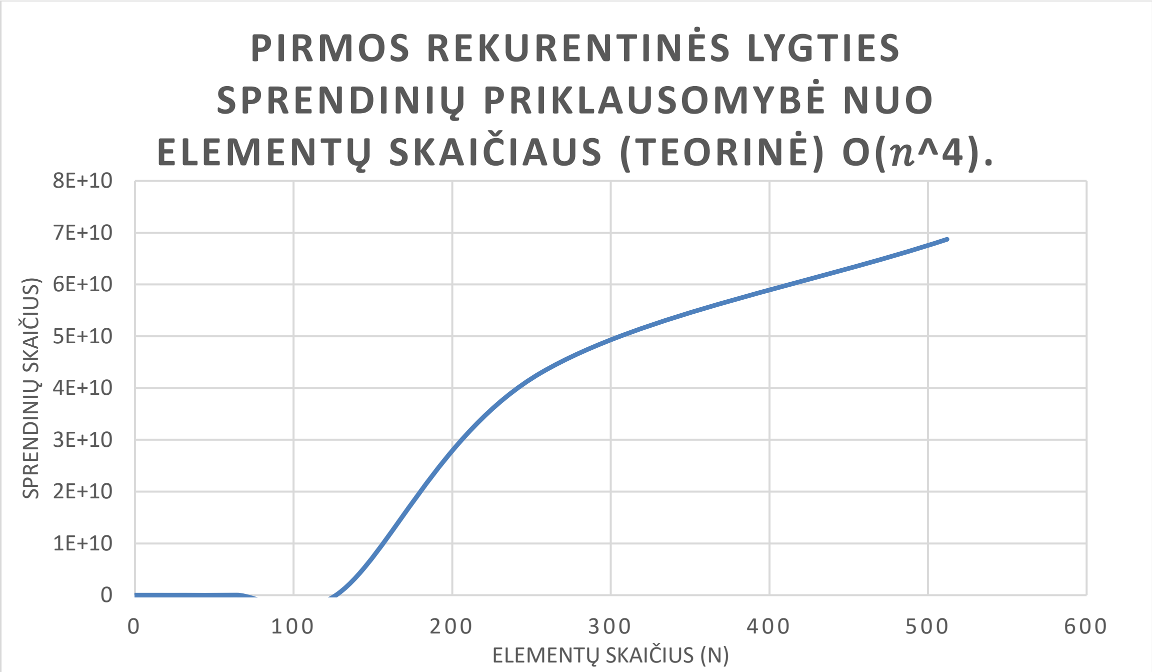
Kadangi galioja $T(n) = O(n^4)\ ir\ T(n) = \lambda(n^4)$, tai galioja ir $T(n) = \theta(n^4)$,nes

$$\frac{33}{32} n^4 < T(n) < \frac{32}{31} n^4, kai\ visiems\ n > 8$$

Našumo testai:






PIRMOS REKURENTINĖS LYGTIES SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ) O(n^4).
SPRENDINIŲ SKAIČIUS)
8E+10
7E+10
6E+10
5E+10
4E+10
3E+10
2E+10
1E+10
0
0
100
200
300
400
500
600
ELEMENTŲ SKAIČIUS (N)

Pirmosios rekurentinės lygties kodas:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace L1_rekurentinės
{
    internal class Program
    {
        static void Main(string[] args)
        {
            //LAIKO TESTAVIMAS
            ulong times = 2;
            ulong n = 2;
            var timer = Stopwatch.StartNew();
            ulong result;
            //Pirmos rekurentines testavimas - O(n^4)
            while (n <= 512)
            {
                timer.Start();
                result = firstRecurrent(new ulong[n]);
                timer.Stop();
                PrintToFile("pirmosRekurentines.csv",
$"{n};{timer.Elapsed.TotalSeconds};{result}", true);
                n = n * times;
                Console.WriteLine();
                System.Console.WriteLine(n);
            }

        }

        static ulong firstRecurrent(ulong[] arr)
        {
            //T(n)=2*T(n/8)+n^4
            ulong result = 0;                    //c1 | 1
            if (arr.Length >= 8)                 //c2 | 1
            {
                result += firstRecurrent(new ulong[arr.Length / 8]);
//T(n/8)  | 1
                result += firstRecurrent(new ulong[arr.Length / 8]);
//T(n/8)  | 1
                for (int i = 0; i < arr.Length; i++)            //c5 |  (n+1)
                    for (int j = 0; j < arr.Length; j++)        //c6 |  (n+1)
                        for (int k = 0; k < arr.Length; k++)    //c7 |  (n+1)
                            for (int l = 0; l < arr.Length; l++) //c8 |  (n+1)
                                result += 1;                      //c9 |  (n)^4
            }
            return result;                                        //c10 | 1
        }

        static void PrintToFile(string fileName, string line, bool append = false)
        {
            using (StreamWriter sw = new StreamWriter(fileName, append))
            {
                sw.WriteLine(line);
            }
        }

    }
}
```

1.2.2 *Antrosios rekurentinės lygties sprendimas*

Realizuojame metodą, kuris atitinka rekurentinės lygties $T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + n$ sudėtingumą (4 paveikslėlis).

