

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Algoritmų sudarymas ir analizė (P170B400)
Laboratorinių darbų ataskaita

Atliko:

IFF-1/6 gr. studentas

Lukas Kuzmickas

2023 m. kovo 23 d.

Priėmė:

Doc. Pilkauskas Vytautas

TURINYS

1.	1 LD laboratorinis darbas	3
1.1.	Pradinė užduotis	3
1.2.	Užduoties sprendimas ir rezultatai	4
1.2.1	Pirmosios rekurentinės lygties sprendimas	4
1.2.2	Antrosios rekurentinės lygties sprendimas	10
1.2.3	Trečiosios rekurentinės lygties sprendimas	16
1.2.4	BMP formato užduoties sprendimas	22
1.3.	Šaltiniai	32
2.	2 LD laboratorinis darbas	33
2.1.	Pradinė užduotis	33
2.2.	Užduočių sprendimas ir rezultatai	35
2.2.1	Pirmosios užduoties dalies sprendimas	35
2.2.2	Antrosios užduoties dalies sprendimas	44
2.3.	Šaltiniai	52
3.	3 LD laboratorinis darbas	52
4.	4 LD laboratorinis darbas	52

1. 1 LD laboratorinis darbas

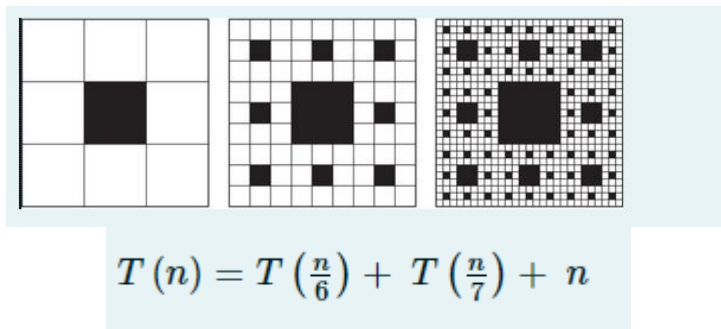
1.1. Pradinė užduotis

Kiekvienai rekurentinei lygčiai (gautai atlikus užduoties pasirinkimo testą):

- Realizuoti metodą, kuris atitiktų pateiktos rekurentinės lygties sudėtingumą, t. y. programinio kodo rekursinių iškviatimų ir kiekvieno iškviatimo metu atliekamų veiksmų priklausomybę nuo duomenų. Metodas per parametrus turi priimti masyvą, kurio duomenų kiekis yra rekurentinės lygties kintamasis n (arba masyvą ir indeksų režius, kurie atitinkamai nurodo masyvo nagrinėjamų elementų indeksus atitinkamame iškviatime) (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)



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

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

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	Kaina	Kiekis
<pre> 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; } } </pre>		
	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.

$$\begin{aligned}
T(n) &= c_1 + c_2 + T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + c_3 + c_4n + c_5n + c_6n + c_7n^2 + c_8n^2 + c_9n^2 + c_{10}n^3 + c_{11}n^3 \\
&\quad + c_{12}n^3 + c_{13}n^4 + c_{14}n^4 + c_{15}n^4 + c_{16} \\
&= T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + c_1 + c_2 + c_{16} + c_3 + n(c_5 + c_6) + n^2(c_7 + c_8 + c_9) \\
&\quad + n^3(c_{10} + c_{11} + c_{12}) + n^4(c_{13} + c_{14} + c_{15});
\end{aligned}$$

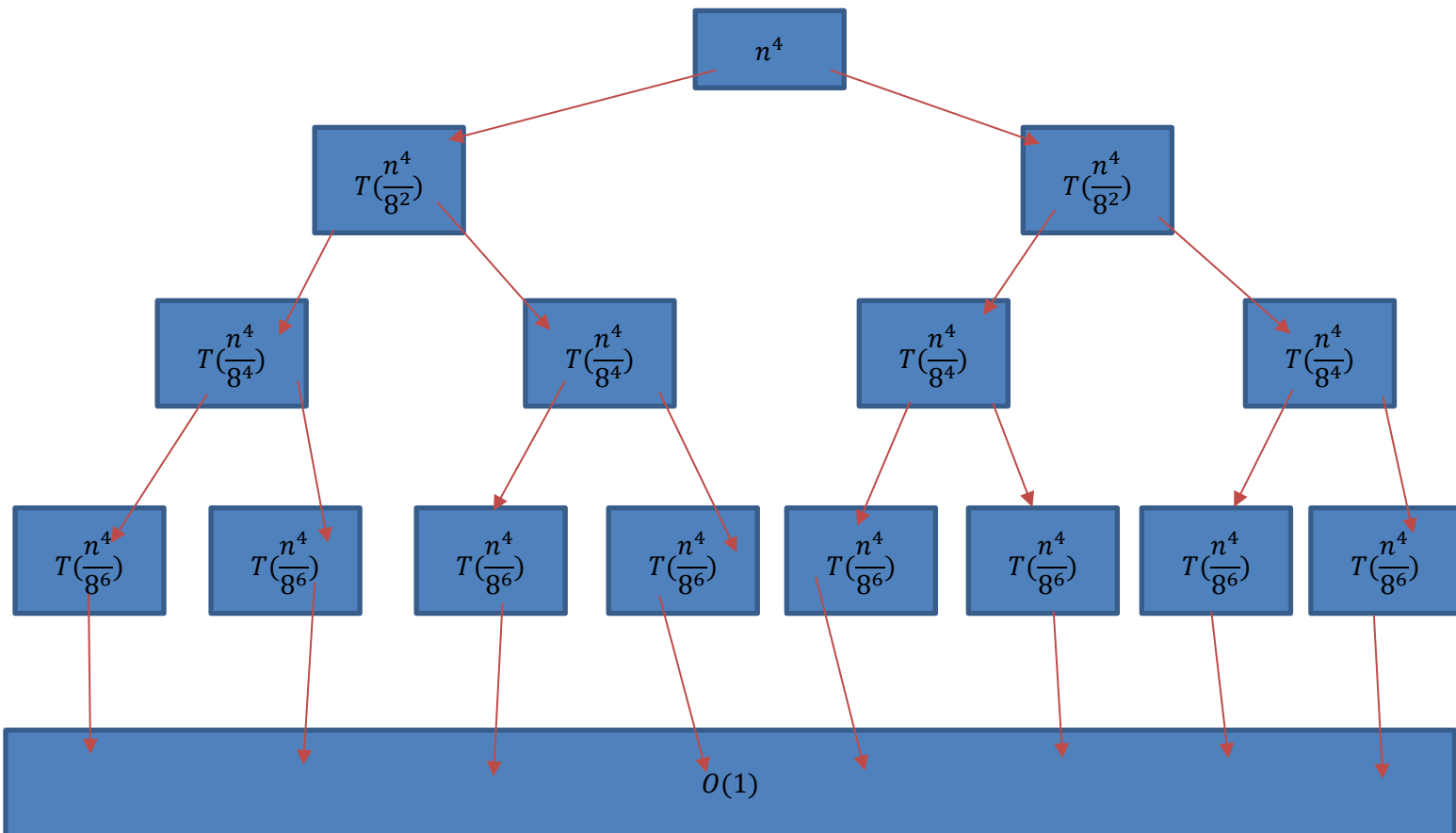
Atmetus konstantas, suprastinus šią išraišką, gauname panašią rekurentinę lygtį į mūsų:

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

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 dėsnumus, sudarome lygtį jų sumavimui:

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

Nagrinėjame pati blogiausią atvejį.

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

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

Toliau nagrinėjame geriausią atvejį.

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

Sprendžiame uždavinį:

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

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

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

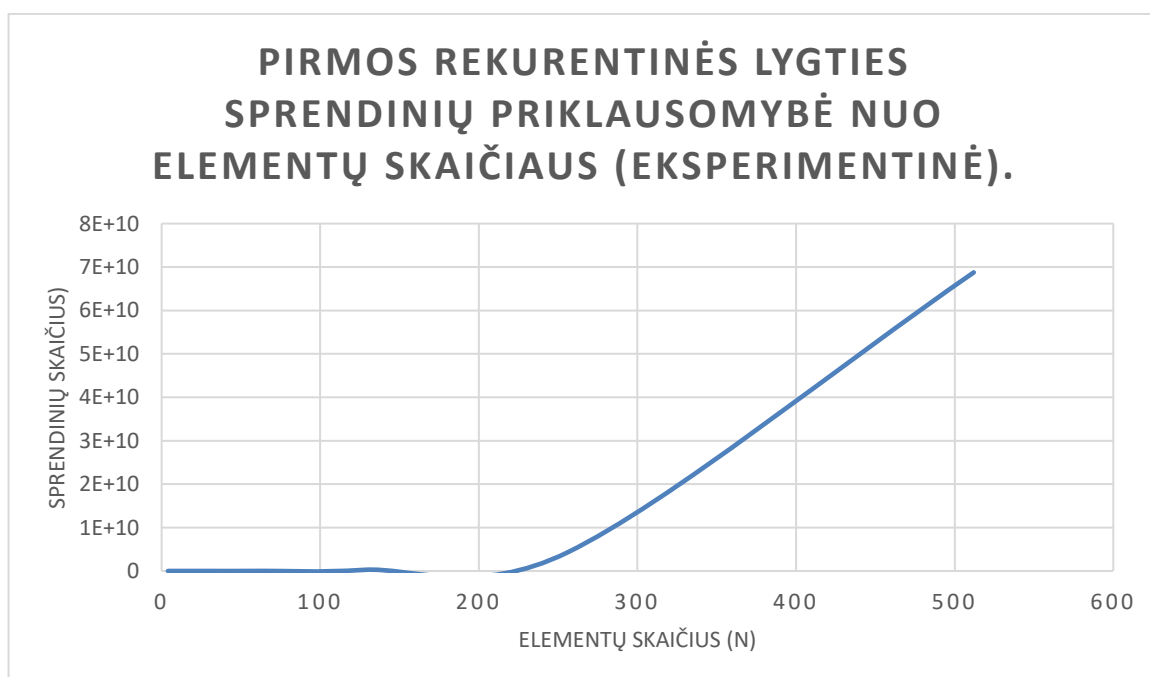
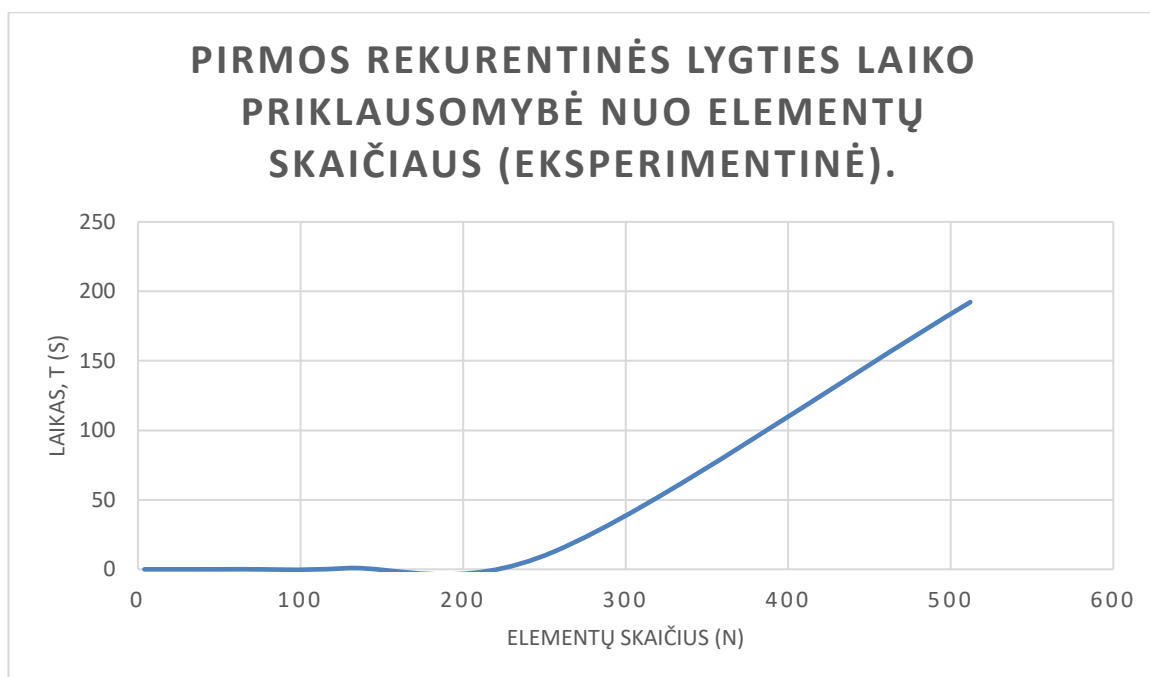
$$T(n) > \frac{1023}{1024} * \frac{32}{31} * n^4 = \frac{33}{32} n^4;$$

