KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Algoritmų sudarymas ir analizė (P170B400)

Laboratorinių darbų ataskaita

Atliko:
IFF-1/6 gr. studentas
Lukas Kuzmickas
2023 m. gegužės 2 d.
Priėmė:
Doc. Pilkauskas Vytautas

KAUNAS 2023

TURINYS

1. 1 LD laboratorinis darbas

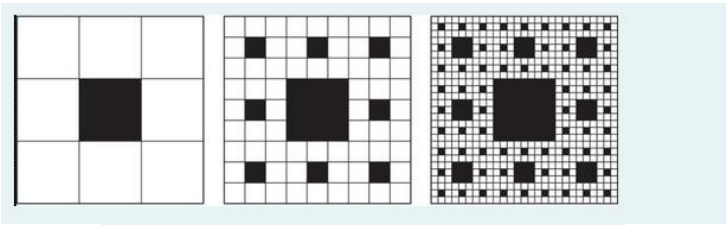
1.1. Pradinė užduotis

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	Kaina	Kiekis
`static ulong T1(ulong[] arr)`		
`{`		
`  ulong result = 0;`	c1	1
`  if (arr.Length >= 8)`	c2	1
`  {`		
`  result += T1(new ulong[arr.Length / 8]);`	T(n/8)	1
`  result += T1(new ulong[arr.Length / 8]);`	T(n/8)	1
`  for (int i = 0; i < arr.Length; i++)`	c3;c4;c5	1;n+1;n
`  for (int j = 0; j < arr.Length; j++)`	c6;c7;c8	n;(n^2)+1;n^2
`  for (int k = 0; k < arr.Length; k++)`	c9;c10;c11	n^2;(n^3)+1;n^3
`  for (int l = 0; l < arr.Length; l++)`	c12;c13;c14	n^3;(n^4)+1;n^4
`  result += 1;`	c15	n^4
`  }`		
`  return result;`	c16	1
`}`		

3 pav. Pirmojo metodo kodo analizė

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

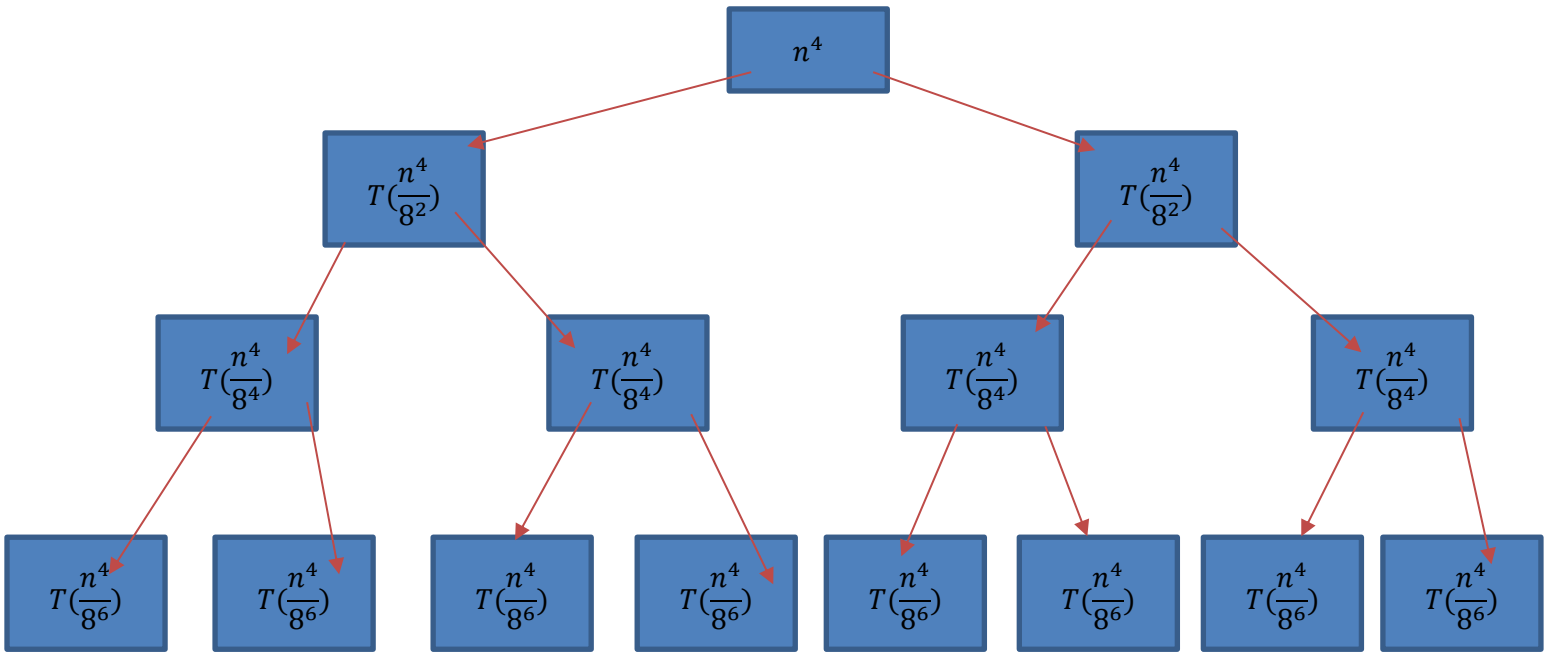
$$T(n) = c_1 + c_2 + T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + c_3 + c_4 n + c_5 n + c_6 n + c_7 n^2 + c_8 n^2 + c_9 n^2 + c_{10} n^3 + c_{11} n^3 + c_{12} n^3 + c_{13} n^4 + c_{14} n^4 + c_{15} n^4 + c_{16}$$
$$= T\left(\frac{n}{8}\right) + T\left(\frac{n}{8}\right) + c_1 + c_2 + c_{16}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