```
3 references
static ulong secondRecurrent(ulong[] arr)
{
    //T(n)=T(n/6)+ T(n/7)+ n
    ulong result = 0;                           //c1 | 1
    if (arr.Length > 0)                         //c2 | 1
    {
        result += secondRecurrent(new ulong[arr.Length / 6]);       //T(n/6) | 1
        result += secondRecurrent(new ulong[arr.Length / 7]);       //T(n/7) | 1
        for (int i = 0; i < arr.Length; i++)         //c5 | n+1
            result += 1;                        //c6  | n
    }
    return result;                              //c7 | 1
}
// T(n) = c1 + c2 + T(n/6) + T(n/7) + (n+1) + n + c7 = T(n/6) + T(n/7) + (n+1) + n
```

4 pav. Antrosios rekurentinės lygties metodas.

Atliekame programinio kodo analizę – nustatome šio metodo (5 paveikslėlis) kainą ir kiekį, taip parodome, kad metodas atitinka rekurentinę lygtį.

Vertinamas programos fragmentas	Kaina	Kiekis
`static ulong T2(ulong[] arr)`		
`{` `ulong result = 0;`	c1	1
`if (arr.Length > 0)`	c2	1
`{`		
`result += T2(new ulong[arr.Length / 6]);`	T(n/6)	1
`result += T2(new ulong[arr.Length / 7]);`	T(n/7)	1
`for (int i = 0; i < arr.Length; i++)`	c3;c4;c5	1;n+1;n
`result += 1;`	c6	n
`}`		
`return result;` `}`	c7	1

5 pav. Antrojo metodo kodo analizė

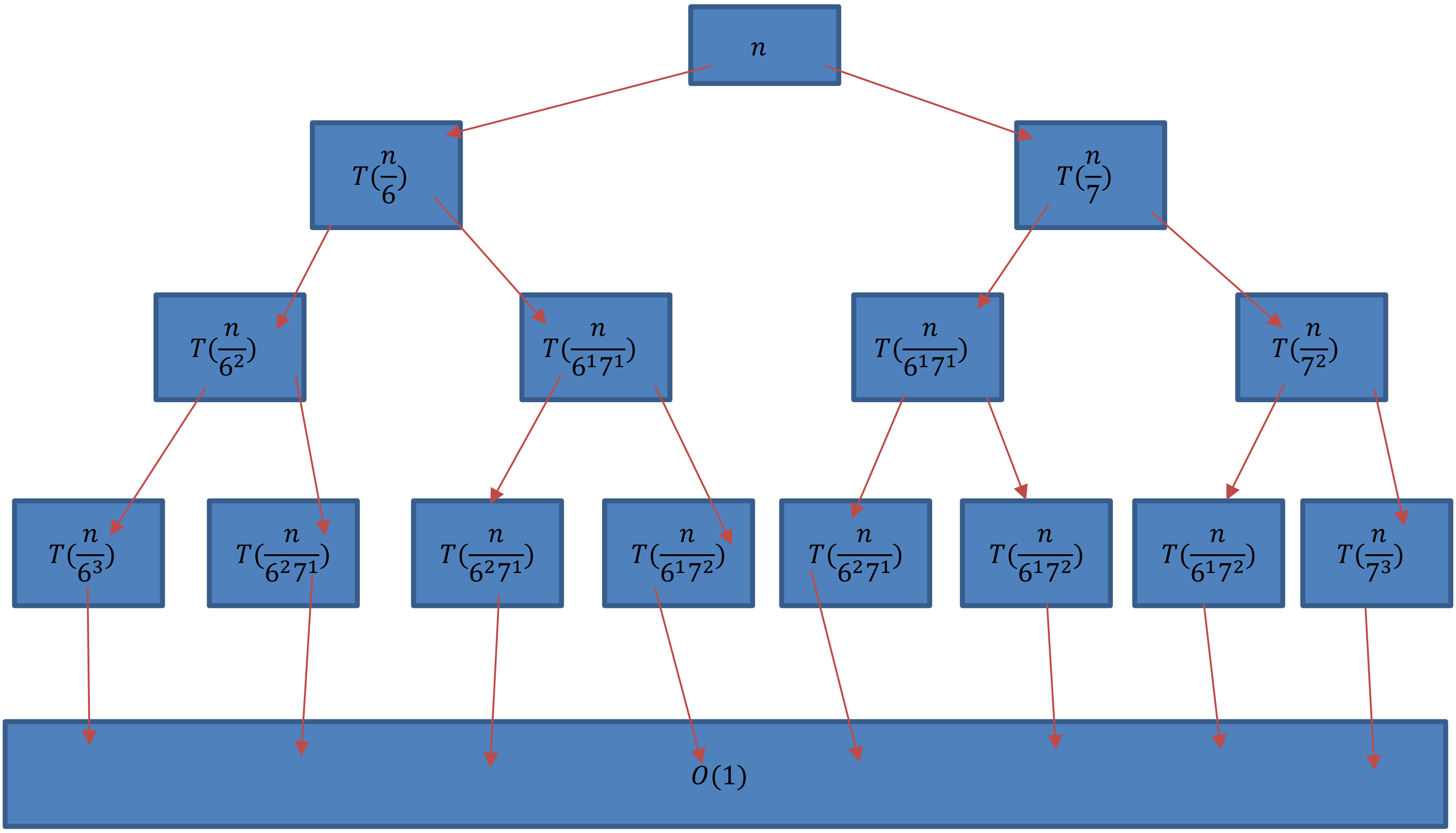
$$T(n) = c_1 + c_2 + T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + c_3 + c_4 n + c_5 n + c_6 n + c_7$$
$$= T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + c_1 + c_2 + c_3 + c_7 + n(c_5 + c_6);$$

Atmetus konstantas, gauname panašią rekurentinę lygtį į mūsų:
$T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + n;$

Rekurentinės lygties sprendimas:

Sprendžiame rekurentinę lygtį: $T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + n;$

Naudosime medžio metodą, sudarome sprendimo medį šiai rekurentinei lygčiai.



Rekurentinis sprendimo medis

Toliau suskaičiuojame, kiekvieno iteracijos lygio svorių sumą.

0 lygis: n
1 lygis: $\left(\frac{1}{6} + \frac{1}{7}\right) n$
2 lygis: $\left(\frac{1}{6^2} + 2\frac{1}{6^1 7^1} + \frac{1}{7^2}\right) n$
3 lygis: $\left(\frac{1}{6^3} + \frac{3}{6^2 7^1} + \frac{3}{6^1 7^2} + \frac{1}{7^3}\right) n$

Pagal šiuos iteracijų dėsningumus, sudarome lygtį jų sumavimui (Niutono binomas).

$$(\frac{1}{6} + \frac{1}{7})^i = (\frac{13}{42})^i;$$

Šis medis nėra simetrinis, todėl medžio aukštis h yra $[log_7 n] \leq h \leq [log_6 n]$;

Nagrinėjame pati blogiausią atvejį:

$$T(n) = n\sum_{i=0}^{h}(\frac{13}{42})^i < n\sum_{i=0}^{\infty}(\frac{13}{42})^i = \frac{n}{1-\frac{13}{42}} = \frac{42}{29}n;$$

$T(n) = O(n);$

Toliau nagrinėjame geriausią atvejį.