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

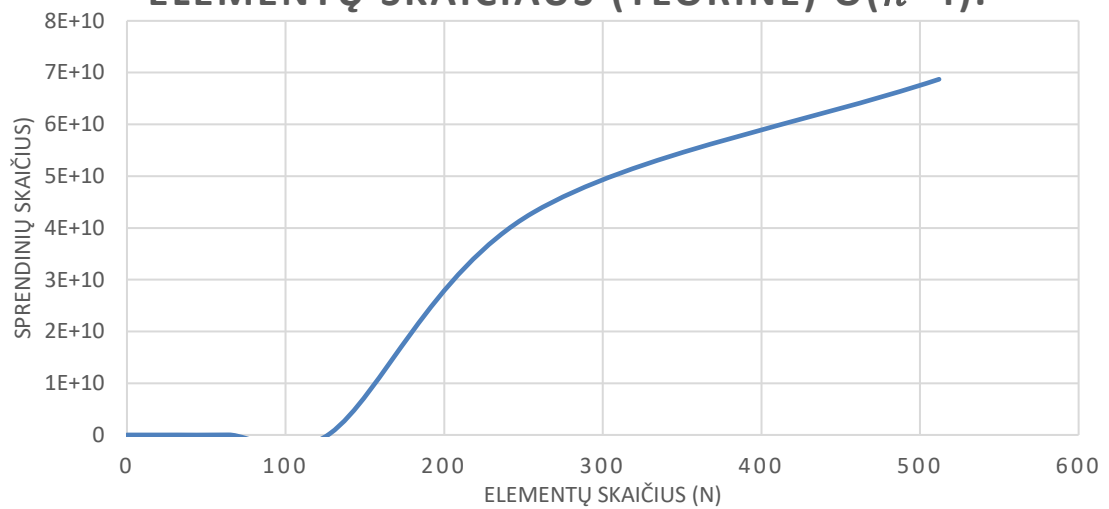
Kadangi galioja $T(n) = O(n^4)$ ir $T(n) = \lambda(n^4)$, tai galioja ir $T(n) = \theta(n^4)$, nes

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

Našumo testai:



**PIRMOS REKURENTINĖS LYGTIES
SPRENDINIŲ PRIKLAUSOMYBĖ NUO
ELEMENTŲ SKAIČIAUS (TEORINĖ) $O(n^4)$.**



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

5 pav. Antrojo metodo kodo analizė

$$\begin{aligned}
 T(n) &= c_1 + c_2 + T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + c_3 + c_4n + c_5n + c_6n + c_7 \\
 &= T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + c_1 + c_2 + c_3 + c_7 + n(c_5 + c_6);
 \end{aligned}$$

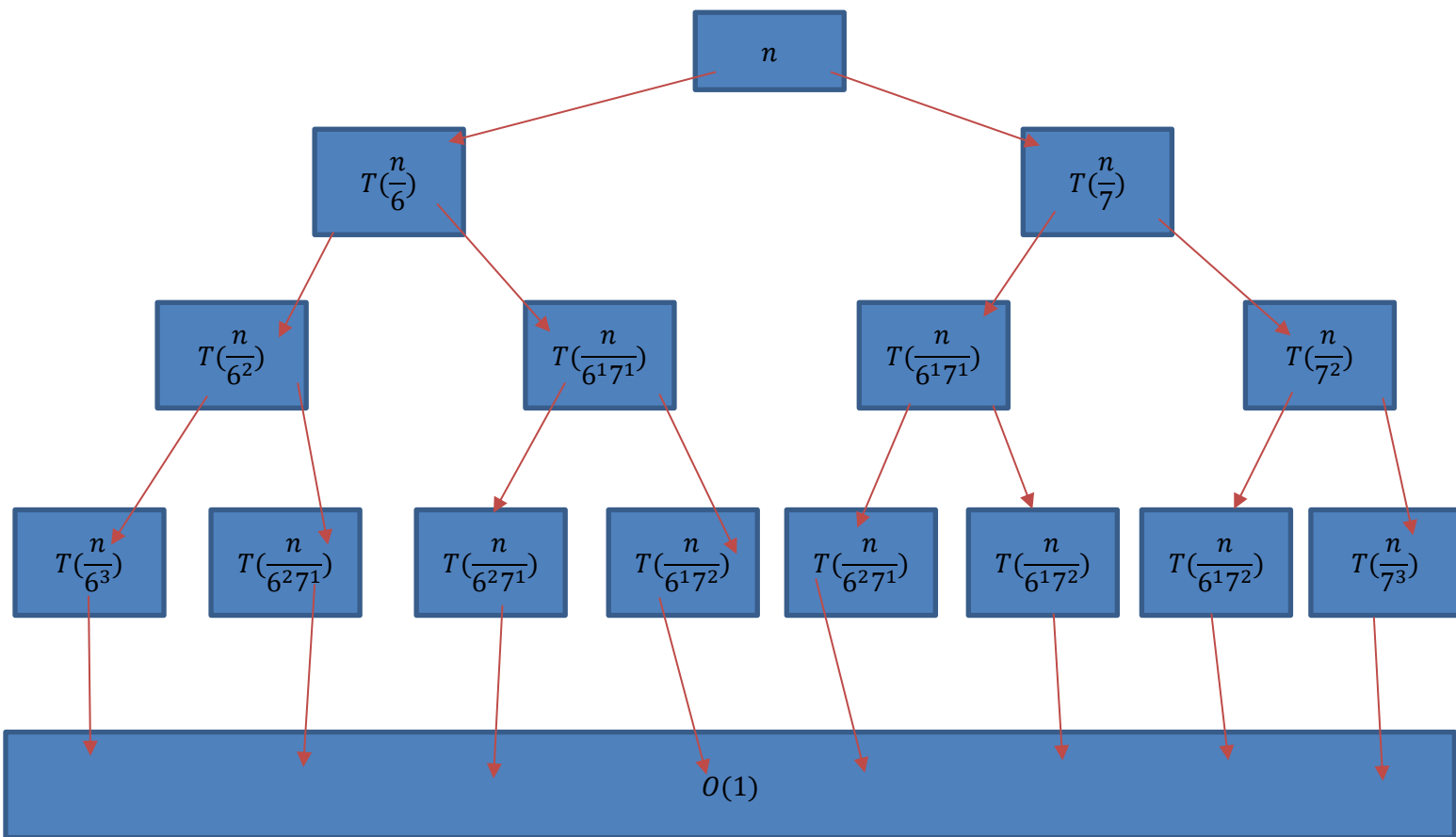
Atmetus konstantas, gauname panašią rekurentinę lygtį į mūsų:

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

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 dėsnumus, sudarome lygtį jų sumavimui (Niutono binomas).

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

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

Nagrinėjame pati blogiausią atvejį:

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

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

Toliau nagrinėjame geriausią atvejį.

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

Sprendžiame uždavinį:

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

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

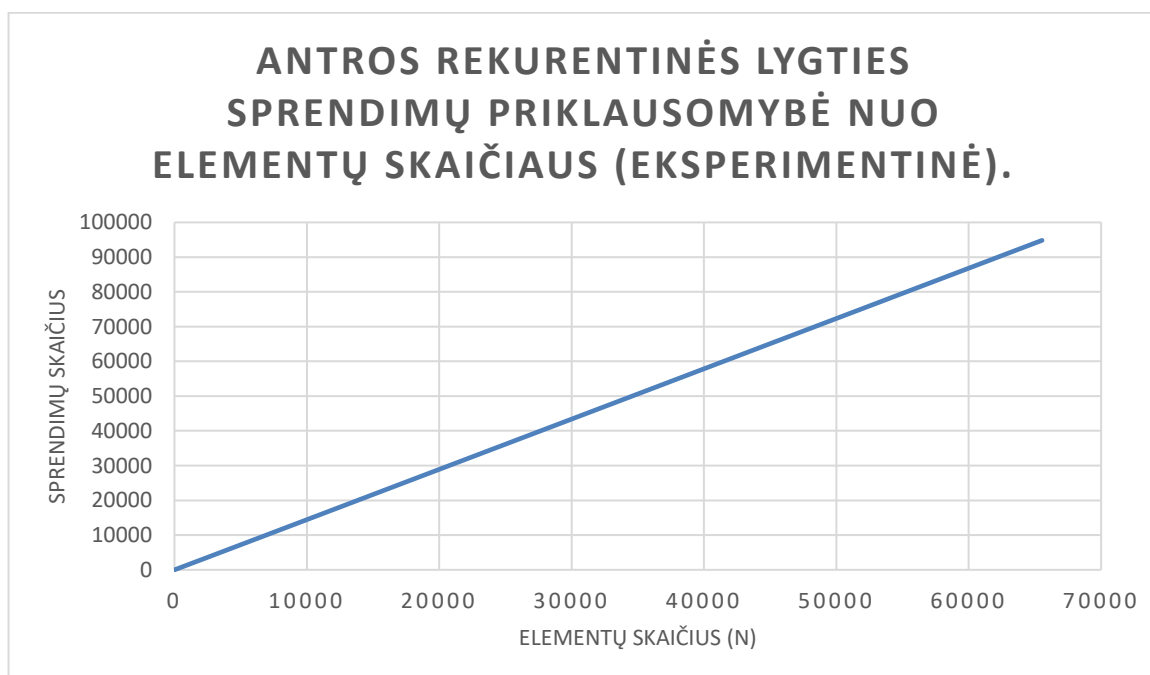
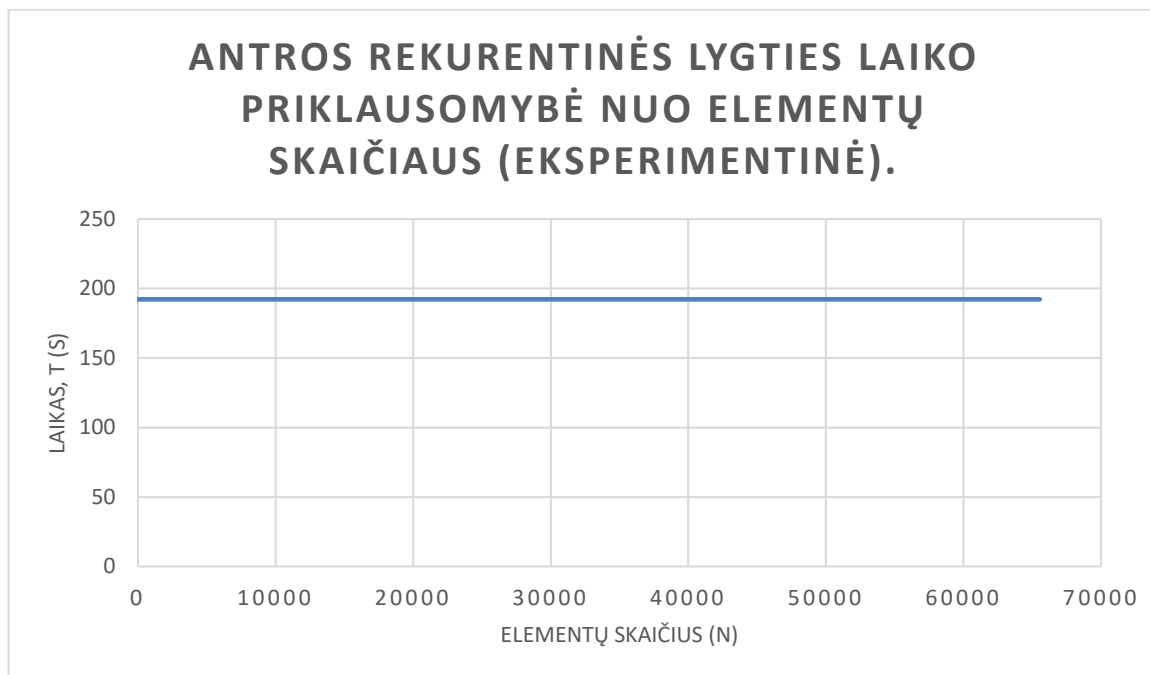
$$T(n) = \frac{42}{29} * \frac{29}{42} * n = n;$$

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

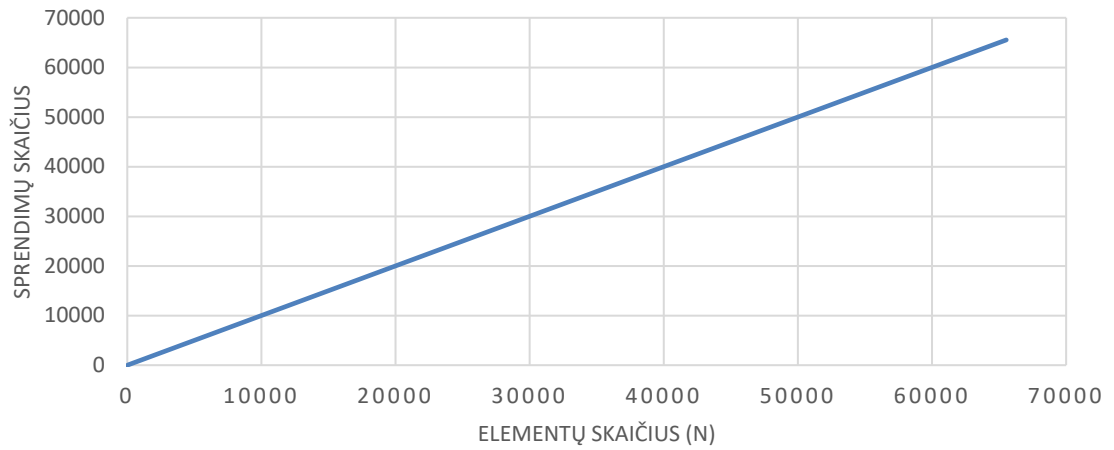
Kadangi galioja $T(n) = O(n)$ ir $T(n) = \lambda(n)$, tai galioja ir $T(n) = \theta(n)$, nes

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

Našumo testai:



**ANTROS REKURENTINĖS LYGTIES
SPRENDIMŲ PRIKLAUSOMYBĖ NUO
ELEMENTŲ SKAIČIAUS (TEORINĖ) $O(N)$.**



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
<pre> static ulong T3(long[] arr, long startI, long endI) { ulong result = 0; if (endI >= 1) { result += T3(arr, 0, endI - 9); result += T3(arr, 0, endI - 1); result += 1; } return result; } </pre>		
	c1	1
	c2	1
	T(n-9)	1
	T(n-1)	1
	c5	1
	c7	1

7 pav. Trečiojo metodo kodo analizė.