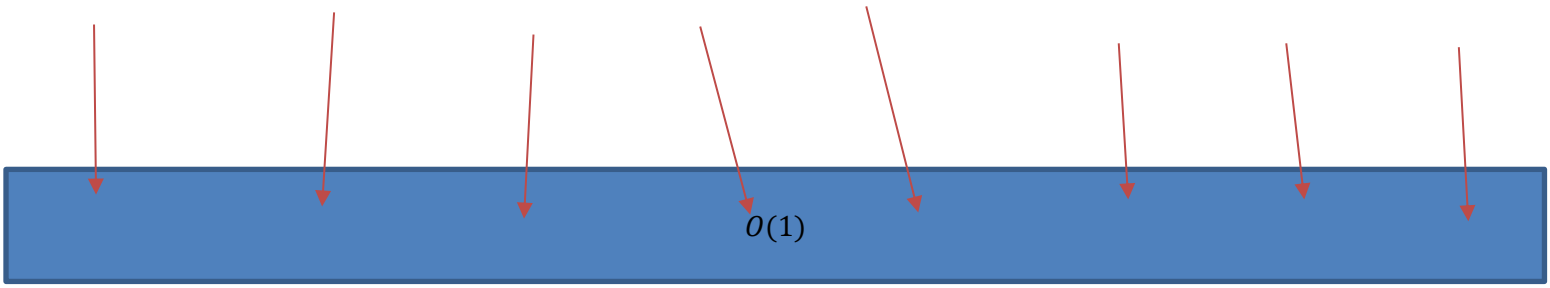
Atmetus konstantas, suprastinus šią išraišką, gauname panašią rekurentinę lygtį į mūsų:
$T(n) = 2 * T\left(\frac{n}{8}\right) + n^4;$

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

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

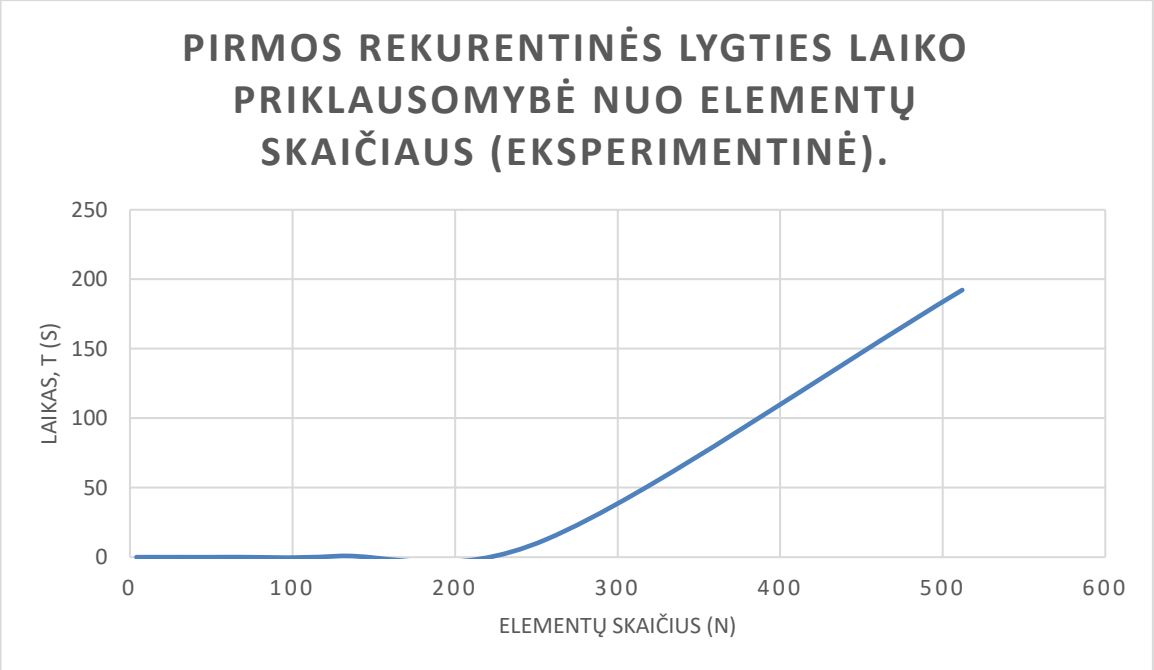
$T(n) > \frac{1023}{1024} * \frac{32}{31} * n^4 = \frac{33}{32} n^4$;