Šis medis nėra simetrinis, todėl medžio aukštis h yra $[log_7 n] \leq h \leq [log_6 n]$;

Sprendžiame uždavinį:

$$T(n) = n\sum_{i=0}^{h}(\frac{13}{42})^i = n\frac{(\frac{13}{42})^{[log_7 n]+1} - 1}{\frac{13}{42} - 1} = \frac{42}{29}n\left(1 - (\frac{13}{42})^{[log_7 n]+1}\right) \geq n;$$

, *nes* $1 - \left(\frac{13}{42}\right)^{[log_7 n]+1} \geq \frac{29}{42}$, *kai n > 0;*

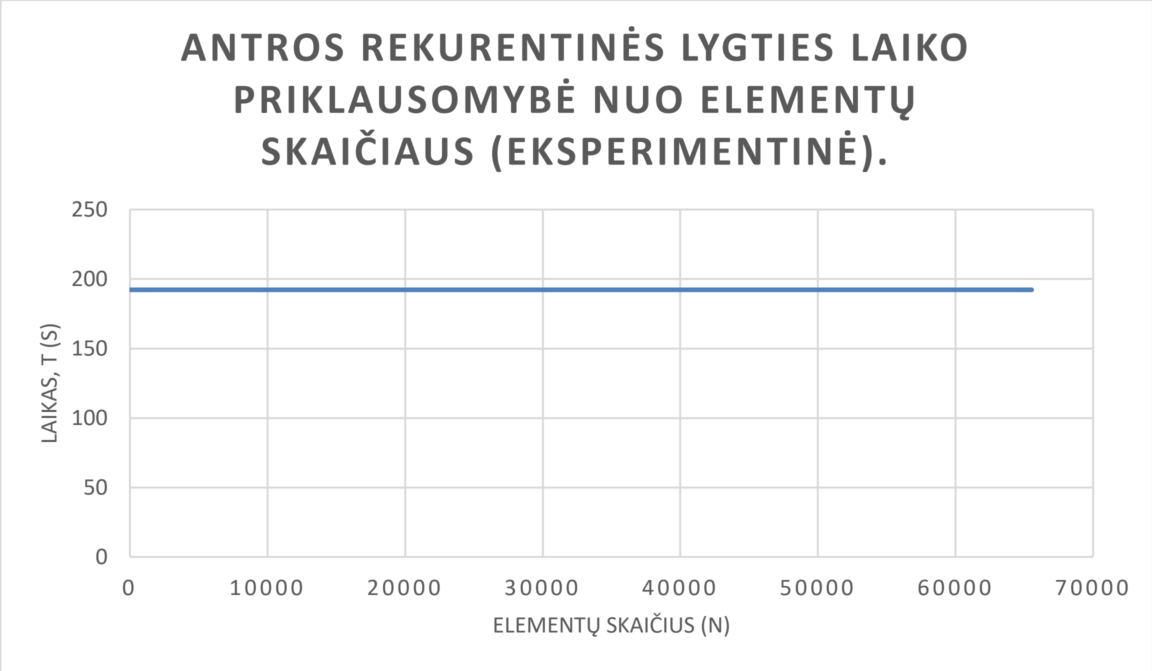
$T(n) = \frac{42}{29} * \frac{29}{42} * n = n;$

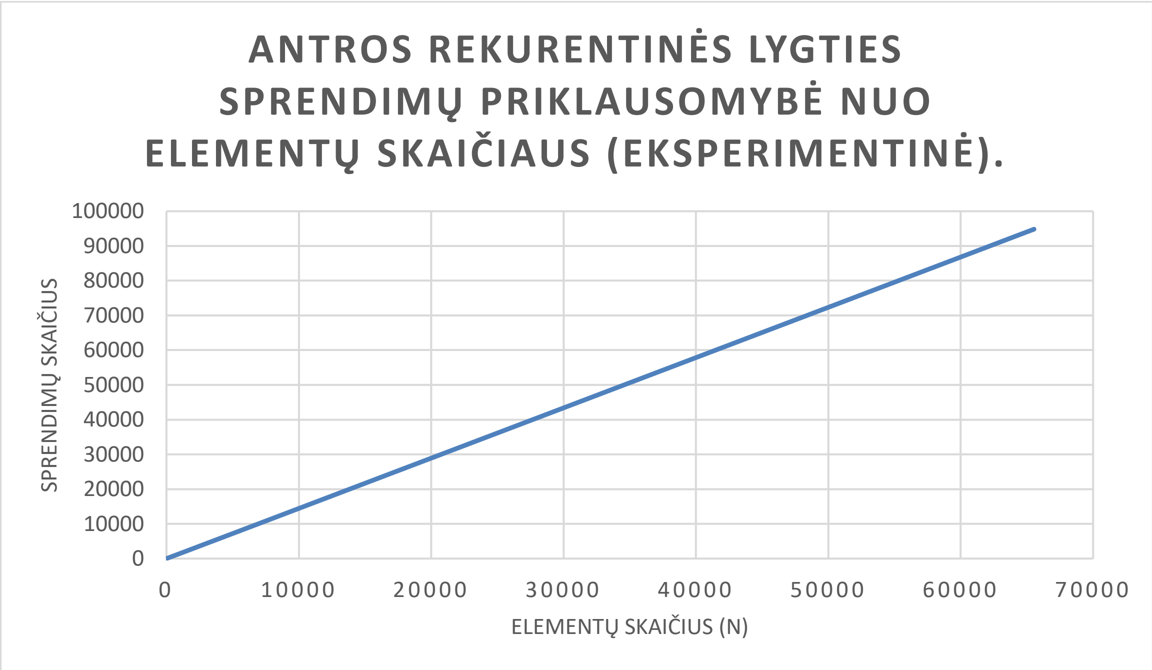
$$T(n) = \lambda(n);$$

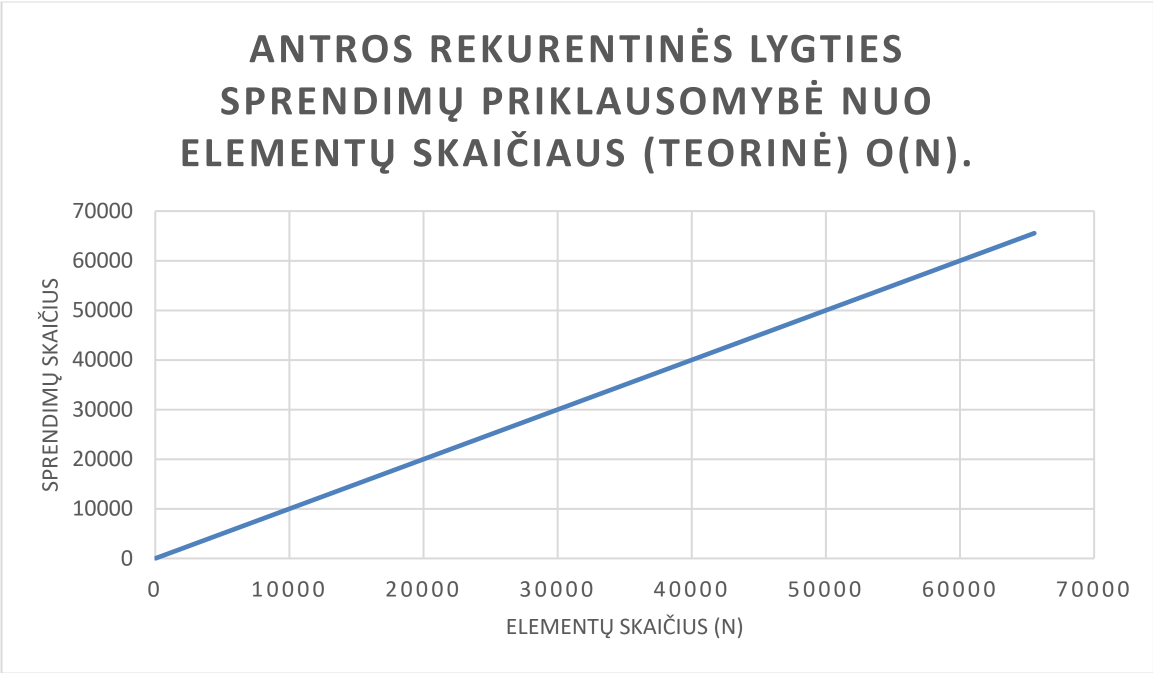
Kadangi galioja $T(n) = O(n)$ *ir* $T(n) = \lambda(n)$, tai galioja ir $T(n) = \theta(n)$,nes

$$n < T(n) < \frac{42}{29}n, kai\ visiems\ n > 8$$

Našumo testai:






ANTROS REKURENTINĖS LYGTIES SPRENDIMŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ) O(N).
SPRENDIMŲ SKAIČIUS
70000
60000
50000
40000
30000
20000
10000
0
0
10000
20000
30000
40000
50000
60000
70000
ELEMENTŲ SKAIČIUS (N)

Antrosios rekurentinės lygties kodas:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace L1_rekurentinės
{
    internal class Program
    {
        static void Main(string[] args)
        {
            //LAIKO TESTAVIMAS
            ulong times = 2;
            ulong n = 2;
            var timer = Stopwatch.StartNew();
            ulong result;

            //Antros rekurentines testavimas - O(n)
            while (n <= 256)
            {
                timer.Start();
                result = secondRecurrent(new ulong[n]);
                timer.Stop();
                PrintToFile("antrosRekurentines.csv",
$"{n};{timer.Elapsed.TotalSeconds};{result}", true);
                n = n * times;
                System.Console.WriteLine(n);
            }

        }

        static ulong secondRecurrent(ulong[] arr)
        {
            //T(n)=T(n/6)+ T(n/7)+ n
            ulong result = 0;                                   //c1 | 1
            if (arr.Length > 0)                                 //c2 | 1
            {
                result += secondRecurrent(new ulong[arr.Length / 6]);
//T(n/6) | 1
                result += secondRecurrent(new ulong[arr.Length / 7]);
//T(n/7) | 1
                for (int i = 0; i < arr.Length; i++)             //c5 | n+1
                    result += 1;                        //c6  | n
            }
            return result;                                      //c7 | 1
        }

        static void PrintToFile(string fileName, string line, bool append = false)
        {
            using (StreamWriter sw = new StreamWriter(fileName, append))
            {
                sw.WriteLine(line);
            }
        }


    }
}
```