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

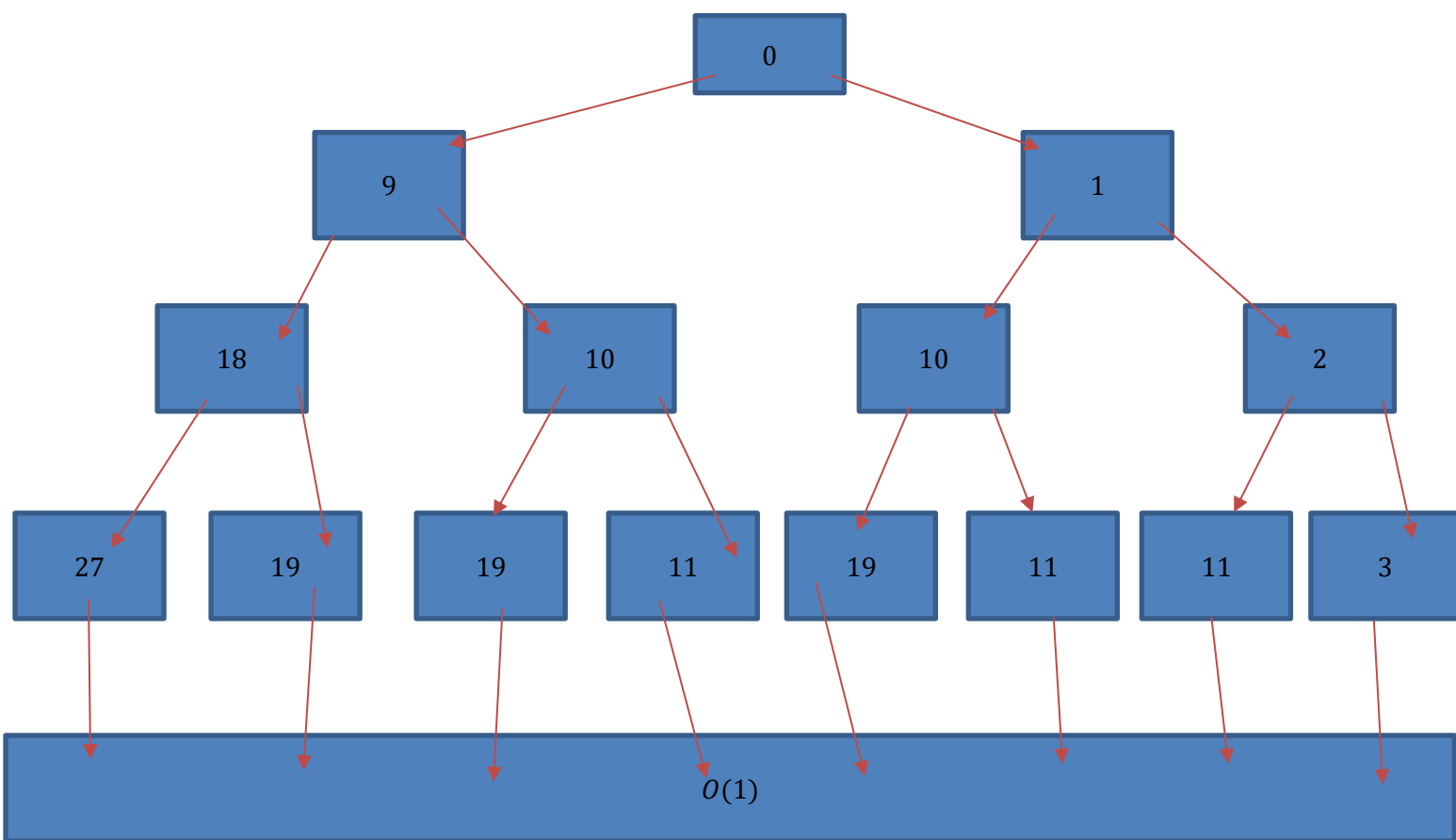
Atmetus konstantas, gauname panašią rekurentinę lygtį į mūsų:

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

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 sistemą, 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.

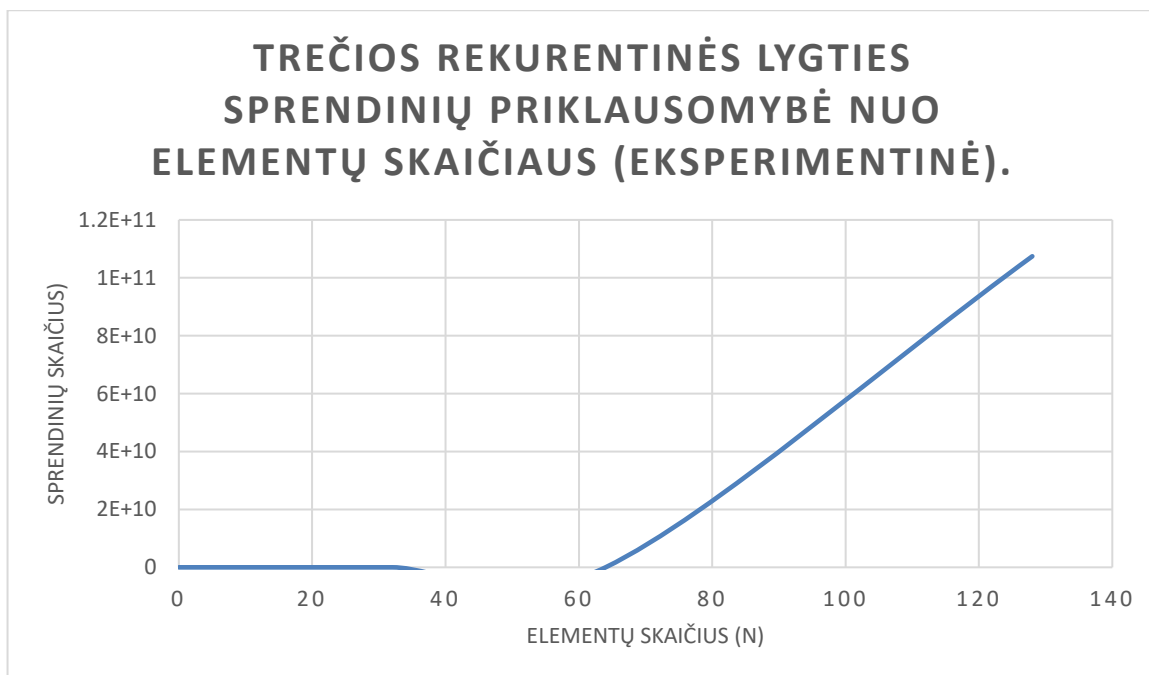
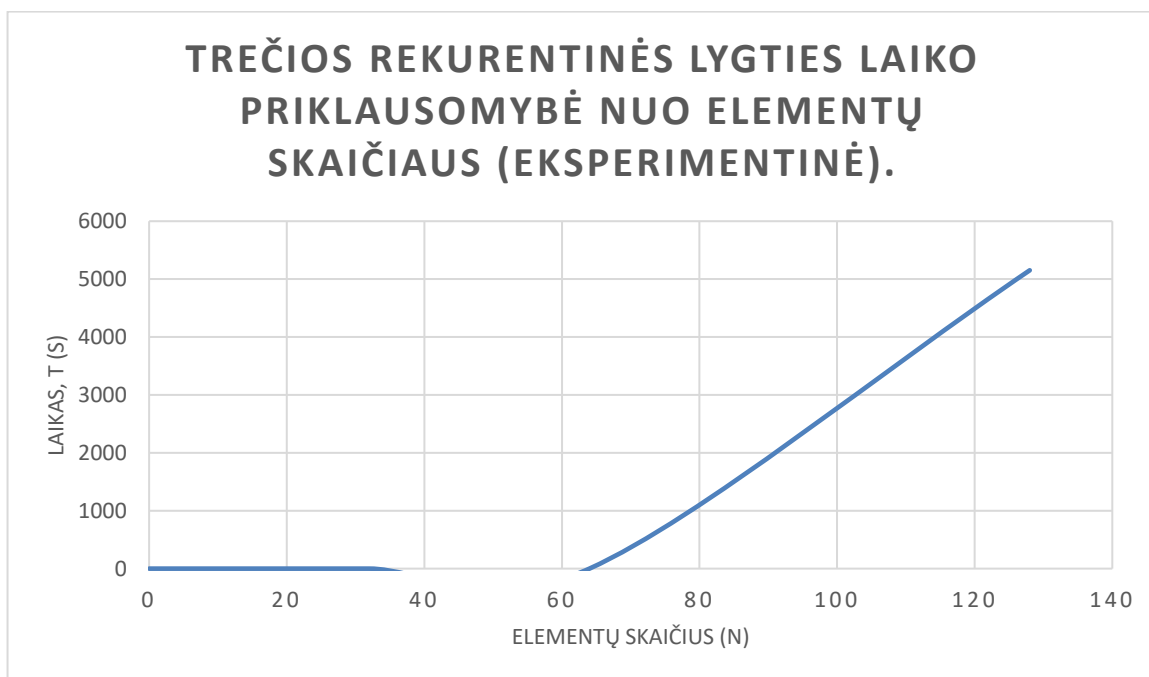
$$\sum_{i=0}^h 2^i - 10 \sum_{i=0}^h i 2^{i-1} = (2^{h+1} - 1) - 10(h 2^h - 2^h + 1);$$

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

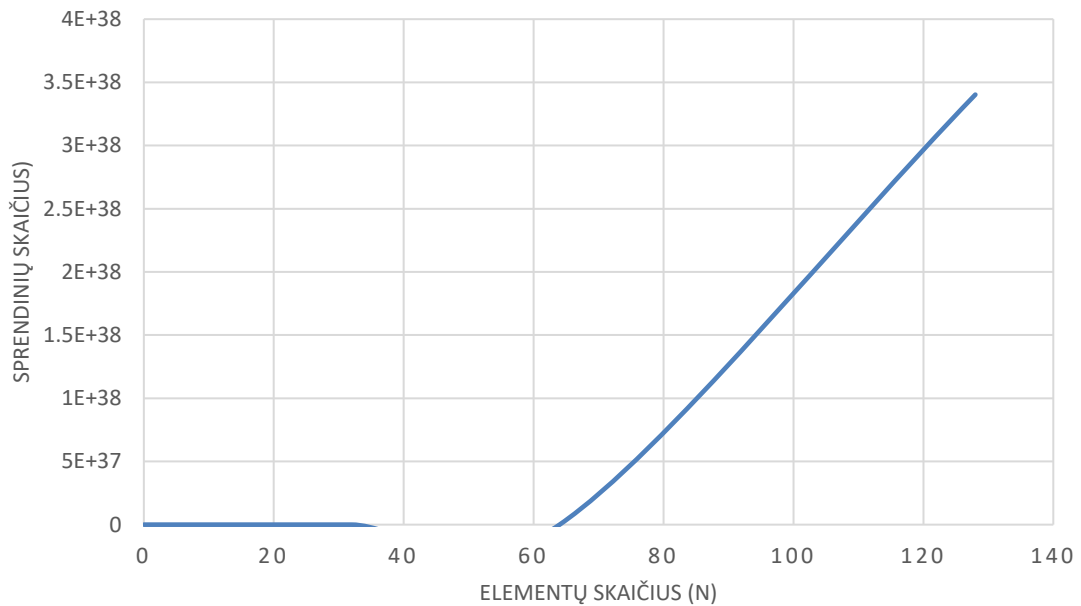
$$T(n) = \lambda \left(2^{\frac{n}{9}} \right) \approx O(2^n);$$

Negavome asimptotiškai tikslaus įvertinimo.

Našumo testai:



TREČIOS REKURENTINĖS LYGTIES SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ) $O(2^N)$.



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
```

```

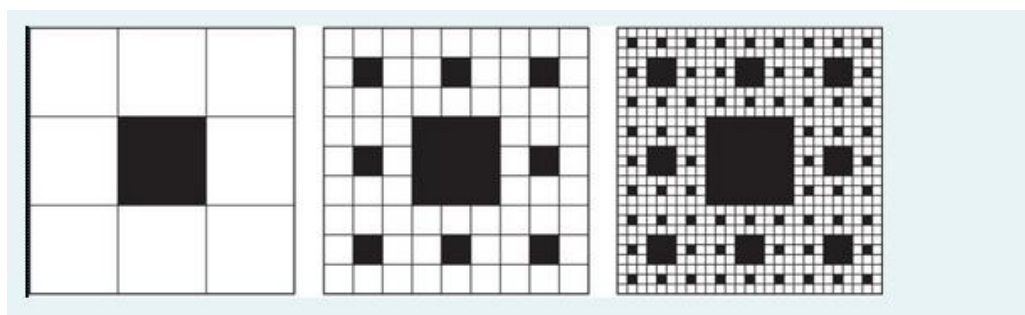
        if (endI >= 1) //c2 | 1
        {
            result += thirdRecurrent(arr, 0, endI - 9); //T(n-
9) | 1
            result += thirdRecurrent(arr, 0, endI - 1); //T(n-
1) | 1
            result += 1; //c5 | 1
        }
        return result; //c6 | 1
    }
    static void PrintToFile(string fileName, 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):