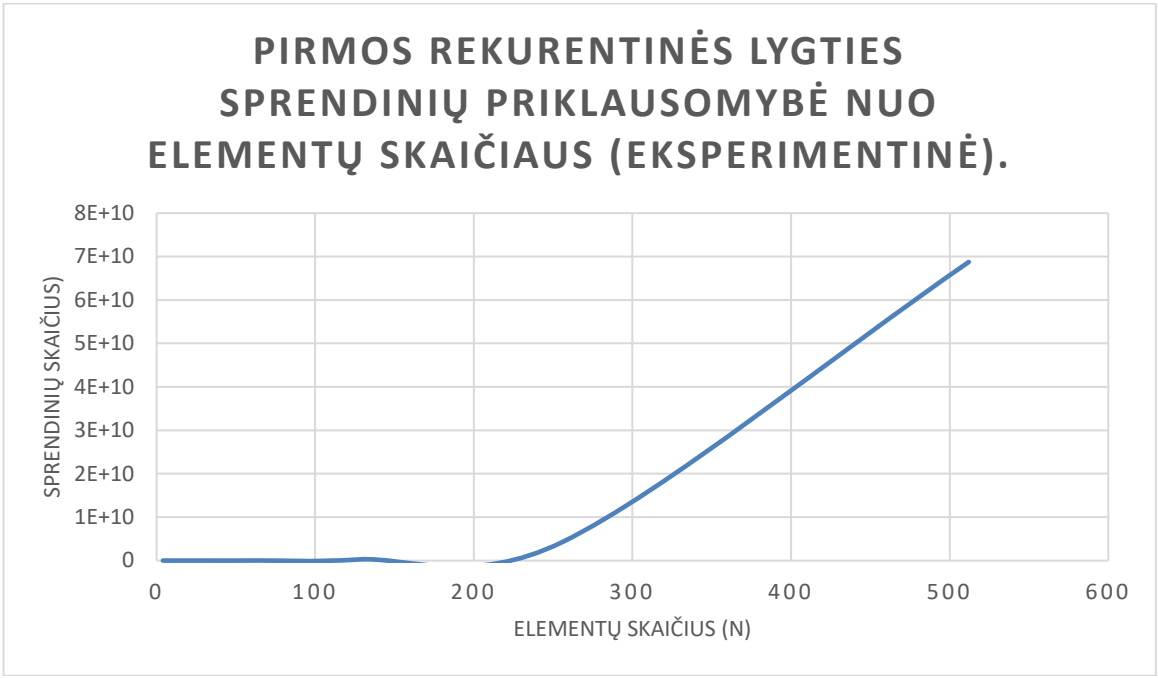
$T(n) = \lambda(n^4)$;

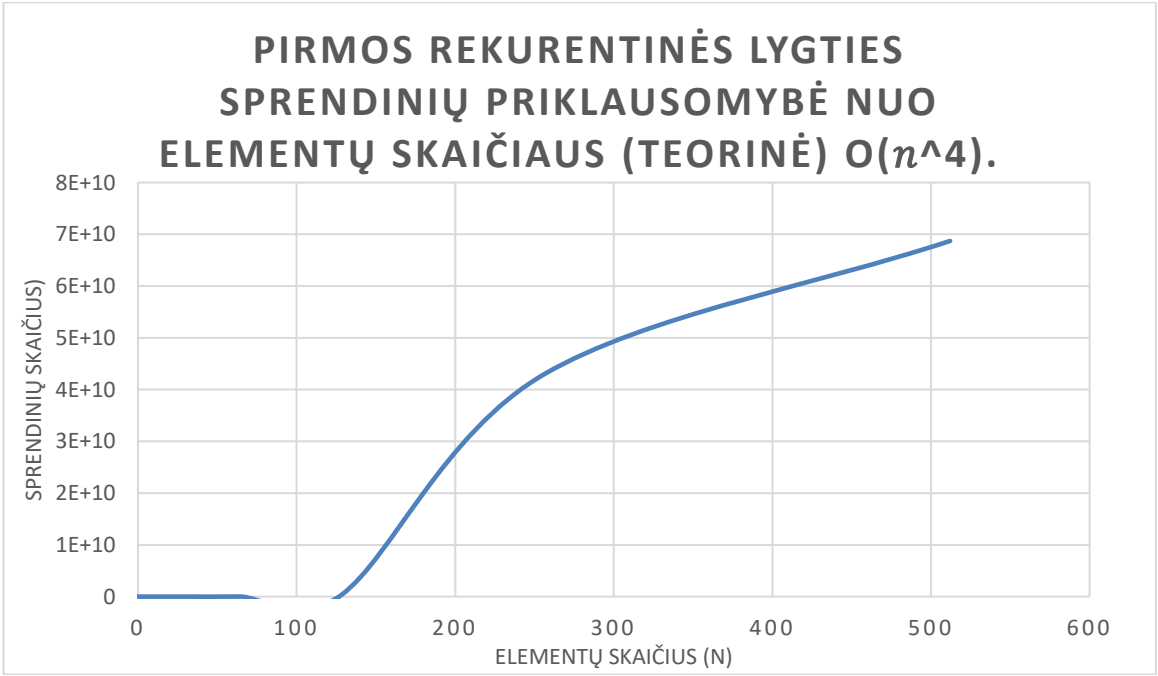
Kadangi galioja $T(n) = O(n^4)$ *ir* $T(n) = \lambda(n^4)$, tai galioja ir $T(n) = \theta(n^4)$,nes

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

Našumo testai:




PIRMOS REKURENTINĖS LYGTIES SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (EKSPERIMENTINĖ).
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)