1.2.3 Trečiosios rekurentinės lygties sprendimas

Realizuojame metodą, kuris atitinka rekurentinės lygties $T(n) = T(n-9) + T(n-1) + 1$ sudėtingumą (6 paveikslėlis).

```
3 references
static ulong thirdRecurrent(long[] arr, long startI, long endI)
{
    //T(n)=T(n-9)+ T(n-1)+1
    ulong result = 0;                                //c1 | 1
    if (endI >= 1)                                   //c2 | 1
    {
        result += thirdRecurrent(arr, 0, endI - 9);           //T(n-9) | 1
        result += thirdRecurrent(arr, 0, endI - 1);           //T(n-1) | 1
        result += 1;                                 //c5 | 1
    }
    return result;                                   //c6 | 1
}
// T(n) = c1 + c2 + T(n-9) + T(n-1) + c5 + c6 = T(n-9) + T(n-1)
```

6 pav. Trečiosios rekurentinės lygties metodas.

Atliekame programinio kodo analizę – nustatome šio metodo (7 paveikslėlis) kainą ir kiekį, taip parodome, kad metodas atitinka rekurentinę lygtį.

Vertinamas programos fragmentas	Kaina	Kiekis
`static ulong T3(long[] arr, long startI, long endI)`		
`{`		
`ulong result = 0;`	c1	1
`if (endI >= 1)`	c2	1
`{`		
`result += T3(arr, 0, endI - 9);`	T(n-9)	1
`result += T3(arr, 0, endI - 1);`	T(n-1)	1
`result += 1;`	c5	1
`}`		
`return result;`	c7	1
`}`		

7 pav. Trečiojo metodo kodo analizė.

$$T(n) = c_1 + c_2 + T(n-9) + T(n-1) + c_5 + c_7;$$

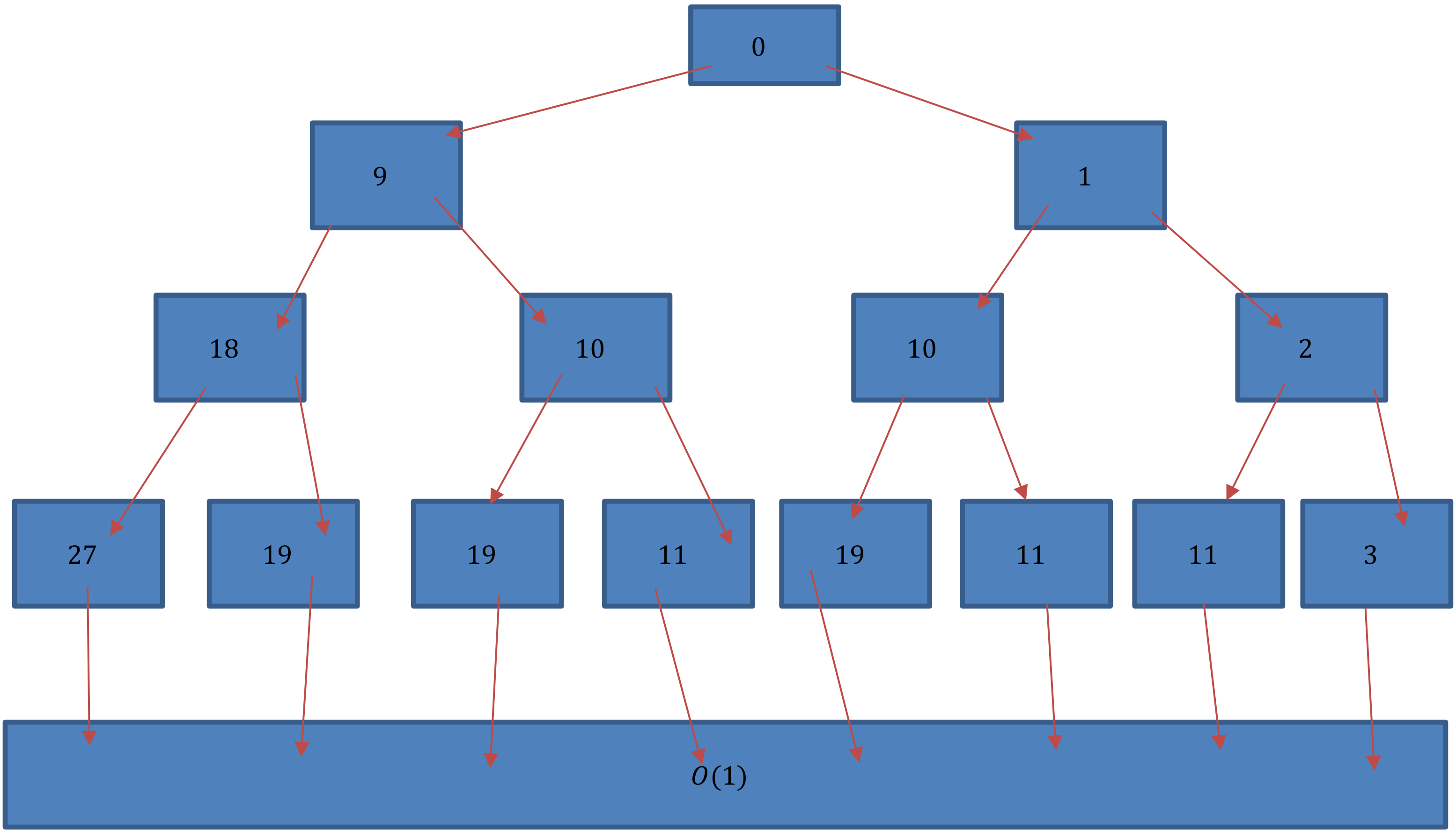
Atmetus konstantas, gauname panašią rekurentinę lygtį į mūsų:

$T(n) = T(n-9) + T(n-1) + 1;$

Rekurentinės lygties sprendimas:

Sprendžiame rekurentinę lygtį: $T(n) = T(n-9) + T(n-1) + 1;$

Naudosime medžio metodą, sudarome sprendimo medį šiai rekurentinei lygčiai.



Rekurentinis sprendimo medis

Sudarome rekursinę lygčių sistema, šiai rekurentinei lygčiai.

$$\begin{cases} S_0 = 0 \\ S_n = 2S_{n-1} + 10 * 2^{n-1} \end{cases}$$

Išsprendžiame lygčių sistemą ir gauname sprendinį, kurį galime įrodyti matematinės indukcijos metodu.

$$S_n = 10n2^{n-1};$$

$$S_{n+1} = 2 * 10n2^{n-1} + 10 * 2^n = 10(n+1)2^n;$$

Randame sprendinį.

$$T(n) = \sum_{i=0}^{h} (2^i n - 10i2^{i-1});$$

Žinome, kad $\sum_{i=0}^{n} i2^i = 2(n2^n - 2^n + 1)$ - (duota formulė).

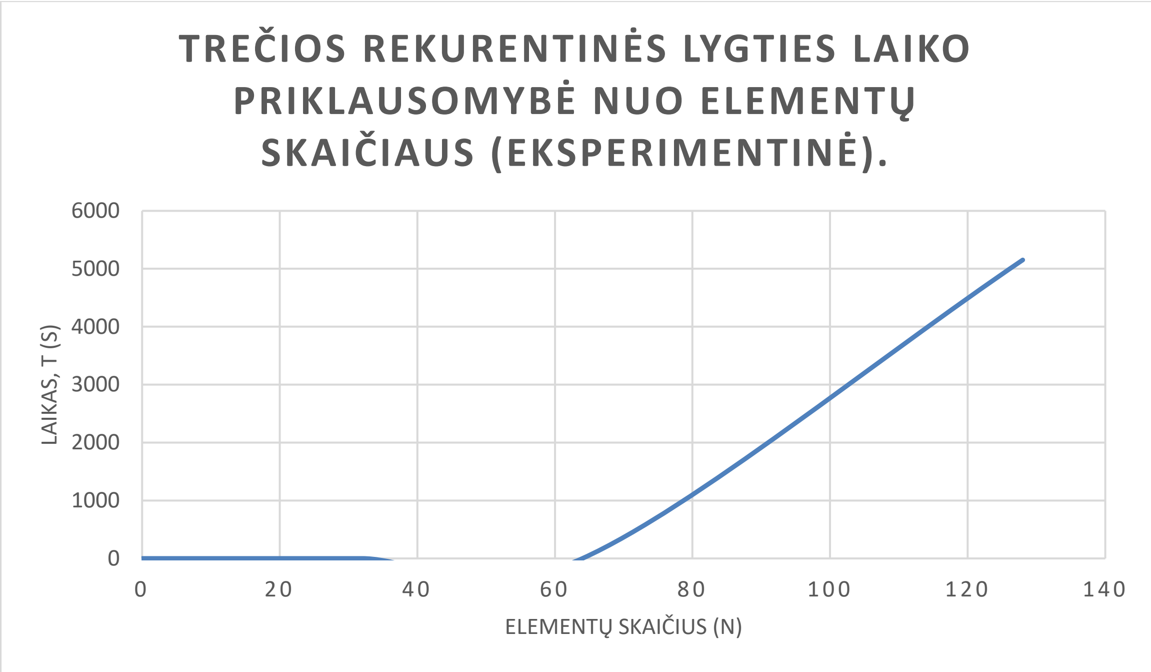
Sprendžiame uždavinį.