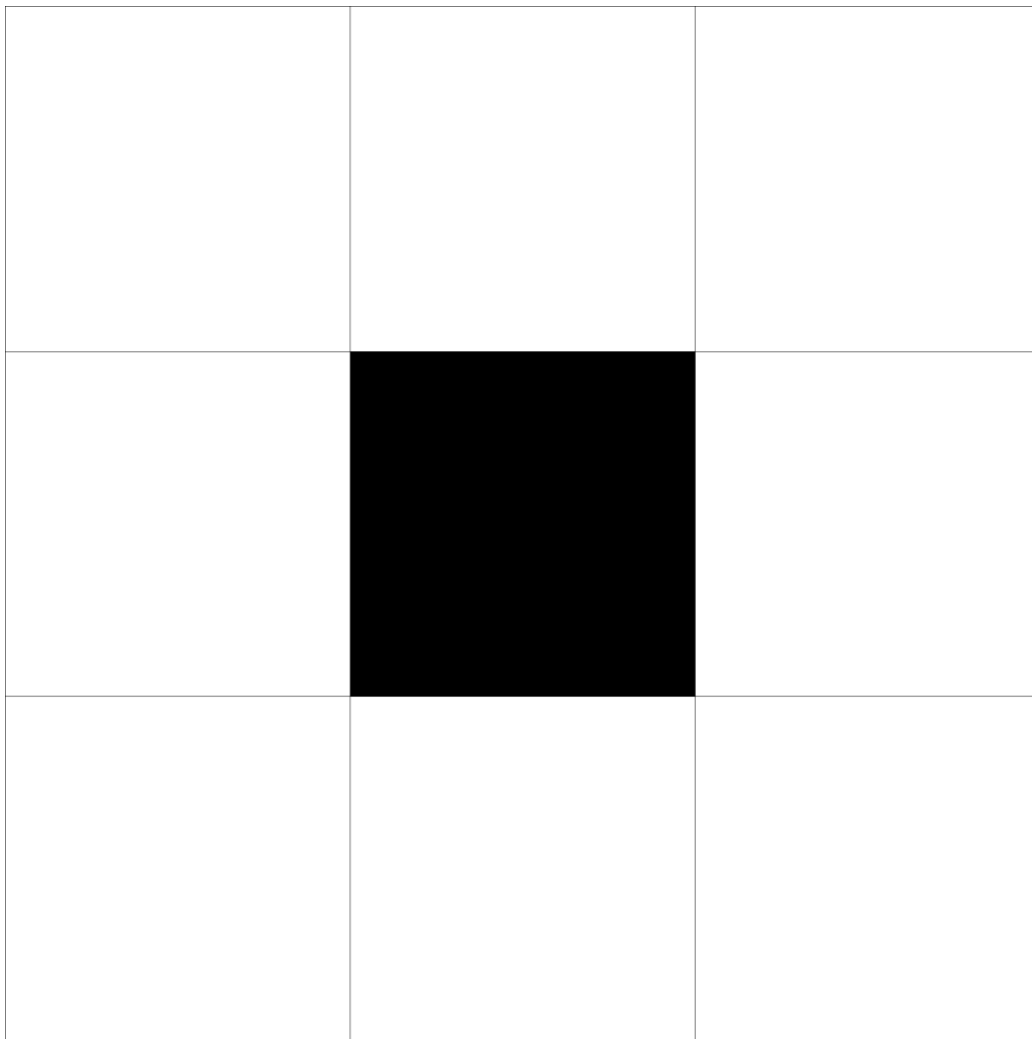
- Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas. (3 balai)
- Eksperimentiškai nustatykite darbo laiko ir veiksmų skaičiaus priklausomybę nuo generuojamo paveikslėlio dydžio (taškų skaičiaus). Gautus rezultatus atvaizduokite grafikais. Grafiką turi sudaryti nemažiau kaip 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)



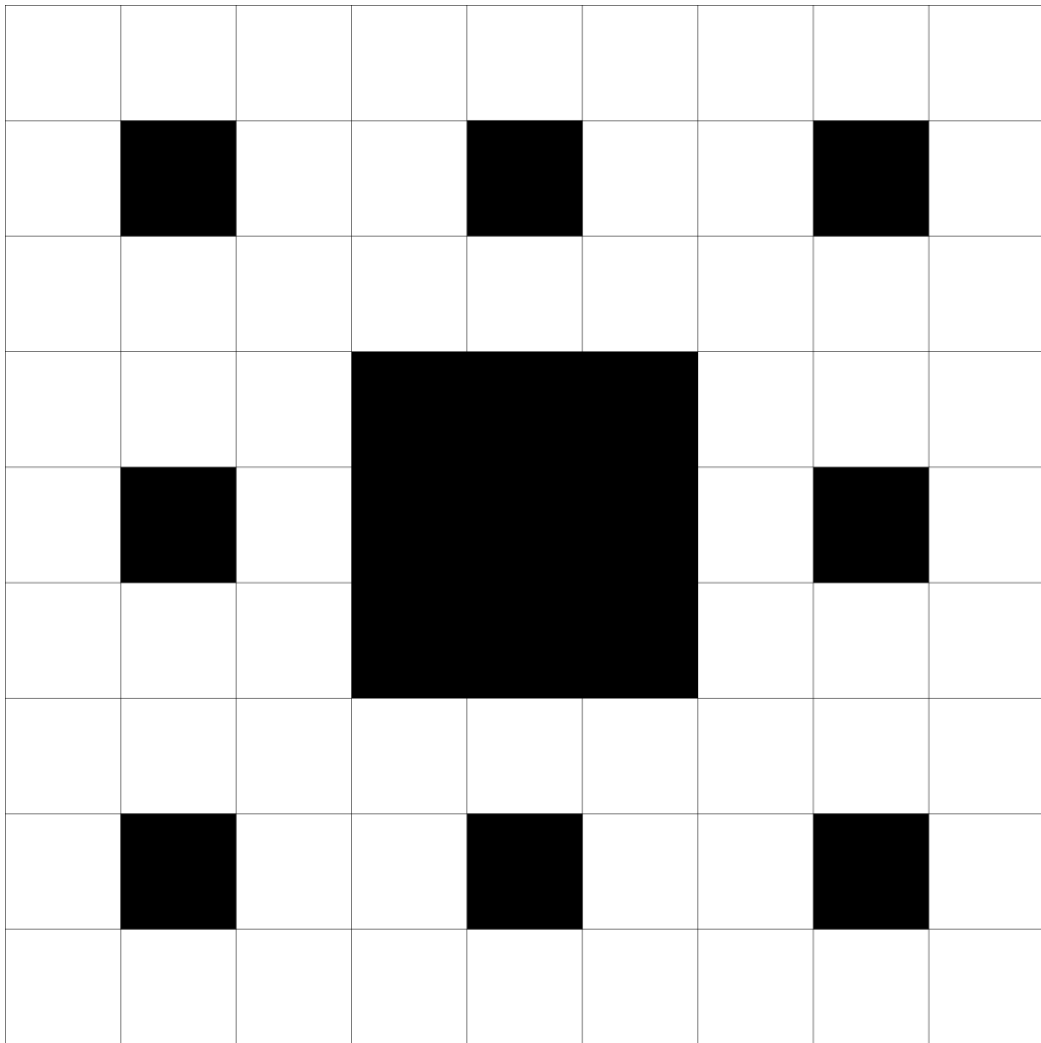
Pav. BMP formato užduotis.

Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas:

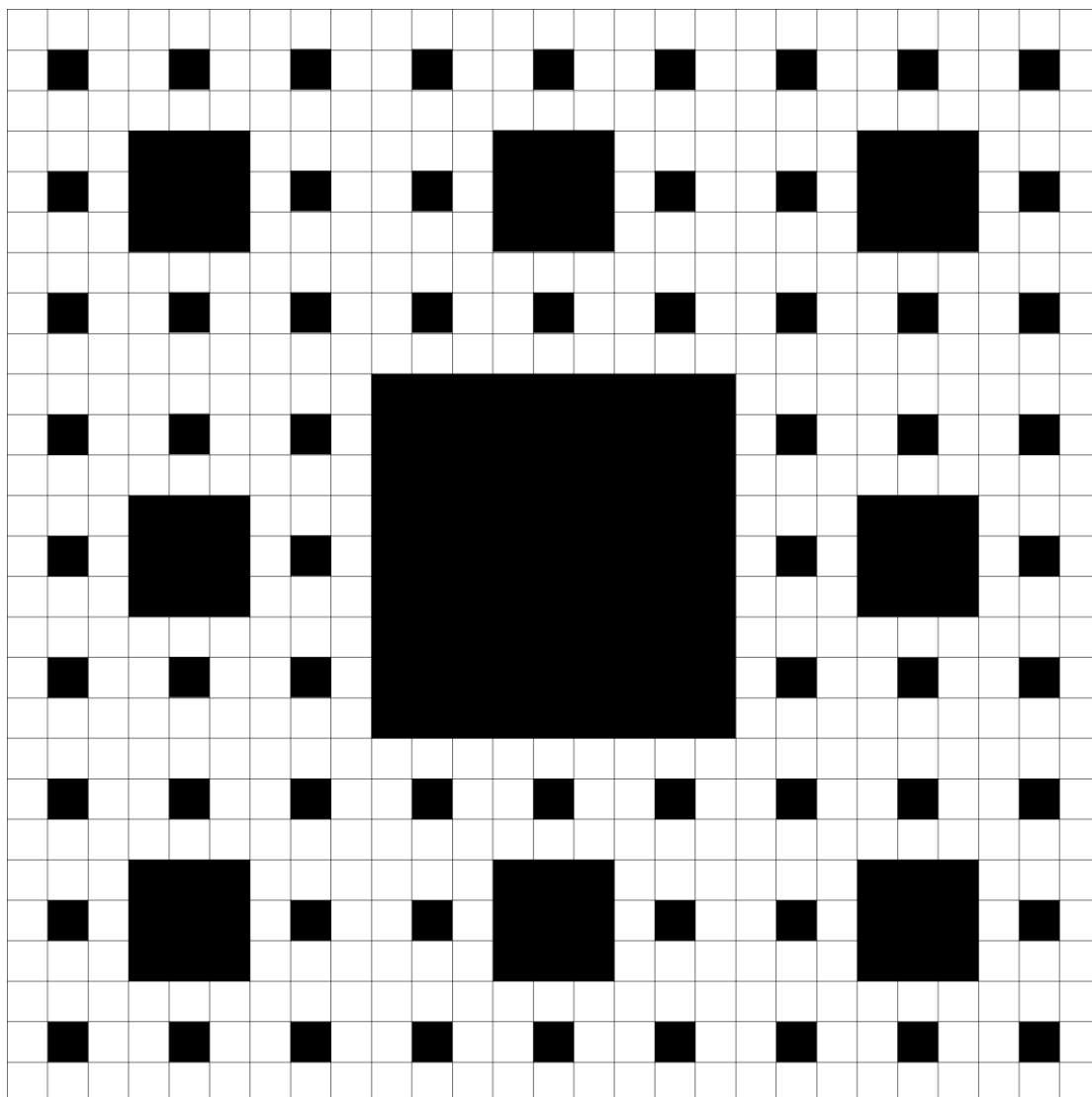
Pirma nuotrauka (image1.bmp)



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

2.1. Pradinė užduotis

1 užduoties dalis (4 balai):

Pateiktiems programinio kodo metodams „*methodToAnalysis(...)*“ (gautiems atlikus užduoties pasirinkimo testą):

- atlikite programinio kodo analizę, bei sudarykite rekurentinę lygtį. Jei metodas neturi vidinių rekursinių kreipinių, apskaičiuokite pateikto metodo asimptotinį sudėtingumą. Jei metodo sudėtingumas priklauso nuo duomenų pateikiamų per parametrus – apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“ (2 balai).
- Metodams, kurie turi rekurentinių kreipinių išspręskite rekurentinę lygtį apskaičiuodami jos asimptotinį sudėtingumą (1 balas).
- Atlikti eksperimentinį tyrimą (našumo testus: vykdymo laiką ir veiksmų skaičių) ir patikrinkite ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus. Jei pateikto metodo asimptotinis sudėtingumas priklauso nuo duomenų, atitinkamai atliekant analizę reikia parinkti tokias testavimo duomenų imtis, kad rezultatai atspindėtų įvertinimus iš viršaus ir iš apačios (1 balas).

```
public static long methodToAnalysis (int[] arr)
{
    long n = arr.Length;
    long k = n;
    if (arr[0] > 0) {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                k -= 2;
            }
        }
    }
    return k;
}
```

```

public static long methodToAnalysis (int n, int[] arr)
{
    long k = 0;
    for (int i = 0; i < n; i++)
    {
        k += arr[i] + FF4(i, arr);
    }

    return k;
}

public static long FF4(int n, int [] arr)
{
    if (n > 0 && arr.Length > n && arr[n] < 0)
    {
        return FF4(n / 2, arr) + FF4(n / 3, arr);
    }
    return n;
}

```

9 pav. Antros užduties metodas.

2 užduties dalis (6 balai):

- Pateikite rekursinį uždavinio sprendimo algoritmą (rekursinis sąryšis su paaiškinimais), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (rekursinis sprendimas netaikant dinaminio programavimo).
- Pritaikykite dinaminio programavimo metodologiją pateiktam uždaviniui (pateikti paaiškinimą), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (taikant dinaminį programavimą).
- Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.

Ant žaidimo lentos ($1 \times n$) langelių surašyti atsitiktiniai teigiami skaičiai (taškai). Pradėjęs pirmame langelyje, žaidėjas vieno ėjimo metu gali pasirinkti – pereiti į kitą langelį ($1 \rightarrow 2$) ir gauti tiek taškų, kiek yra tame langelyje, ar persokti du langelius ($1 \rightarrow 4$) ir gauti du kartus daugiau taškų, nei yra užrašyta tame langelyje. Kokia yra mažiausia taškų suma, kurią žaidėjas gali surinkti paskutiniame langelyje?

10 pav. Antra uždavinio dalis.

2.2. Užduočių sprendimas ir rezultatai

2.2.1 Pirmosios užduoties dalies sprendimas

Atliekame pirmojo metodo methodToAnalysis kodo analizę.

<pre> public static long methodToAnalysis (int[] arr) { long n = arr.Length; long k = n; if (arr[0] > 0) { for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { k -= 2; } } } return k; } </pre>	Svoris	Kiekis
	c1	1
	c2	1
	c3	1
	c4;c5;c6	$x*1;x*(n+1);x*n$
	c7;c8;c9	$x*n;x*(n^2+1);x*(n^2)$
	c10	$x*n^2$
	c11	1

11 pav. Pirmojo metodo kodo analizė.

c_i – konstantos, kurios yra $O(1)$;

Apskaičiuojame šiuos svorius ir įvertiname asimptotinį sudėtingumą.