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

5 pav. Antrojo metodo kodo analizė

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

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

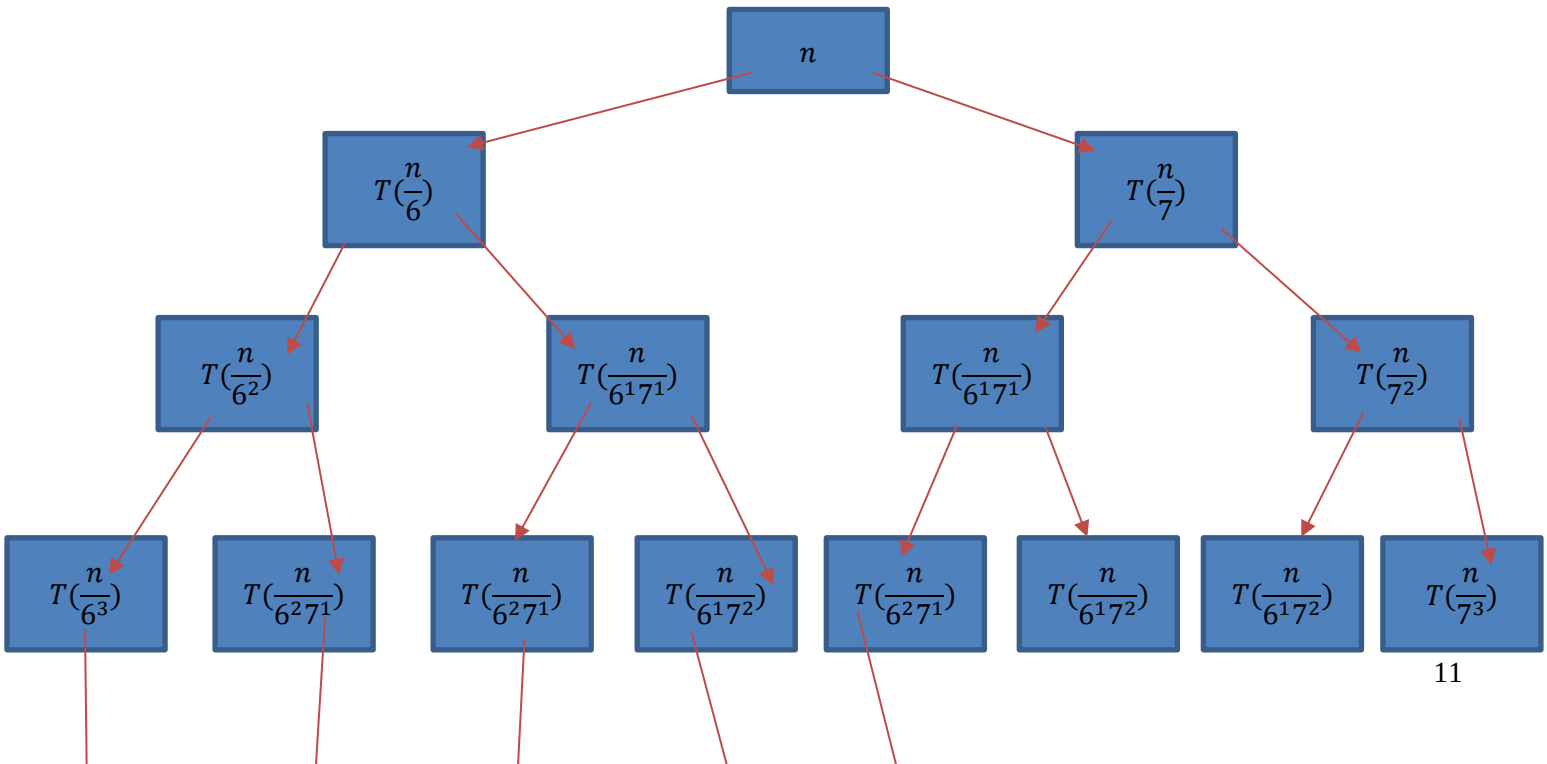
Atmetus konstantas, gauname panašią rekurentinę lygtį į mūsų:

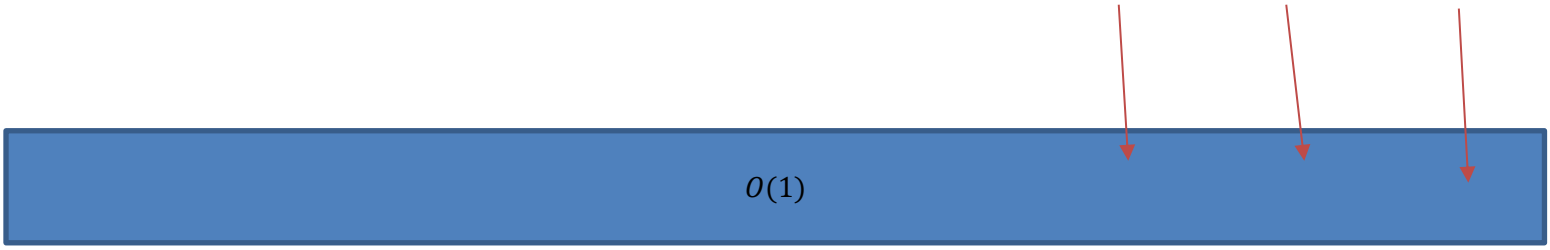
$T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{n}{7}\right) + n;$