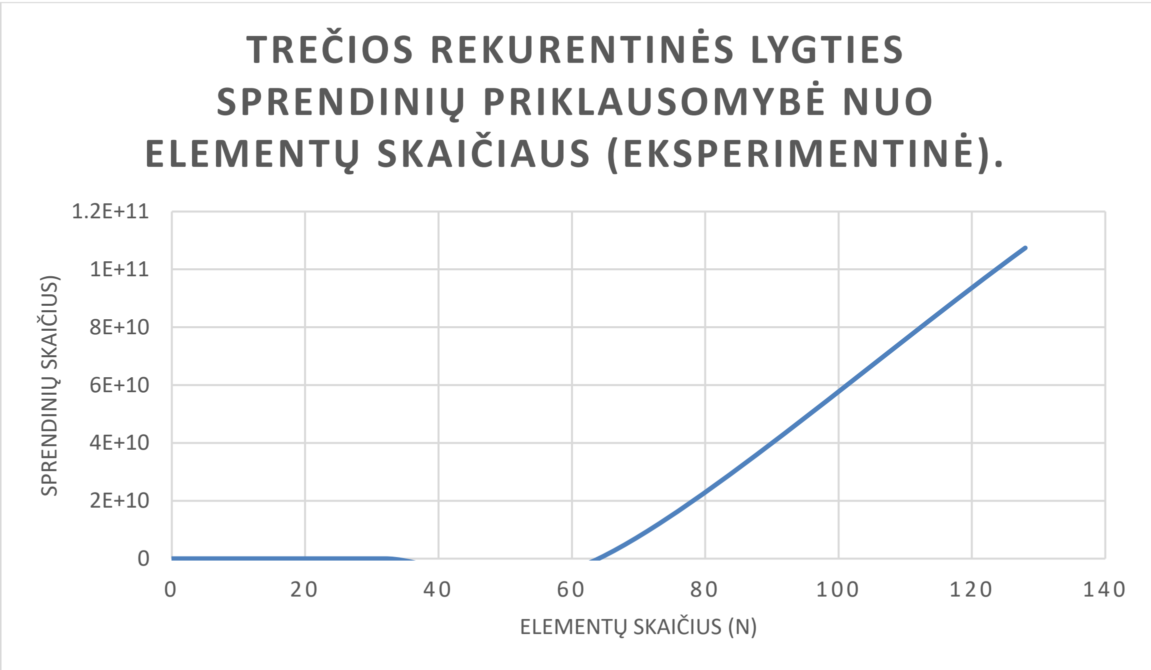
$\sum_{i=0}^{h} 2^i - 10\sum_{i=0}^{h} i2^{i-1} = (2^{h+1} - 1) - 10(h2^h - 2^h + 1)$;

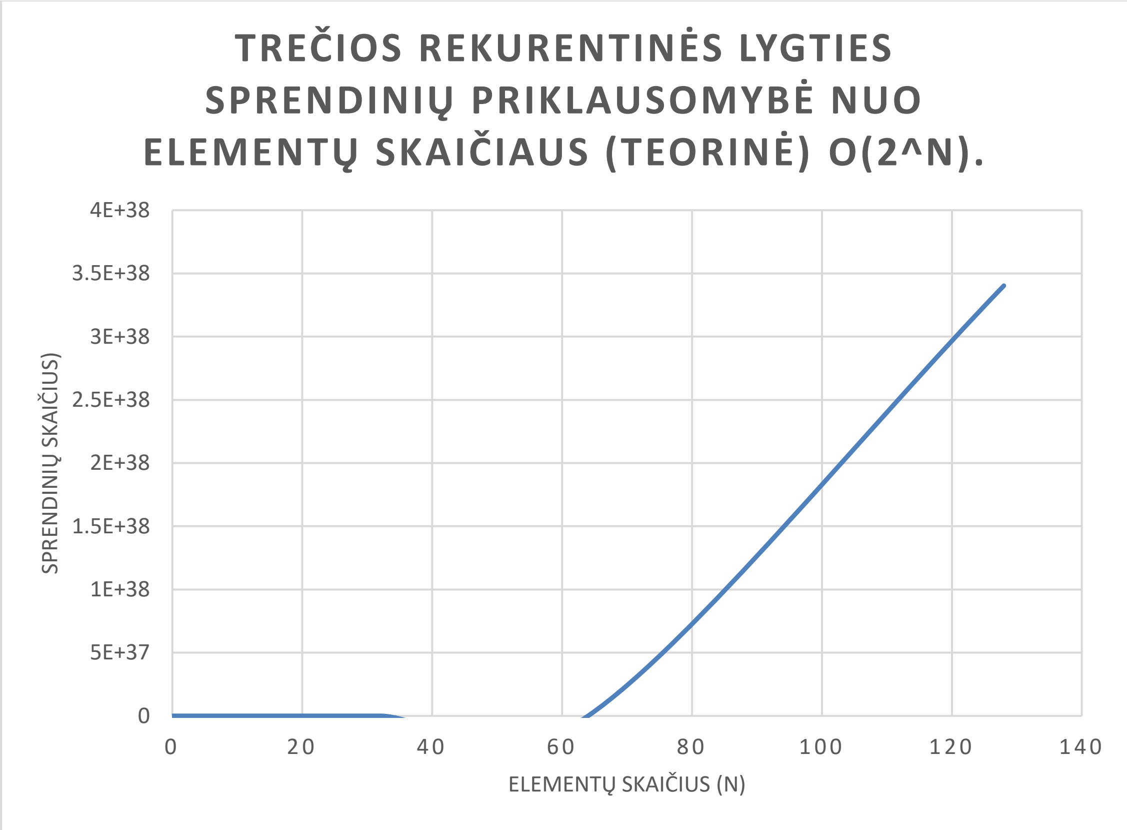
Apatinį įvertinimą galime įvertinti, kurio visos šakos yra neilgesnės, kaip $h = [\frac{n}{9}]$;

$T(n) = \lambda\left(2^{\frac{n}{9}}\right) \approx O(2^n)$;
Negavome asimptotiškai tikslaus įvertinimo.

Našumo testai:







Trečiosios rekurentinės lygties kodas:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace L1_rekurentinės
{
    internal class Program
    {
        static void Main(string[] args)
        {
            //LAIKO TESTAVIMAS
            ulong times = 2;
            var timer = Stopwatch.StartNew();
            ulong result;
            //Trecios rekurentines testavimas - O(2^n)
            long h = 3;
            while (h <= 100)
            {
                timer.Start();
                result = thirdRecurrent(new long[h], 0, h);
                timer.Stop();
                PrintToFile("treciosRekurentines.csv",
$"{h};{timer.Elapsed.TotalSeconds};{result}", true);
                h = h * (long)times;
                System.Console.WriteLine(h);
            }
        }
        static ulong thirdRecurrent(long[] arr, long startI, long endI)
        {
            //T(n)=T(n-9)+ T(n-1)+1
            ulong result = 0;                                   //c1 | 1
```

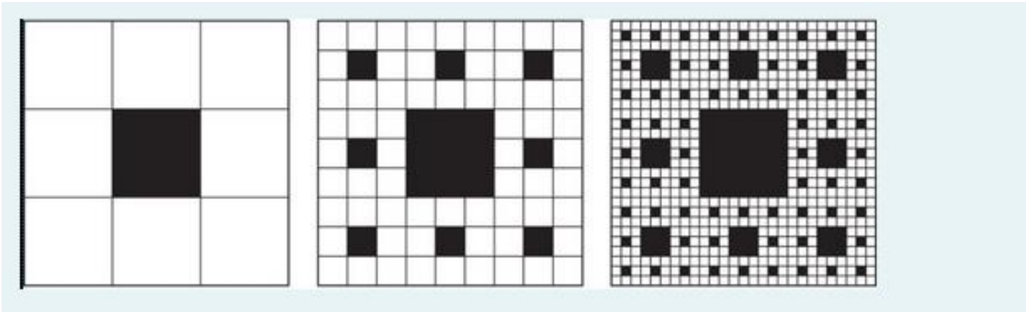
```
            if (endI >= 1)                                  //c2 | 1
            {
                result += thirdRecurrent(arr, 0, endI - 9);             //T(n-
9) | 1
                result += thirdRecurrent(arr, 0, endI - 1);             //T(n-
1) | 1
                result += 1;                                //c5 | 1
            }
            return result;                                  //c6 | 1
        }
        static void PrintToFile(string fileName, string line, bool append = false)
        {
            using (StreamWriter sw = new StreamWriter(fileName, append))
            {
                sw.WriteLine(line);
            }
        }