Įvedame kintamąjį elementą x , jis parodo, ar mūsų metode yra įvykdomas if sąlygą, jeigu $x = 0$, tai

$$T(n) = c_1 + c_2 + c_{11} = O(1);$$

Jeigu mūsų kintamasis x įvykdomas, t.y. $x = 1$, tai

$$T(n) = c_1 + c_2 + c_3 + (c_4 * x * 1) + c_5 * x(n + 1) + c_6 * x(n) + c_7 * x(n) + c_8 * x(n^2 + 1) + c_9 * x(n^2) + c_{10} * x(n^2);$$

Suprastiname visą funkciją ir gauname:

$$T(n) = n^2;$$

Turime du sprendimus – Viršutiniai ir apatiniai rėžiai, kai $x = 0$,

$$T(n) = O(1);$$

Ir, kai $x = 1$,

$$T(n) = O(n^2);$$

Atliekame antrojo metodo methodToAnalysis kodo analizę.

<pre> public static long methodToAnalysis (int n, int[] arr) { long k = 0; for (int i = 0; i < n; i++) { k += arr[i] + FF4(i, arr); } return k; } </pre>	Svoris	Kiekis
	c1	1
	c2,c3,c4	1;n+1;n
	$\sum_{i=0}^n FF4(i)$	1
	c5	1

12 pav. Antrojo metodo kodo analizė.

Išsivedame kintamąjį x , kuris parodo ar mūsų if sąlyga buvo įvykdyta FF4 metode.

Po analizės susumuojame visus šiuos svorius ir apskaičiuojame rekurentinę lygtį.

$$T(n) = c_1 + c_2 + c_3(n + 1) + c_4(n) + \sum_i^n FF4(i) + c_5;$$

$$FF4(n) = c_6 + c_7 + c_8 + \left(FF4\left(\frac{n}{2}\right) + FF4\left(\frac{n}{3}\right)\right) * x + c_9, \text{ kai mūsų } x = 1, \text{ t.y. praeinama if sąlyga.}$$

$$FF4(n) = c_6 + c_7 + c_8 + c_9 = O(1), \text{ kai mūsų } x = 0, \text{ t.y. nepraeinama if sąlyga.}$$

Suprastiname konstantas ir įstatome FF4 lygtį į $T(n)$, gauname rekurentinių sąryšių nelygybę.

Ši sąlyga galioja, kai mūsų $x = 1$.

$$T(n) = \left(FF4\left(\frac{n}{2}\right) + FF4\left(\frac{n}{3}\right)\right) + \sum_i^n FF4(i) + n;$$

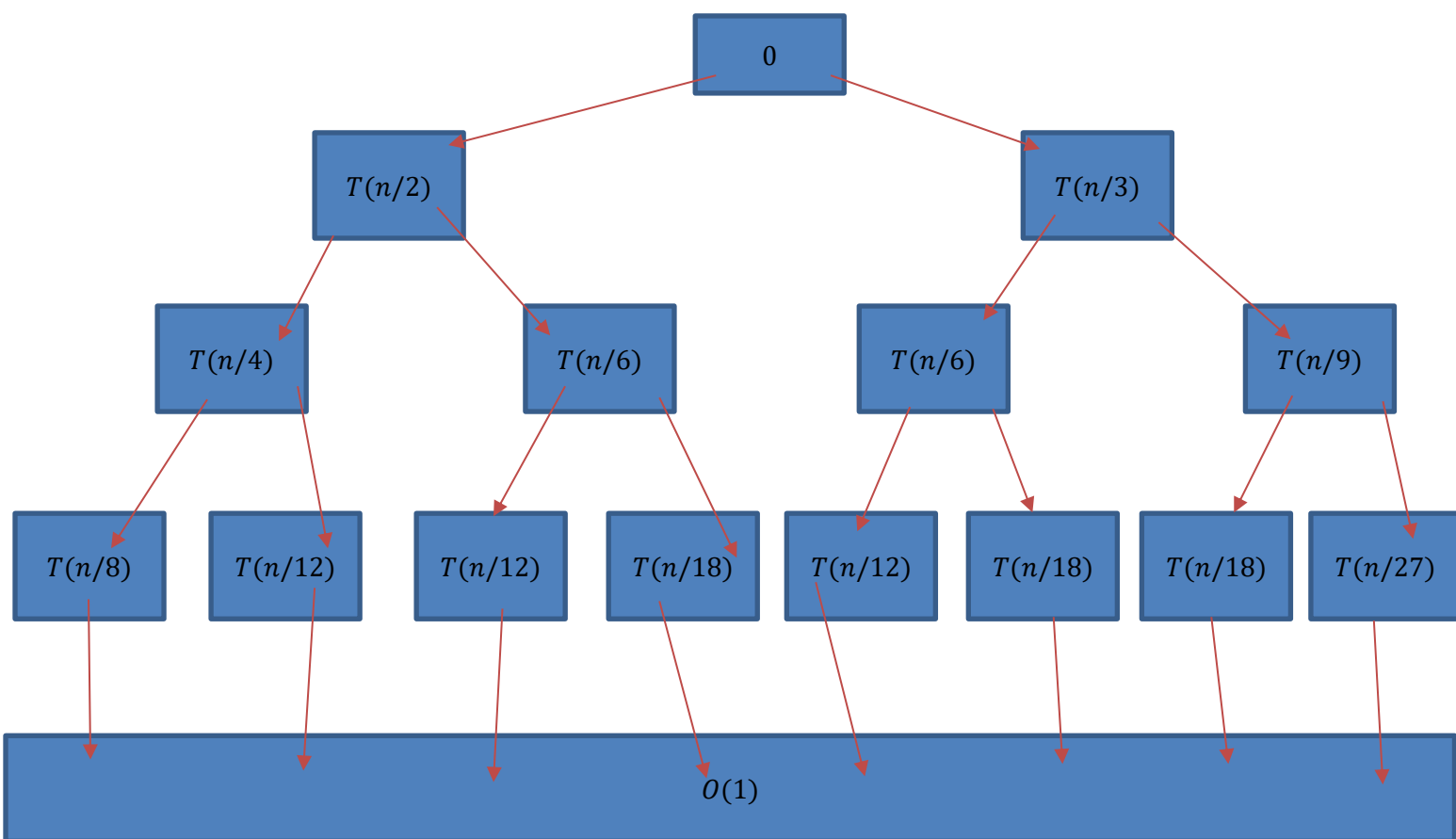
Ši sąlyga galioja, kai mūsų $x = 0$.

$$T(n) = n;$$

Sprendžiame $FF4(n) = FF4\left(\frac{n}{2}\right) + FF4\left(\frac{n}{3}\right)$ rekurentinę lygtį.

1 sprendimo būdas medžio metodas:

Ši lygtis turi rekurentinių kreipinių, todėl išsprendžiame ją. Pradžioje sudarome rekurentinės lygties sprendimo medį šiai užduočiai.



Rekurentinis sprendimo medis

Toliau suskaičiuojame, kiekvieno iteracijos lygio svorių sumą.

0 lygis: 0

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

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

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

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

$$\left(\frac{1}{2} + \frac{1}{3}\right)^i = \left(\frac{5}{6}\right)^i;$$

Šis medis nėra simetrinis, todėl medžio aukštis h yra $\lceil \log_3 n \rceil \leq h \leq \lceil \log_2 n \rceil$;

Nagrinėjame pati blogiausią atvejį:

$$FF4(n) = n \sum_{i=0}^h \left(\frac{5}{6}\right)^i < \sum_{i=0}^{\infty} \left(\frac{5}{6}\right)^i n = \frac{n}{1 - \frac{5}{6}} = \frac{6}{1}n;$$

$$FF4(n) = O(n);$$

Toliau nagrinėjame geriausią atvejį.

Šis medis nėra simetrinis, todėl medžio aukštis h yra $\lceil \log_3 n \rceil \leq h \leq \lceil \log_2 n \rceil$;

Sprendžiame uždavinį:

$$FF4(n) = \sum_{i=0}^h \left(\frac{5}{6}\right)^i = \frac{\left(\frac{5}{6}\right)^{\lceil \log_3 n \rceil + 1} - 1}{\frac{5}{6} - 1} = \frac{6}{1} \left(1 - \left(\frac{5}{6}\right)^{\lceil \log_3 n \rceil + 1}\right) \geq n;$$

$$, nes 1 - \left(\frac{5}{6}\right)^{\lceil \log_3 n \rceil + 1} \geq \frac{1}{6}, kai n > 3;$$

$$FF4(n) = \left(\frac{5}{6} * \frac{1}{6}\right)n = \frac{5}{36}n;$$

$$FF4(n) = \lambda(n);$$

Kadangi galioja $FF4(n) = O(n)$ ir $FF4(n) = \lambda(n)$, tai galioja ir $FF4(n) = \theta(n)$, nes

$$\frac{5}{36}n < FF4(n) < 6n, kai visiems n > 3$$

Išstatome šiuos rezultatus į pagrindinę lygtį

$$T(n) = \left(FF4\left(\frac{n}{2}\right) + FF4\left(\frac{n}{3}\right)\right) * n + n$$

$$T(n) = (n) * n + n = n^2 + n$$

Atmetus n , kadangi ji labai maža, gauname, kad (kai mūsų $x = 1$, t.y. if sąlyga yra tenkinama)

$$T(n) = n^2$$

Kitu atveju mūsų lygtis gaunasi

$$T(n) = n$$

Antras sprendimo būdas:

Naudojame pagrindines teoremas ir surandame asimptotinį sudėtingumą.

Turime rekurentinę lygtį:

$$FF4(n) = FF4\left(\frac{n}{2}\right) + FF4\left(\frac{n}{3}\right) \text{ rekurentinę lygtį.}$$

Tikriname pirmąją teoremos dalį :

Užsirašome bendrą lygties formą:

$$FF4(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Iš šios lygties atrenkame kintamuosius iš savo duotosios lygties

$$a = 2, \text{ nes turime du rekursinius kreipinius}$$

$$b = \frac{\frac{n}{2}}{\frac{n}{3}} = \frac{3}{2}$$

$$f(n) = 0$$

Surandame pagalbines reikšmes, kurias naudosime teoremoje

$$\log_b(a) = \log_{\frac{3}{2}}(2) \approx 1,7$$

Pagal pagrindinę teoremą, turime palyginti šią reikšmę su mūsų $f(n)$,

- Jeigu $\log_b(a) > f(n)$, t.y. pirmasis atvejis, tada galime apskaičiuoti mūsų asimptotinį sudėtingumą:

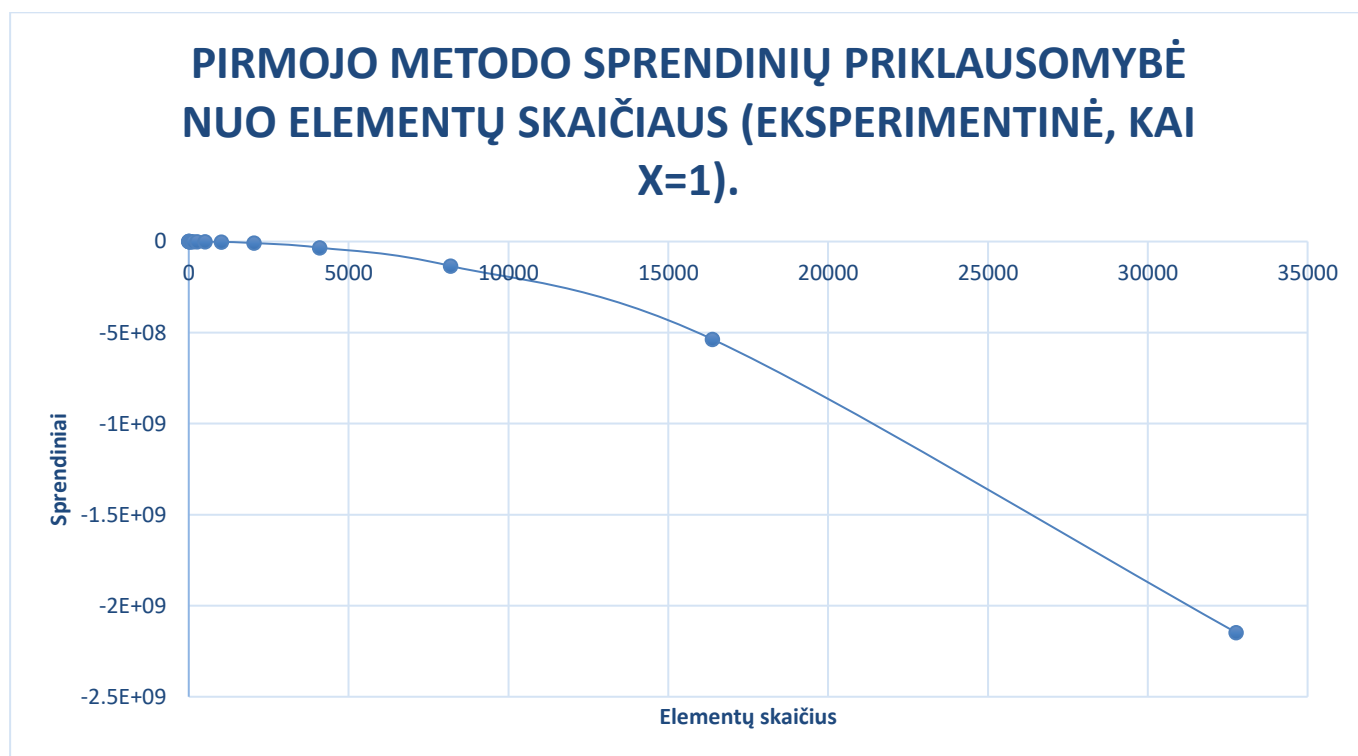
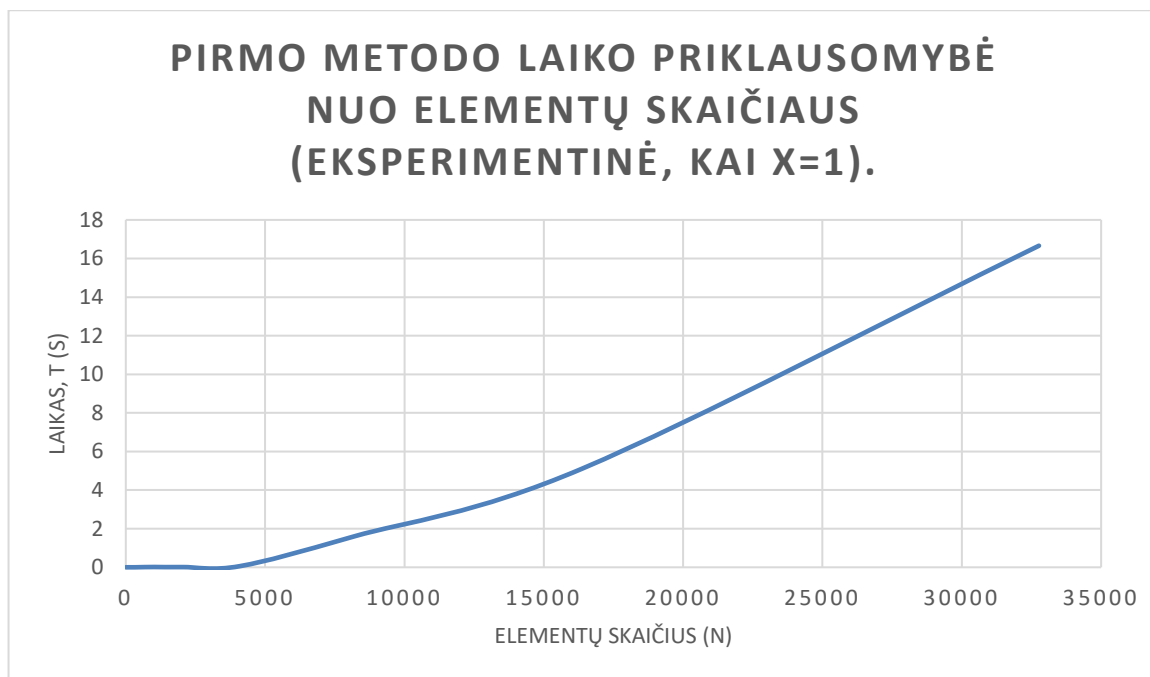
$$FF4(n) = n^{\log_b(a)} \approx n^{1,7}$$