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* $n > 0$;

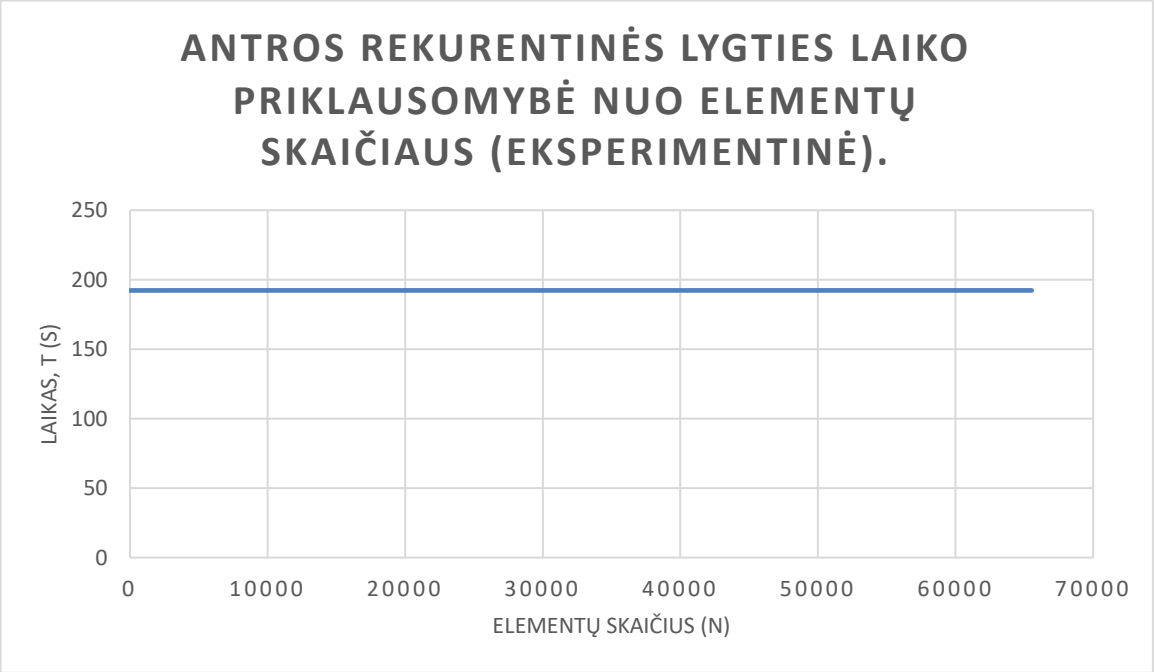
$T(n) = \frac{42}{29} * \frac{29}{42} * n = n;$