    }
}
```

1.2.4 BMP formato užduoties sprendimas

Naudojant rekursiją ir nenaudojant grafinių bibliotekų sudaryti nurodytos struktūros BMP formato (gautą atlikus užduoties pasirinkimo testą):

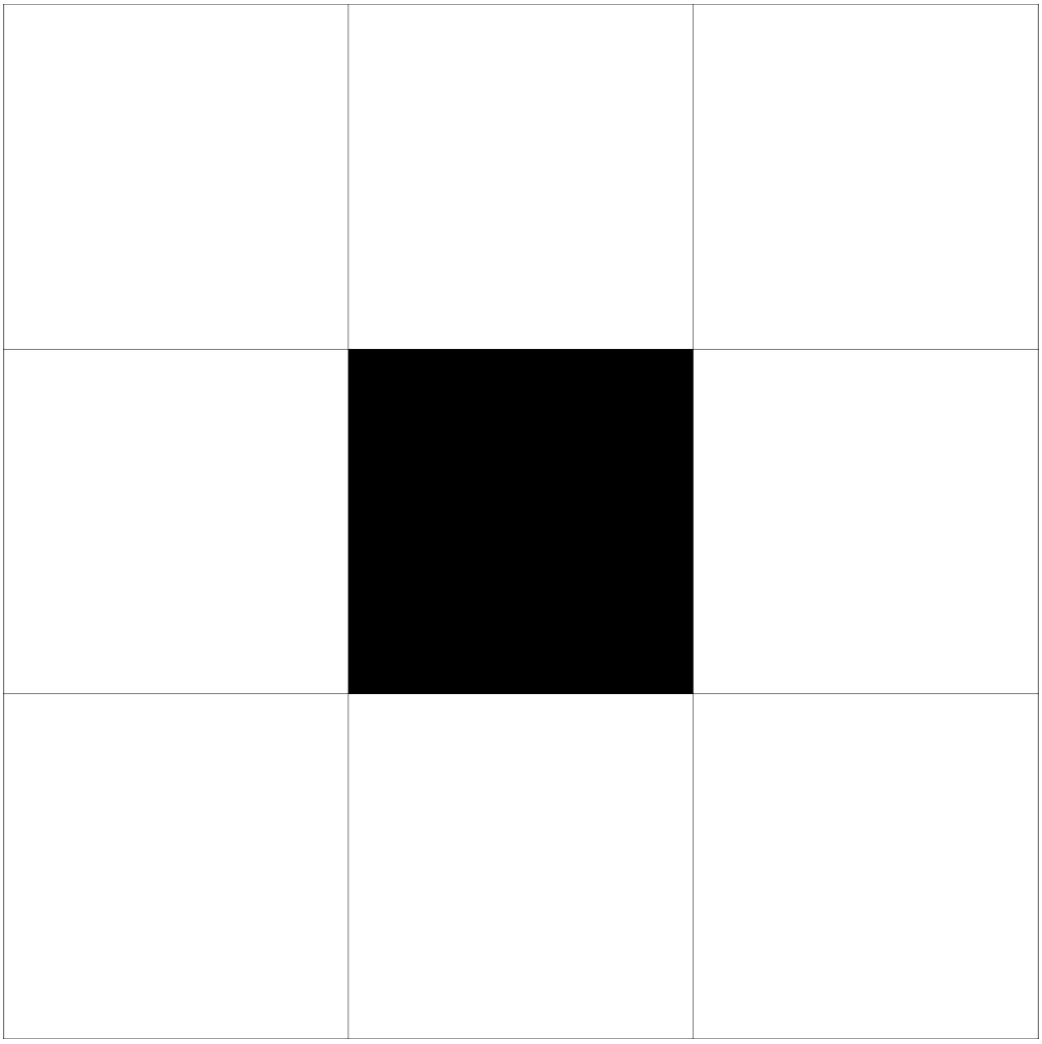
- Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas. (3 balai)
- Eksperimentiškai nustatykite darbo laiko ir veiksmų skaičiaus priklausomybę nuo generuojamo paveikslėlio dydžio (taškų skaičiaus). Gautus rezultatus atvaizduokite grafikais. Grafiką turi sudaryti nemažiau kaip 5 taškai ir paveikslėlio taškų skaičius turi didėti proporcingai (kartais). (1 balas)
- Analitiškai įvertinkite procedūros, kuri generuoja paveikslėlį, veiksmų skaičių sudarydami rekurentinę lygtį ir ją išspręskite. Gautas rezultatas turi patvirtinti eksperimentinius rezultatus. (1 balas)



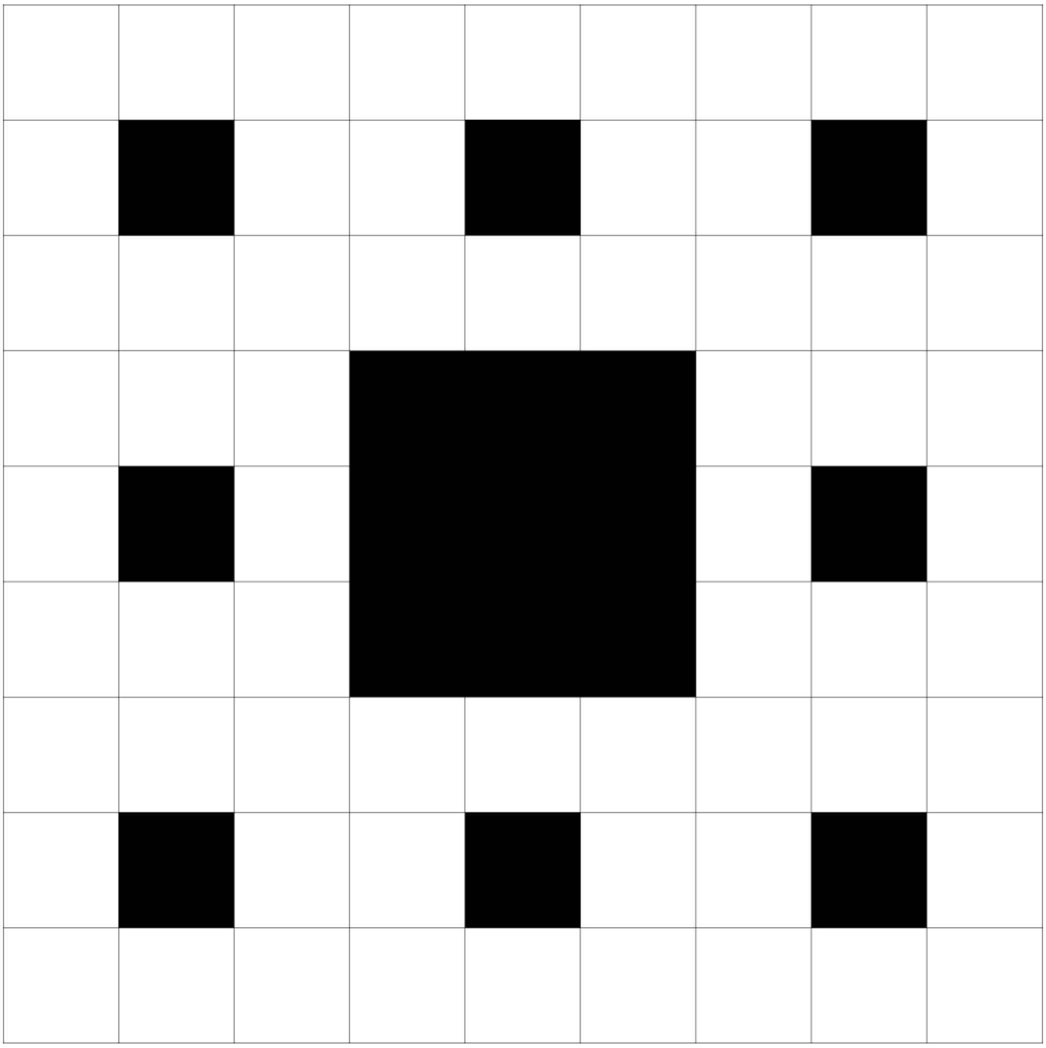
Pav. BMP formato užduotis.

Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas:

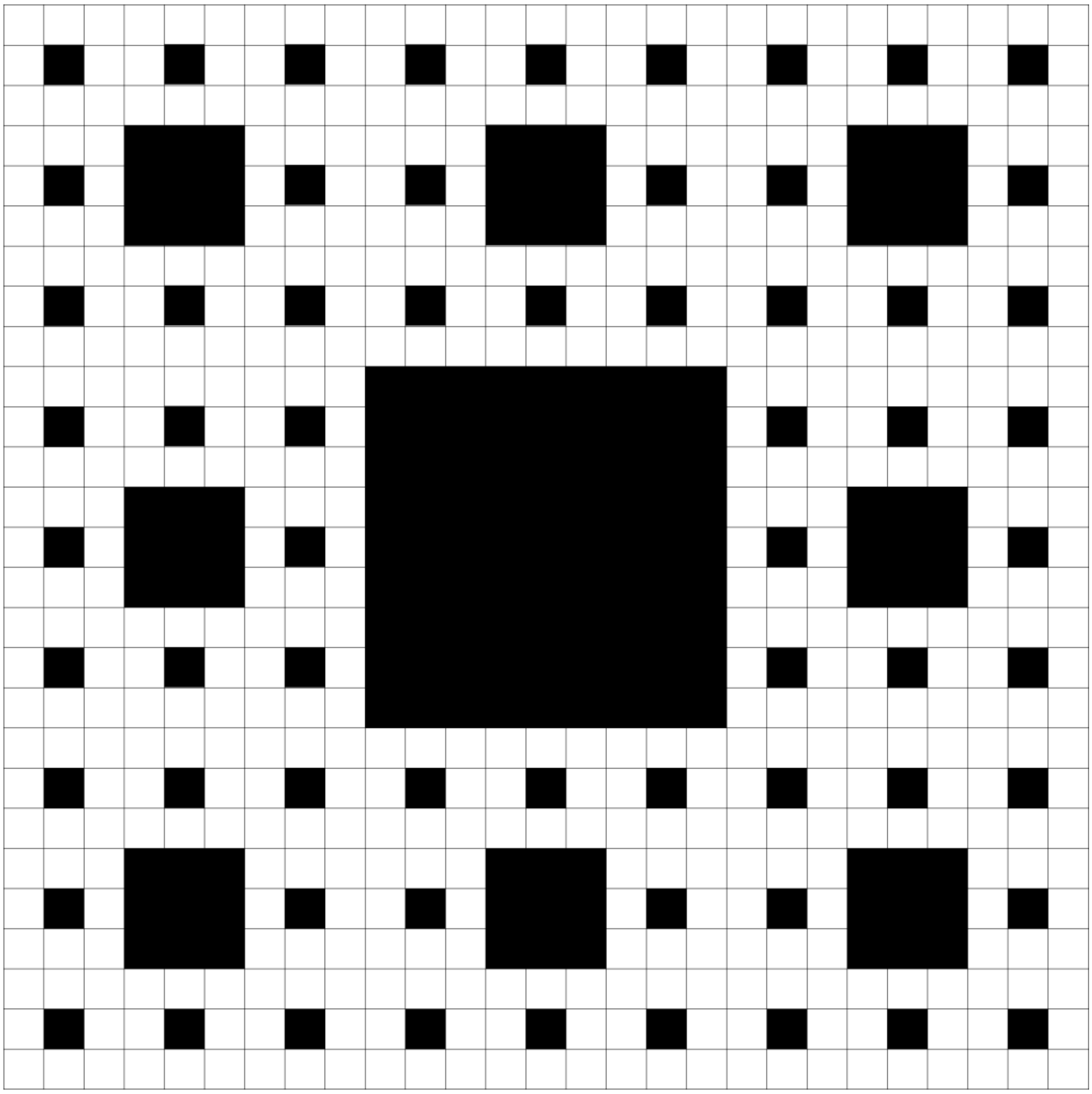
Pirma nuotrauka (image1.bmp)



Antra nuotrauka (image2.bmp)



Trečia nuotrauka image3.bmp)



BMP programos kodas (C#):

Program.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace L1_bmp
{
    public class Program
    {
        const string firstImage = "..//..//image1.bmp";

        static void Main(string[] args)
        {
            //pasirenkamas rekursijos gylis
            int recursionDepth = 3;
            uint pixelCount = 16000;
            bool correctionNeeded = false;
            if (pixelCount % 800 == 0)
            {
                correctionNeeded = true;
            }
            // ((1 * pixelCount + 31) / 32) * 4
            // (pixelCount / 32 + 1) * 4
            uint rowSize = ((1 * pixelCount + 31) / 32) * 4; //152
            byte[] structure = InOutUtils.MakeStructure(pixelCount, rowSize);
            byte[] image = new byte[pixelCount * rowSize];
            Stopwatch s = new Stopwatch();
            s.Start();
            TaskUtils.DrawSquare(image, 0, 0, (int)rowSize, (int)pixelCount,
(int)rowSize, recursionDepth);
            TimeSpan ts = s.Elapsed;
            Console.WriteLine(ts);
            InOutUtils.PrintImage(firstImage, structure, image);
        }
    }
}
```