Šis atvejis mums tinka ir galime nustatyti, kad mūsų rekursinės lygties asimptotinis sudėtingumas yra lygus $FF4(n) \approx n^{1,7}$. Blogiausias šio metodo asimptotinis sudėtingumas. Tada mūsų bendras asimptotinis sudėtingumas būtų

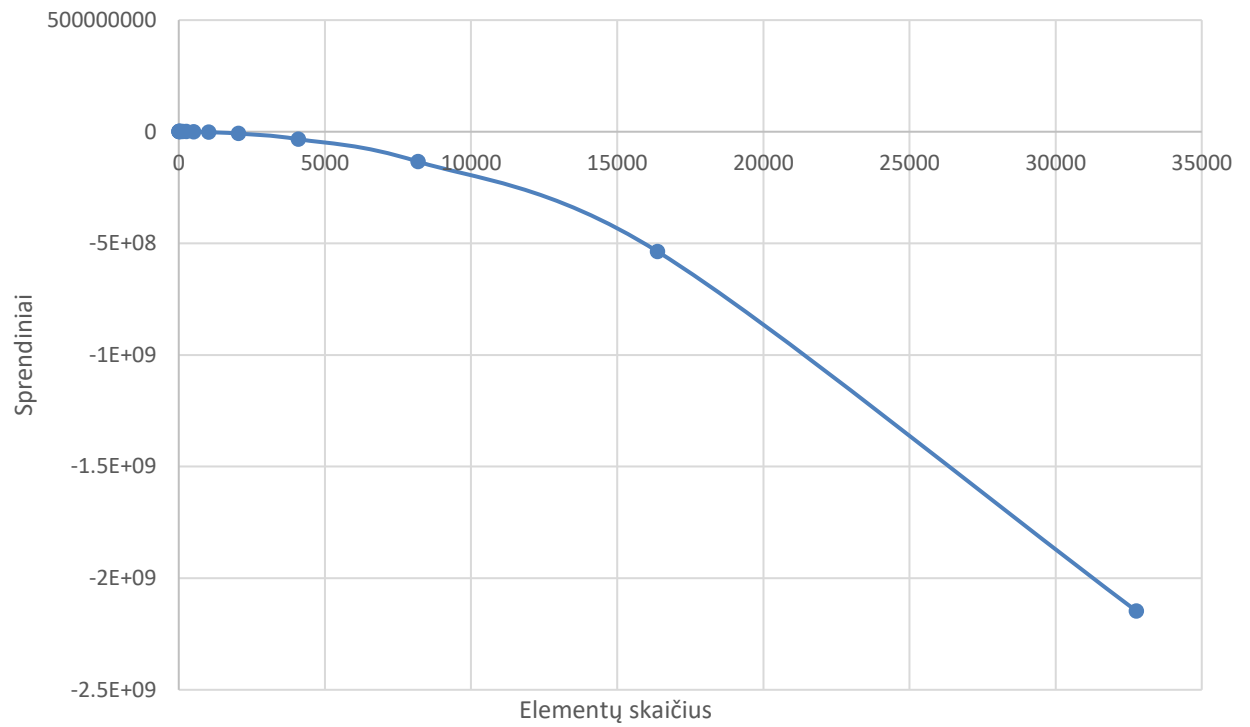
$$T(n) = (n^{1,7} * n) + n, \text{ jeigu } x = 1$$

- Jeigu $\log_b(a) = f(n)$, t.y. antrasis atvejis, šis atvejis mums netinka kadangi $f(n)$ nėra lygus $\log_b(a)$
- Jeigu $\log_b(a) < f(n)$, t.y. trečiasis atvejis, šis atvejis taip pat mums netinka, kadangi $f(n)$ nėra didesnis už $\log_b(a)$

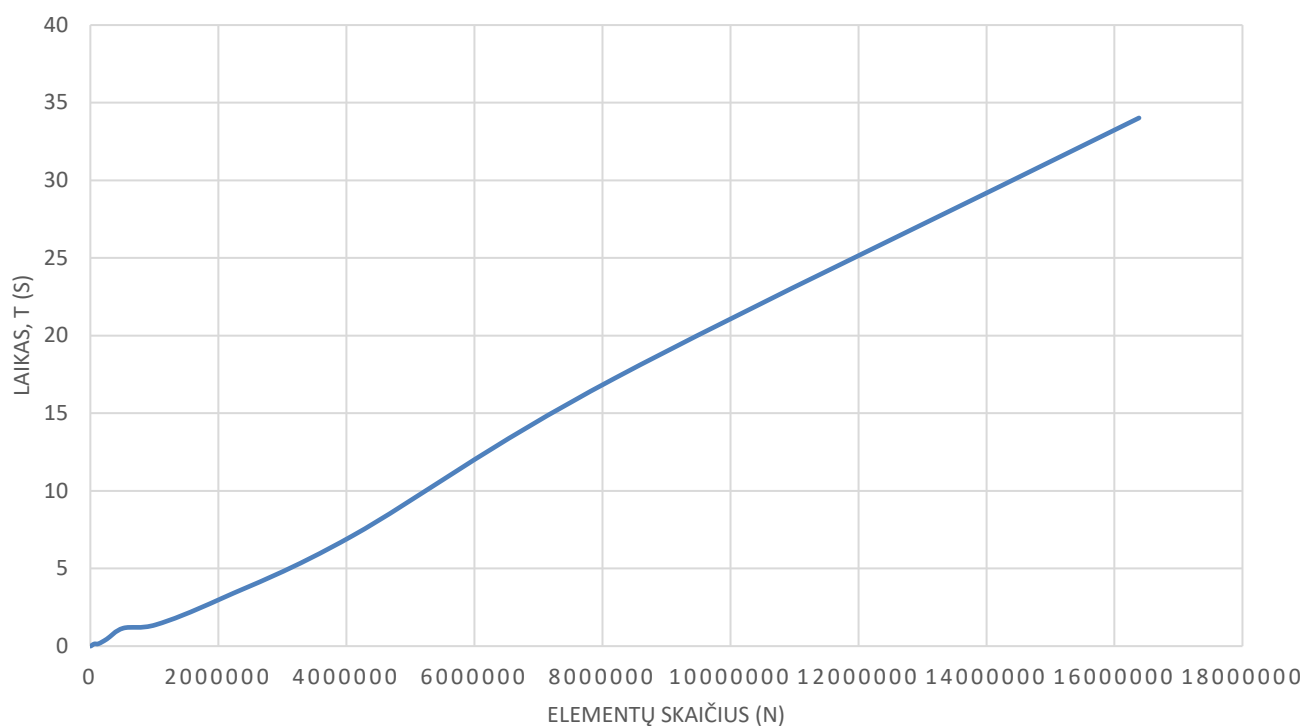
Atliekame našumo testus (pirmajam ir antrajam metodams).



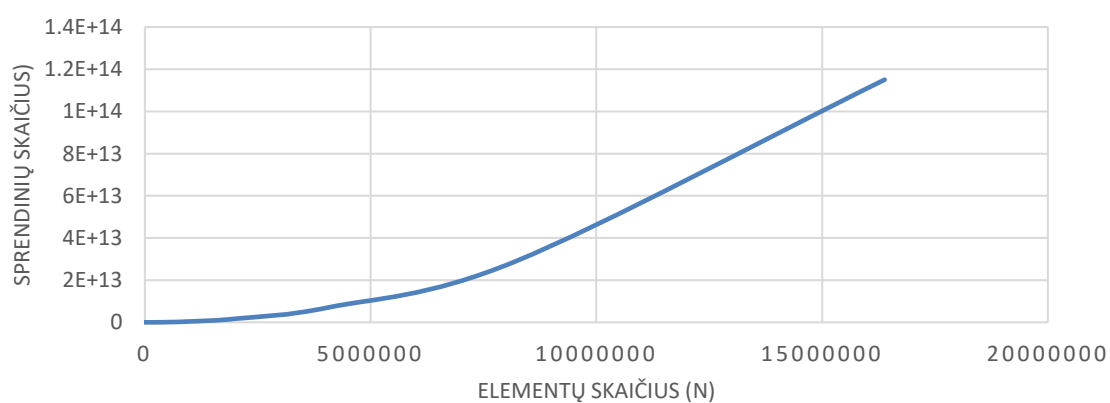
PIRMOJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ, KAI $\chi=1$)



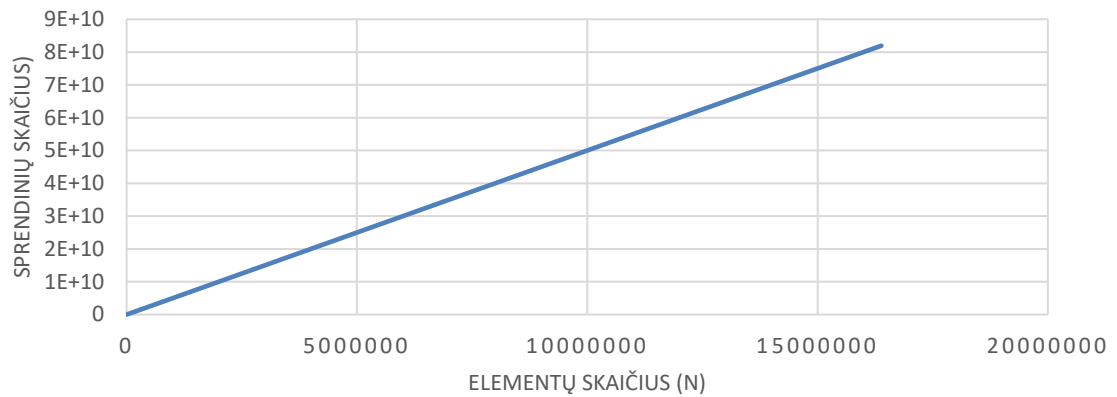
ANTROJO METODO LAIKO PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (EKSPERIMENTINĖ, KAI $X=1$).



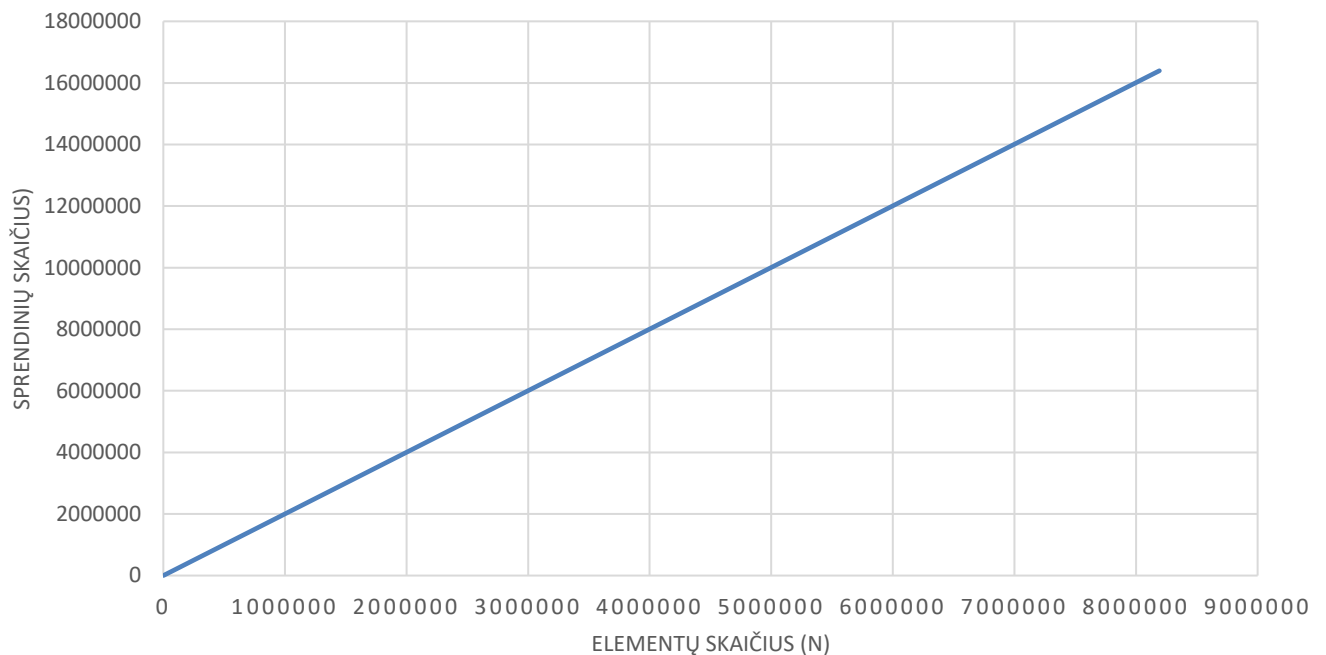
ANTROJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (EKSPERIMENTINĖ, KAI $X=1$) $O(N^2)$.