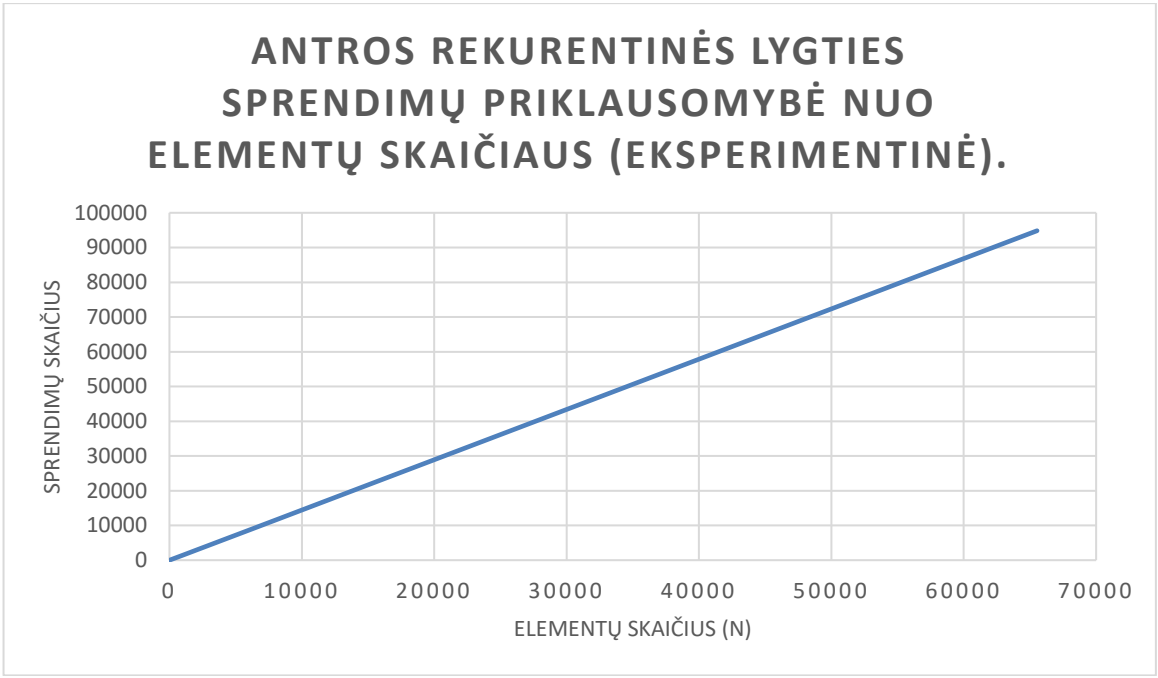
$$T(n) = \lambda(n);$$

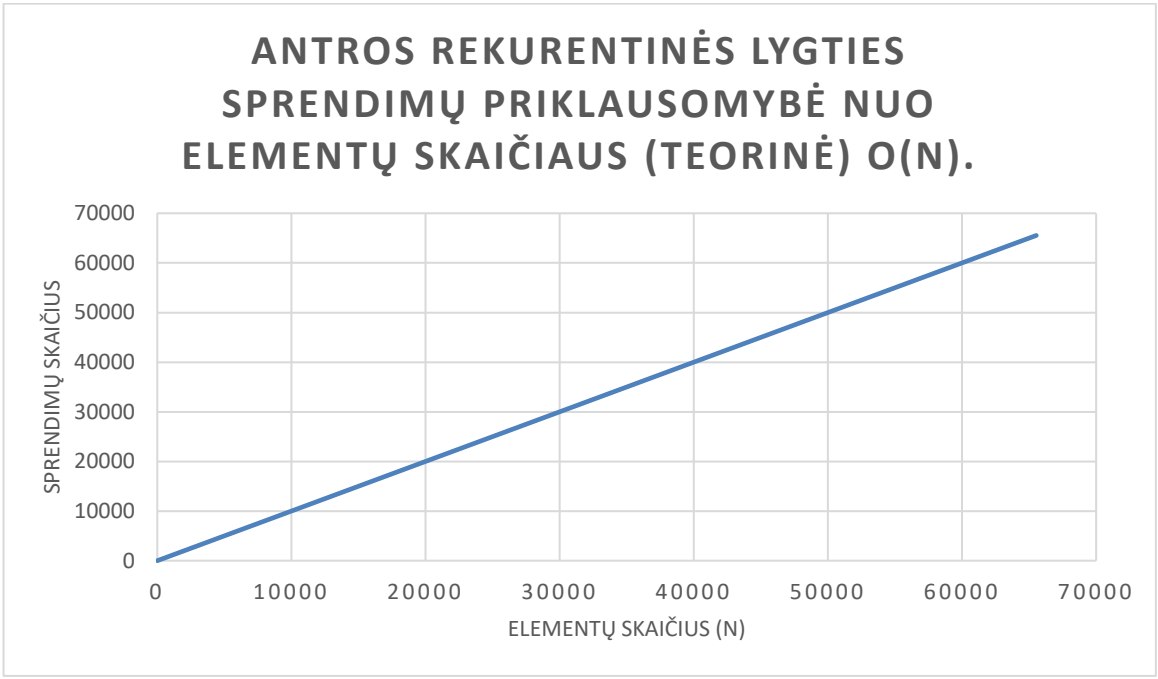
Kadangi galioja $T(n) = O(n)$ *ir* $T(n) = \lambda(n)$, tai galioja ir $T(n) = \theta(n)$,nes

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

Našumo testai:




ANTROS REKURENTINĖS LYGTIES SPRENDIMŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (EKSPERIMENTINĖ).
SPRENDIMŲ SKAIČIUS
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0
10000
20000
30000
40000
50000
60000
70000
ELEMENTŲ SKAIČIUS (N)


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.

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

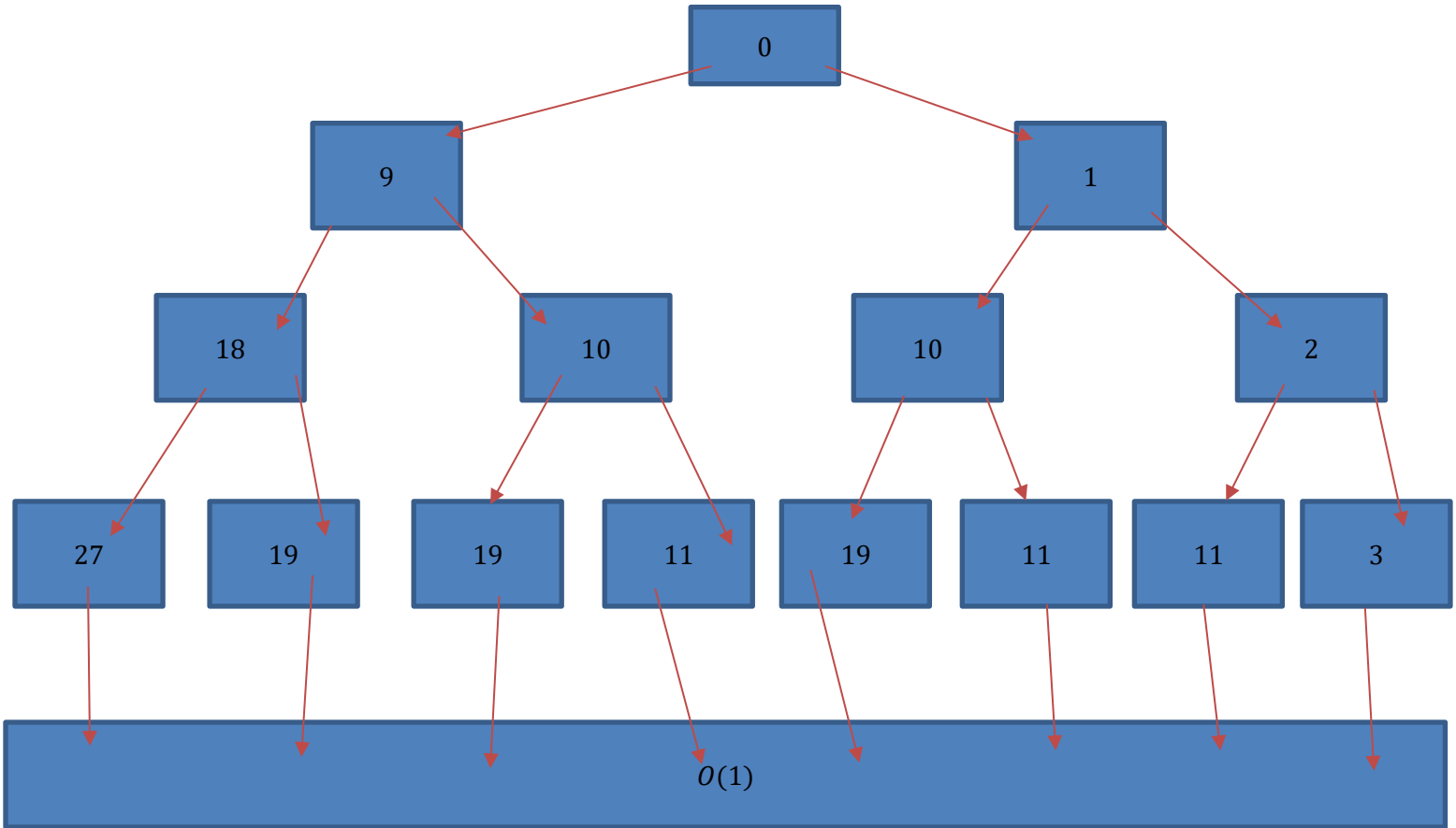
Atmetus konstantas, gauname panašią rekurentinę lygtį į mūsų:

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

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).

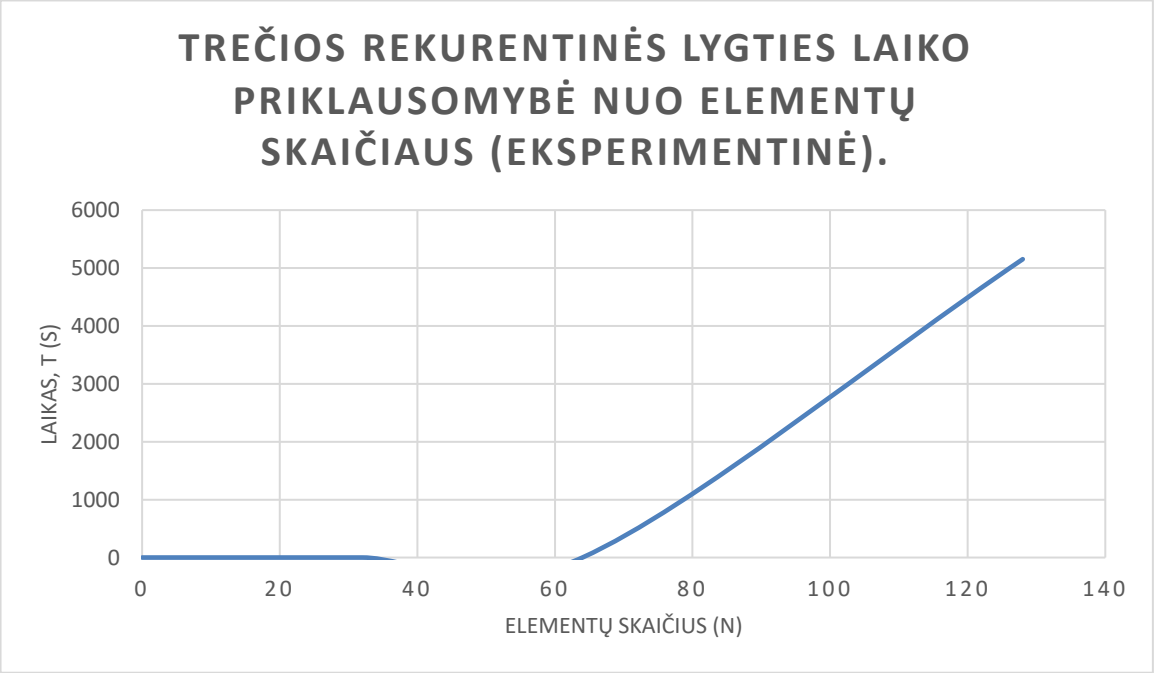
Sprendžiame uždavinį.