TaskUtils.cs

```
using System.Runtime.CompilerServices;

namespace L1_bmp
{
    public static class TaskUtils
    {
        private static void DrawFilledSquare(byte[] image, int x, int y, int
rowSize /*baitu 152*/, int pixelCount, int originalRowSize)
        {
            for (int i = y + (pixelCount / 3); i < y + 2 * pixelCount / 3; i++)
            {
                for (int j = x + i * originalRowSize + (rowSize / 3); j < x + i *
originalRowSize + 2 * (rowSize / 3); j++)
                {
                    image[j] = 0xFF;
```

```
                }
            }
        }
        private static void DrawHorizontal(byte[] image, int x, int y, int
rowSize, int offset, int originalRowSize)
        {
            for (int i = offset - 8; i < offset; i++)
            {
                for (int j = x; j < rowSize; j++)
                {
                    image[i * originalRowSize + j] = 0xFF;
                }
            }
        }
        private static void DrawVertical(byte[] image, int y, int originalRowSize,
int pixelCount, int offset, byte dot)
        {
            for (int i = y; i < pixelCount; i++)
            {
                image[i * originalRowSize + offset] = dot;
            }
        }

        //prideti rekursijos gyli
        public static void DrawSquare(byte[] image, int x, int y, int rowSize
/*152*/, int pixelCount, int originalRowSize, int recNumber)
        {
            if(recNumber == 1)
            {
                //kairiausia linija
                DrawVertical(image, y, originalRowSize, y + pixelCount, 0,
0b11111111);
                //desiniausia linija
                DrawVertical(image, y, originalRowSize, y + pixelCount,
(pixelCount / 8) - 1, 0b11111111);
                //virsutine linijos
                DrawHorizontal(image, x, y, x + rowSize, pixelCount + y,
originalRowSize);
                //apatine linijos
                DrawHorizontal(image, x, y, x + rowSize, 8, originalRowSize);
                //pradinis kvadratas
                DrawVertical(image, y, originalRowSize, y + pixelCount, (rowSize /
3) + x, 0b11111111);
                DrawVertical(image, y, originalRowSize, y + pixelCount, 2 *
(rowSize / 3) + x, 0b11111111);
                DrawHorizontal(image, x, y, x + rowSize, (pixelCount / 3) + y + 8,
originalRowSize);
                DrawHorizontal(image, x, y, x + rowSize, 2 * (pixelCount / 3) + y,
originalRowSize);
                //uzpildome kvadrato viduri
                DrawFilledSquare(image, x, y, rowSize, pixelCount,
originalRowSize);
            }

            else if(recNumber == 2 || recNumber== 3)
            {
                double iterationCount = Math.Pow(3, recNumber);
                int random = (int)(iterationCount / 3);
                DrawHorizontal(image, x, y, x + rowSize, pixelCount + y,
originalRowSize);
                DrawHorizontal(image, x, y, x + rowSize, 8, originalRowSize);
                DrawVertical(image, y, originalRowSize, y + pixelCount, 0,
0b11111111);
                DrawVertical(image, y, originalRowSize, y + pixelCount,
(pixelCount / 8) - 1, 0b11111111);
```

```
                DrawFilledSquare(image, x, y, rowSize, pixelCount,
originalRowSize);
                if (iterationCount==27)
                {
                    for(int j = 0; j < 27; j++)
                    {
                        DrawHorizontal(image, x, y, x + rowSize, j * (pixelCount /
(int)iterationCount) + y + 8, originalRowSize);
                        DrawVertical(image, y, originalRowSize, y + pixelCount, j
* (rowSize / (int)iterationCount) + x, 0b11111111);
                    }
                    for (int i = 0; i < 9; i++)
                    {
                        DrawFilledSquare(image, x, y, rowSize / 9, pixelCount / 9,
originalRowSize);
                        DrawFilledSquare(image, x, y, rowSize, pixelCount,
originalRowSize);
                        DrawFilledSquare(image, x, i * pixelCount / 9, rowSize /
9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, pixelCount / 9, i * pixelCount /
9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 2 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 3 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 4 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 5 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 6 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 7 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, 8 * pixelCount / 9, i * pixelCount
/ 9, rowSize / 9, pixelCount / 9, originalRowSize);
                        DrawFilledSquare(image, x, y, rowSize / 3, pixelCount / 3,
originalRowSize);
                        //2
                        DrawFilledSquare(image, x, pixelCount / 3, rowSize / 3,
pixelCount / 3, originalRowSize);
                        //3
                        DrawFilledSquare(image, x, 2 * pixelCount / 3, rowSize /
3, pixelCount / 3, originalRowSize);
                        //4
                        DrawFilledSquare(image, pixelCount / 3, y, rowSize / 3,
pixelCount / 3, originalRowSize);
                        //5
                        DrawFilledSquare(image, pixelCount / 3, pixelCount / 3,
rowSize / 3, pixelCount / 3, originalRowSize);
                        //6
                        DrawFilledSquare(image, pixelCount / 3, 2 * pixelCount /
3, rowSize / 3, pixelCount / 3, originalRowSize);
                        //7
                        DrawFilledSquare(image, 2 * pixelCount / 3, y, rowSize /
3, pixelCount / 3, originalRowSize);
                        //8
                        DrawFilledSquare(image, 2 * pixelCount / 3, pixelCount /
3, rowSize / 3, pixelCount / 3, originalRowSize);
                        //9
                        DrawFilledSquare(image, 2 * pixelCount / 3, 2 * pixelCount
/ 3, rowSize / 3, pixelCount / 3, originalRowSize);
                    }
                }
                if (iterationCount==9)
                {
                    for (int i = 0; i < iterationCount; i++)
```

```
                    {
                        DrawHorizontal(image, x, y, x + rowSize, i * (pixelCount /
(int)iterationCount) + y + 8, originalRowSize);
                        DrawVertical(image, y, originalRowSize, y + pixelCount, i
* (rowSize / (int)iterationCount) + x, 0b11111111);
                        //1
                        DrawFilledSquare(image, x, y, rowSize / 3, pixelCount / 3,
originalRowSize);
                        //2
                        DrawFilledSquare(image, x, pixelCount / 3, rowSize / 3,
pixelCount / 3, originalRowSize);
                        //3
                        DrawFilledSquare(image, x, 2 * pixelCount / 3, rowSize /
3, pixelCount / 3, originalRowSize);
                        //4
                        DrawFilledSquare(image, pixelCount / 3, y, rowSize / 3,
pixelCount / 3, originalRowSize);
                        //5
                        DrawFilledSquare(image, pixelCount / 3, pixelCount / 3,
rowSize / 3, pixelCount / 3, originalRowSize);
                        //6
                        DrawFilledSquare(image, pixelCount / 3, 2 * pixelCount /
3, rowSize / 3, pixelCount / 3, originalRowSize);
                        //7
                        DrawFilledSquare(image, 2 * pixelCount / 3, y, rowSize /
3, pixelCount / 3, originalRowSize);
                        //8
                        DrawFilledSquare(image, 2 * pixelCount / 3, pixelCount /
3, rowSize / 3, pixelCount / 3, originalRowSize);
                        //9
                        DrawFilledSquare(image, 2 * pixelCount / 3, 2 * pixelCount
/ 3, rowSize / 3, pixelCount / 3, originalRowSize);
                    }
                }

            }
            else
            {
                Console.WriteLine("Not supported!");
            }
        }

    }
}
```