**ANTROJO METODO SPRENDINIŲ
PRIKLAUSOMYBĖ NUO ELEMENTŲ
SKAIČIAUS (TEORINĖ, KAI $X=1$) $O(N)$, NES
PRALEIDŽIAMA IF SĄLYGA SU DUOMENIMIS.**



**ANTROJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ
NUO ELEMENTŲ SKAIČIAUS (TEORINĖ, KAI $X=1$)
TESTINIAI DUOMENYS PEREINA FF4 METODO IF
SĄLYGĄ.**



2.2.2 Antrosios užduoties dalies sprendimas

Ant žaidimo lentos ($1 \times n$) langelių surašyti atsitiktiniai teigiami skaičiai (taškai). Pradėjęs pirmame langelyje, žaidėjas vieno ėjimo metu gali pasirinkti – pereiti į kitą langelį ($1 \rightarrow 2$) ir gauti tiek taškų, kiek yra tame langelyje, ar persokti du langelius ($1 \rightarrow 4$) ir gauti du kartus daugiau taškų, nei yra užrašyta tame langelyje. Kokia yra mažiausia taškų suma, kurią žaidėjas gali surinkti paskutiniame langelyje?

11 pav. Antra užduoties dalis.

2 užduoties dalis (6 balai):

- Pateikite rekursinį uždavinio sprendimo algoritmą (rekursinis sąryšis su paaiškinimais), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (rekursinis sprendimas netaikant dinaminio programavimo).
- Pritaikykite dinaminio programavimo metodologiją pateiktam uždaviniui (pateikti paaiškinimą), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (taikant dinaminį programavimą).
- Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.

Užduoties paaiškinimas:

Turime žaidimų lentą ant kurios atsitiktinai yra surašyti teigiami skaičiai (taškai). Katik pradėjęs žaidimą žaidėjas vieno ėjimo metu gali pasirinkti – pereiti į kitą langelį ir gauti tiek taškų, kiek tame langelyje arba persokti du langelius ir gauti du kartus daugiau taškų, nei yra užrašyta tame langelyje.

Reikia surasti mažiausią taškų suma, kurią žaidėjas gali surinkti paskutiniame langelyje.

Pačią lentą galime įsivaizduoti, kaip masyvą su duomenimis. Turime masyvo pradžios ir galo indeksus. Kadangi mums reikia pasiekti paskutinį masyvo indekso elementą ir apskaičiuoti šių taškų mažiausią sumą, kuria gali surinkti žaidėjas.

Pradžioje metodo, turime patikrinti dvi dalis: ar mes pasiekėme pabaigą mūsų masyvo t.y. mūsų pradinio ir galinio masyvo indeksai sutampa, jeigu taip grąžiname elementą. Kita dalis : ar jau tikriname priešpaskutinį elementą, tada grąžiname šių dviejų indeksų elementų sumą. Šiuos du atvejus patikrine, galime eiti prie pagrindinės algoritmo dalies.

Rekursyvinė arba dinaminė dalis skirsis sprendimo metodika. Idėja tokia, kad metoda kviečiame ir su Bibliotekos Math.Min funkcijos pagalba galime patikrinti, kuris ėjimas mums kainuoja mažiausiai t.y. ar ėjimas vienu langeliu ir gavimas tik tų taškų ar ėjimas dviem langeliais ir dvigubas taškų gavimas. Taip kviesdami metodus, tikriname šią mažiausią galima taškų sumą. Lygindami, kuris ėjimas mums mažiausiai kainuoja galime surasti pačią mažiausią taškų sumą.

Rekursinis uždavinio sprendimas (naudojant rekursinius sąryšius, rekursija) :

```
public static int MinimumPoints(int[] board, int start, int end)
{
    if (start == end) // pasiekėme lentos pabaigą //1
    {
        return board[start]; //1
    }
    else if (start + 1 == end) // esame priešpaskutinėje lentos
    pozicijoje //1
    {
        return board[start] + board[end]; //1
    }
    else // rekursija: pasirenkame ar šoksime vieną langelį ar
    du (mažesnio įvertinimas)
    {
        int pointsNext = MinimumPoints(board, start + 1, end);
        //T(n+1)
        int pointsJump = MinimumPoints(board, start + 2,
        end)*2; //T(n+2)
        return Math.Min(pointsNext, pointsJump) + board[start];
        //n
    }
}
```

Šiame metode pradžioje patikriname ar nepasiekėme lentos pabaigos ir ar nepasiekėme priešpaskutinio elemento. Jeigu viena iš situacijų pasitvirtina prie sumos pridedame tam tikrus rezultatus. Pagrindinė metodo implementacija yra du rekursiniai iškvietimai metodo, jame tikriname taškų sumą, kurią gauname perėjus vieną langelį arba du langelius, t.y. atrenkama mažiausia šių taškų suma ir pridedama prie bendro taškų kiekio. Taip implementuodami rekursiją galime gauti mažiausią taškų sumą.

Dinaminio programavimo uždavinio sprendimas (dinaminio programavimo metodologija) :

```
public static int MinimumPoints(int[] board, int start, int end)
{
    int[] minPoints = new int[end + 1]; //sukuriame masyva
    taskams kaupiti
    minPoints[end] = board[end]; //masyve irasome paskutinio
    langelio tasku reiksme

    for (int i = end - 1; i >= start; i--) //iteruojame nuo
    priespaskutines pozicijos ir tikriname tasku ejimus
    {
        if (i + 1 == end) //patikrinimas jeigu esame
        priespaskutineje pozicijoje
        {
            minPoints[i] = board[i] + board[end]; //pridedame
            prie masyvo reikiamas reiksmes
        }
        else //kitiems atvejams atliekame dviejų elementų
        lyginimą ir gražiname mažesnę taškų sumą
        {
            int pointsNext = minPoints[i + 1]; //vienas šuolis
            int pointsJump = minPoints[i + 2] * 2; //dvigubas
            šuolis
            minPoints[i] = Math.Min(pointsNext, pointsJump) +
            board[i]; //palyginimas
        }
    }

    return minPoints[start]; //grąžiname mažiausią taškų sumą
}
```

Kitaip negu naudojant rekursiją, šiuos problemos sprendimui naudojame dinaminio programavimo metodologiją, t.y. vengiame pasikartojančių veiksmų, rezultatų skaičiavimo. Todėl įsivedame masyvą minPoints (mažiausiai taškų sumai kaupiti) ir jame kaupiame reikalingus uždaviniui atlikti taškus. Iteruodami nuo galo išvengiame pabaigos patikrinimo sąlygos, taip sutaupydami resursų ir nekartodami skaičiavimų. Taip iteruodami per visą masyvą, atrenkame, kuris šokimas mums kainuoja mažiau taškų ir taip galime priskaičiuoti šiuos taškus ir laikyti masyve minPoints išvengdami rekursiškų pasikartojimų. Pabaigoje gražiname pati pradinį masyvo elementą – t.y. mūsų mažiausią taškų sumą.

Rekursinio algoritmo kodo analizė:

3 references				
public static int MinimumPoints(int[] board, int start, int end)				
{				
if (start == end) // pasiekėme lentos pabaigą //1	c1		1	
{				
return board[start]; //1	c2		1	
}				
else if (start + 1 == end) // esame priešpaskutinėje lentos pozicijoje //1	c3		1	
{				
return board[start] + board[end]; //1	c4		1	
}				
else // rekursija: pasirenkame ar šoksime vieną langelį ar du (mažesnio įvertinimas)				
{				
int pointsNext = MinimumPoints(board, start + 1, end); //T(n+1)	c5			T(n-1)
int pointsJump = MinimumPoints(board, start + 2, end) * 2; //T(n+2)	c6			T(n-2)
return Math.Min(pointsNext, pointsJump) + board[start]; //n	c7		n	
}				
}				

12 pav. Rekursinio kodo analizė.

Atliekame rekursinio metodo kodo analizę ir susumuojame visus svorius.

c_i – konstantos, kurios yra $O(1)$;

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 * T(n - 1) + c_6 * T(n - 2) + c_7 * n$$

Suprastiname visas konstantas ir gauname rekursinių sąryšių lygtį :

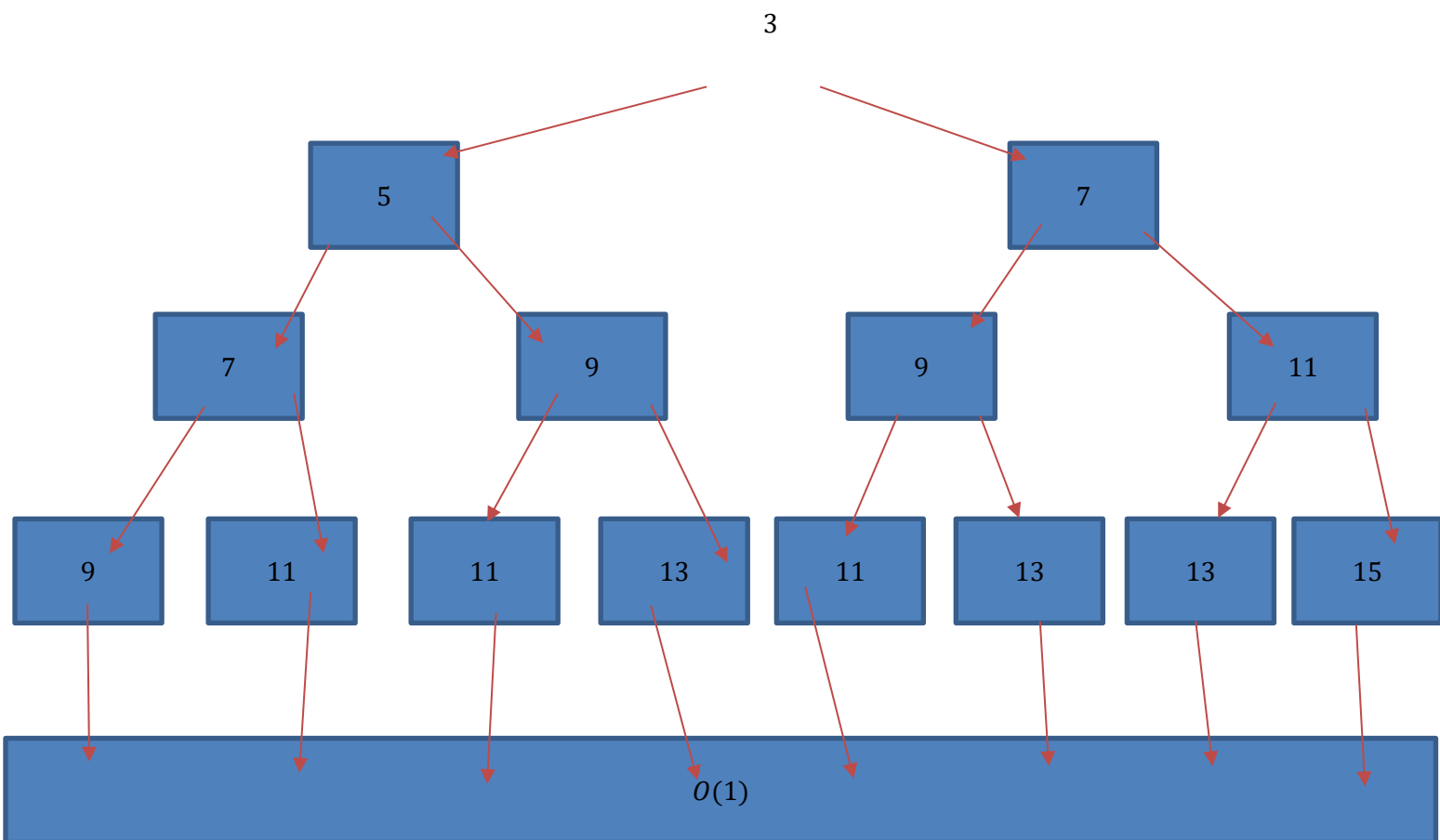
$$T(n) = (n - 1) + T(n - 2) + n \text{ t.y. } \textit{blogiausias mūsų metodo asimptotinis sudėtingumas}$$

Pats geriausias metodo asimptotinis sudėtingumas bus :

$$T(n) = c_1 + c_2 + c_3 + c_4$$

Šią lygtį galime išspręsti naudodami medžio metodą.

Iš pradžių susidarome sprendinių medį



Rekurentinis sprendimo medis

Apskaičiuojame kiekvieno lygio sumą:

1 lygis : 3

2 lygis : 12

3 lygis : 36

4 lygis: 96

Sudarome rekursinę lygčių sistema, šiai rekurentinei lygčiai.

$$\begin{cases} S_0 = 0 \\ S_n = 2S_{n-1} + 3 * 2^{n-1} \end{cases}$$

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

$$S_n = 3n2^{n-1};$$

$$S_{n+1} = 2 * 3n2^{n-1} + 3 * 2^n = 3(n+1)2^n;$$

Randame sprendinį.

$$T(n) = \sum_{i=0}^h (2^i n - 3i2^{i-1});$$

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

Sprendžiame uždavinį.

$$\sum_{i=0}^h 2^i - 3 \sum_{i=0}^h i2^{i-1} = (2^{h+1} - 1) - 3(h2^h - 2^h + 1);$$

Apatinį įvertinimą galime įvertinti, kurio visos šakos yra neilgesnės, kaip $h = \lceil \frac{n}{3} \rceil$;

$$T(n) = \lambda \left(2^{\frac{n}{3}} \right) \approx O(2^n * n) \approx O(n2^n)$$

Negavome asimptotiškai tikslaus įvertinimo.

Dinaminio programavimo algoritmo kodo analizė:

	Svoris	Kiekis
<code>int[] minPoints = new int[end + 1]; //sukuriame masyva taskams kaupiti</code>	c1	n
<code>minPoints[end] = board[end]; //masyve irasome paskutinio langelio tasku reiksme</code>	c2	1
<code>for (int i = end - 1; i >= start; i--) //iteruojame nuo priespaskutines pozicijos ir tikriname tasku ejimus</code>	c3;c4;c5	1;n+1;n
<code>{</code>		
<code>if (i + 1 == end) //patikrinimas jeigu esame priespaskutineje pozicijoje</code>	c6	1
<code>{</code>		
<code>minPoints[i] = board[i] + board[end]; //pridedame prie masyvo reikiamas reiksmes</code>	c7	1
<code>}</code>		
<code>else //kitiems atvejams atliekame dviejų elementų lyginimą ir gražiname mažesnę taškų sumą</code>		
<code>{</code>		
<code>int pointsNext = minPoints[i + 1]; //vienas šuolis</code>	c8	n
<code>int pointsJump = minPoints[i + 2] * 2; //dvigubas šuolis</code>	c9	n
<code>minPoints[i] = Math.Min(pointsNext, pointsJump) + board[i]; //palyginimas</code>	c10	n
<code>}</code>		
<code>return minPoints[start]; //grąžiname mažiausią taškų sumą</code>		
<code>}</code>	c11	1

13 pav. Dinaminio kodo analizė.

Atliekame dinaminio kodo analizę ir susumuojame visus svorius ir gauname mūsų metodo asimptotinį sudėtingumą.

c_i – konstantos, kurios yra $O(1)$;

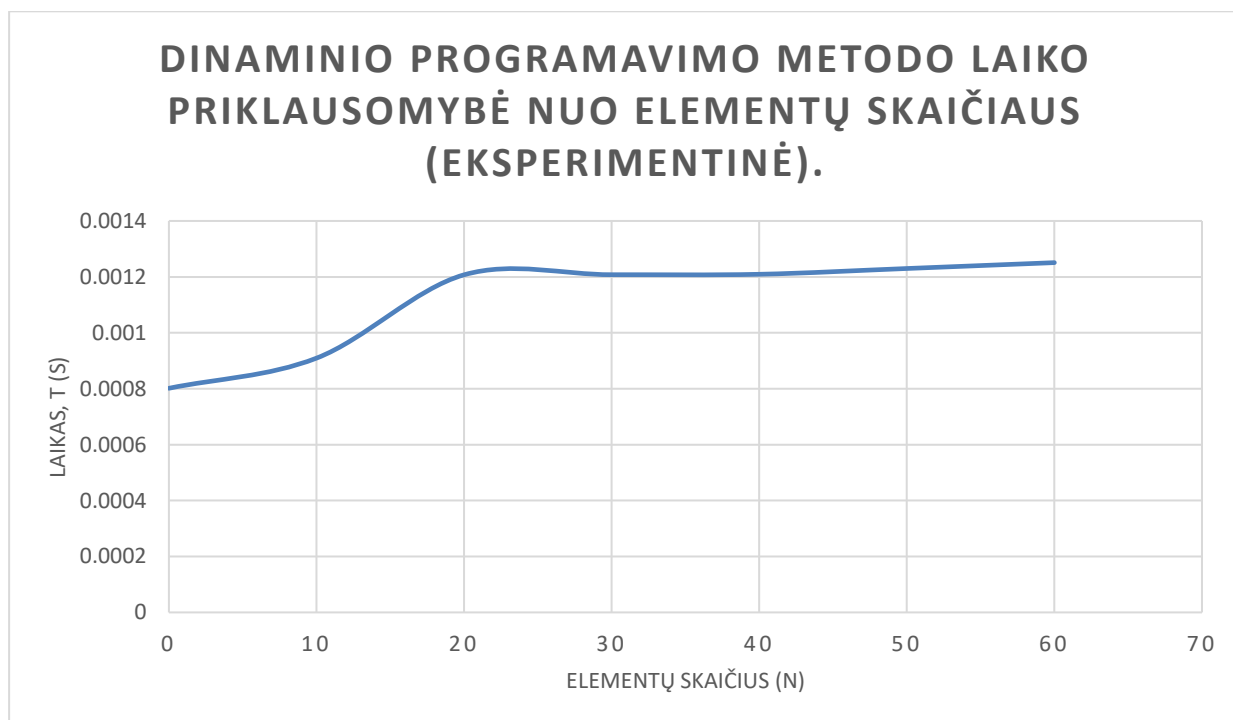
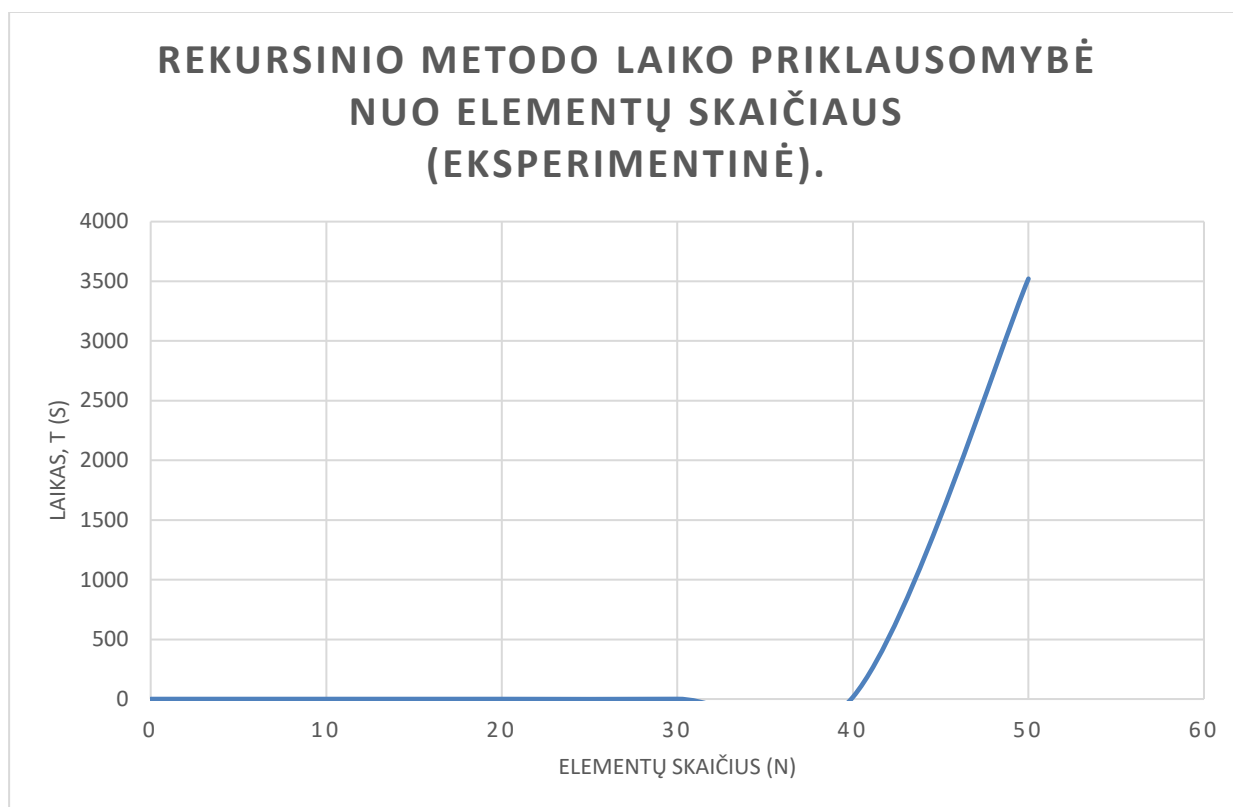
$$T(n) = c_1n + c_2 + c_3 + c_4(n + 1) + c_5(n) + c_6(n) + c_7(n) + c_8(n) + c_9(n) + c_{10}(n) + c_{11}$$

Suprastiname konstantas ir gauname, kad blogiausiu ir geriausiu atveju mūsų metodo asimptotinis sudėtingumas yra lygus (jeigu duomenys yra teisingi)

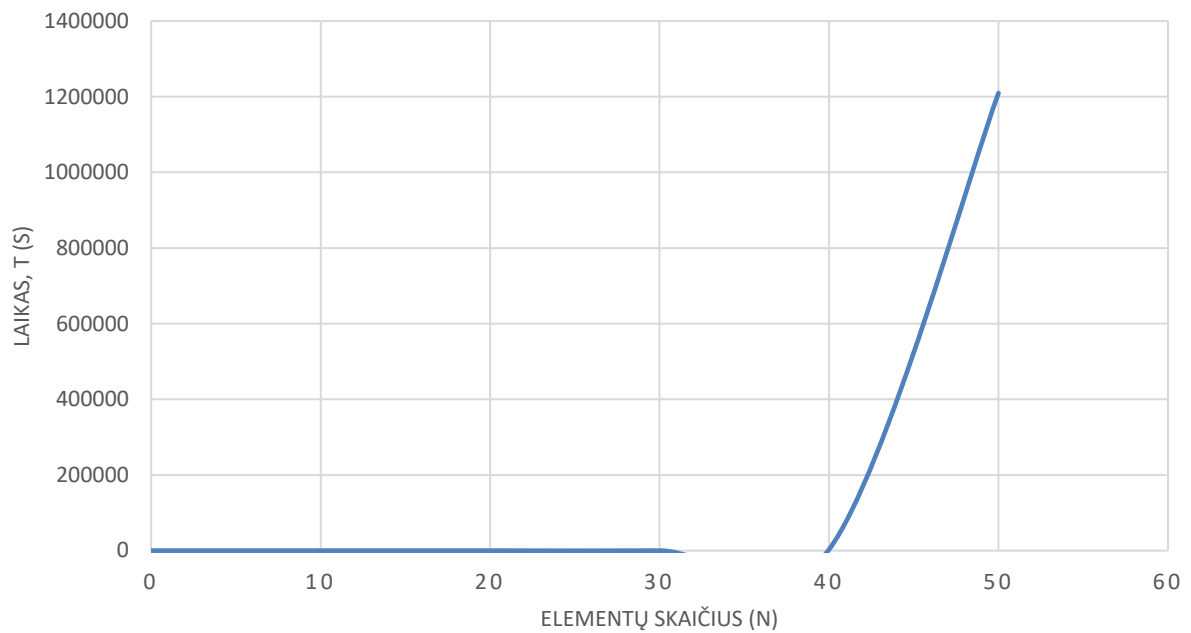
$$T(n) = n$$

IŠIMTIS : MATH.MIN funkcija padaro mūsų algoritmą n^2

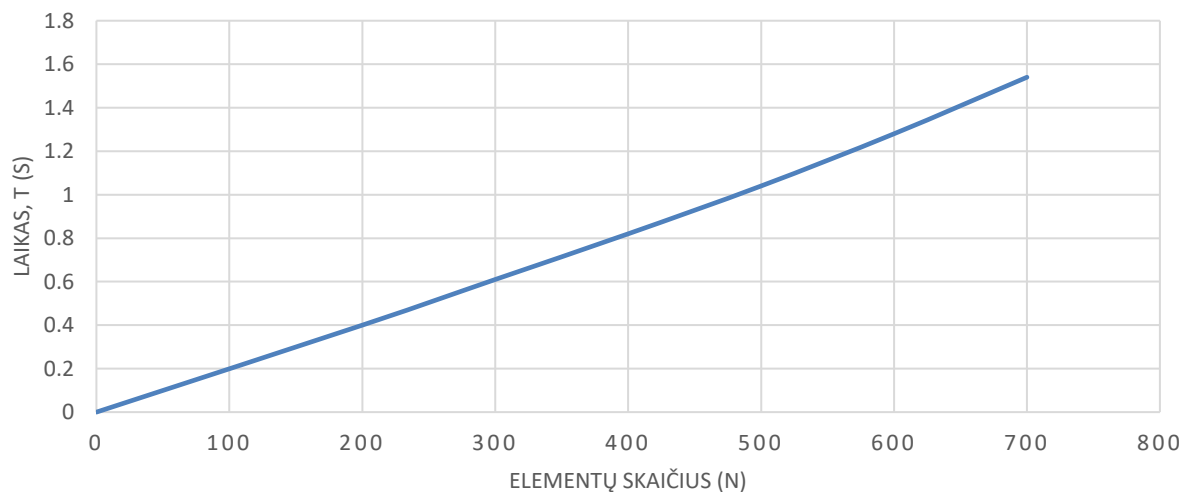
Našumo analizė :



REKURSINIO METODO LAIKO PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ).



DINAMINIO PROGRAMAVIMO METODO LAIKO PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ).



2.3. Šaltiniai

<https://moodle.ktu.edu/course/view.php?id=2470>

3. 3 LD laboratorinis darbas

4. 4 LD laboratorinis darbas