InOutUtils.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using L1_bmp;

namespace L1_bmp
{
    public static class InOutUtils
    {
        public static void PrintImage(string FileName, byte[] structure, byte[]
image)
        {
```

```
            using (FileStream f = new FileStream(FileName, FileMode.Create,
FileAccess.Write))
            {
                f.Write(structure, 0, structure.Length);
                f.Write(image, 0, image.Length);
            }
        }
        public static byte[] MakeStructure(uint pixelCount, uint rowSize)
        {
            // Header
            ushort signature = 19778;
            uint fileSize = pixelCount * rowSize + 62;
            uint reserved = 0;
            uint dataOffset = 62;

            // InfoHeader
            uint size = 40;
            uint width = pixelCount;
            uint height = pixelCount;
            ushort planes = 1;
            ushort bitsPerPixel = 1;
            uint compression = 0;
            uint imageSize = 0;
            uint xPixelsPerM = 0;
            uint yPixelsPerM = 0;
            uint colorsUsed = 0;
            uint importantColors = 0;

            // ColorTable
            uint colorTable1 = 16777215;
            uint colorTable2 = 0x0;

            byte[] structure = new byte[62];
            int position = 0;
            position = AppendToByteArray(structure,
BitConverter.GetBytes(signature), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(fileSize), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(reserved), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(dataOffset), position);
            position = AppendToByteArray(structure, BitConverter.GetBytes(size),
position);
            position = AppendToByteArray(structure, BitConverter.GetBytes(width),
position);
            position = AppendToByteArray(structure, BitConverter.GetBytes(height),
position);
            position = AppendToByteArray(structure, BitConverter.GetBytes(planes),
position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(bitsPerPixel), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(compression), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(imageSize), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(xPixelsPerM), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(yPixelsPerM), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(colorsUsed), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(importantColors), position);
            position = AppendToByteArray(structure,
BitConverter.GetBytes(colorTable1), position);
```

```
            AppendToByteArray(structure, BitConverter.GetBytes(colorTable2),
position);
            return structure;
        }

        private static int AppendToByteArray(byte[] main, byte[] add, int
position)
        {
            for (int i = position, j = 0; i < position + add.Length; i++, j++)
            {
                main[i] = add[j];
            }
            return position + add.Length;
        }
    }
}
```

1.3. Šaltiniai

https://moodle.ktu.edu/course/view.php?id=2470
https://en.wikipedia.org/wiki/BMP_file_format

2. 2 LD laboratorinis darbas

3. 3 LD laboratorinis darbas

4. 4 LD laboratorinis darbas