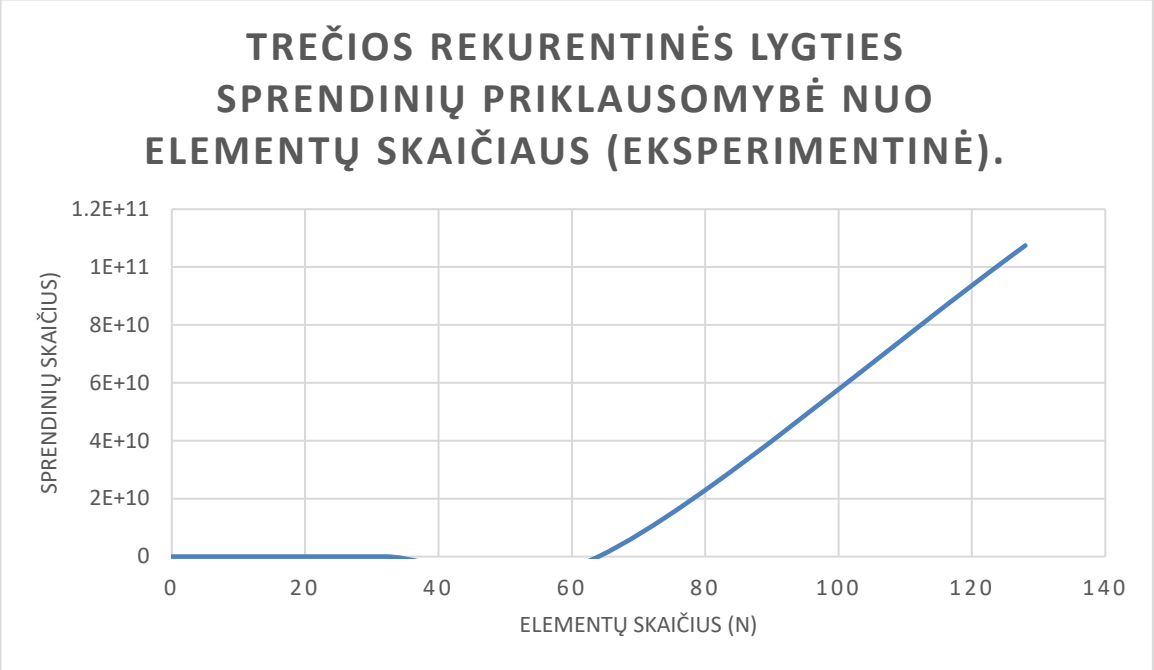
$\sum_{i=0}^{h} 2^i - 10\sum_{i=0}^{h} i2^{i-1} = (2^{h+1} - 1) - 10(h2^h - 2^h + 1);$

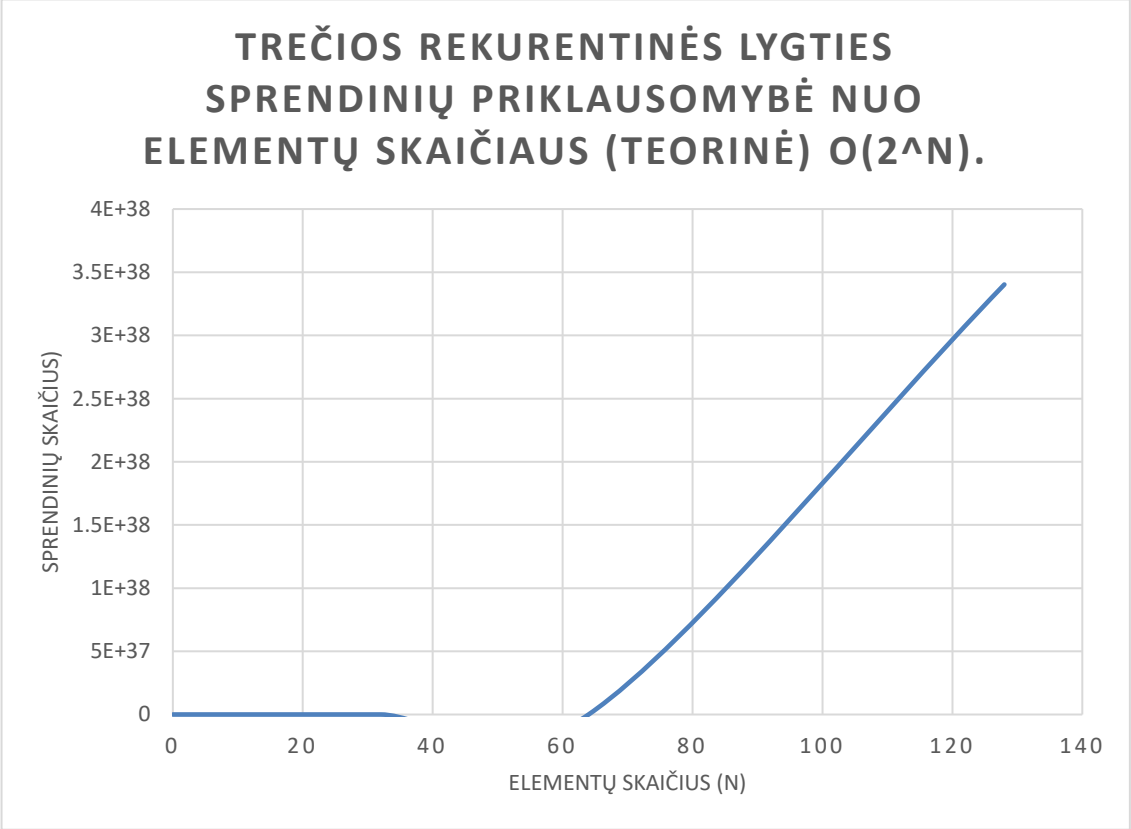
Apatinį įvertinimą galime įvertinti, kurio visos šakos yra neilgesnės, kaip $h = [\frac{n}{9}]$;

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

Našumo testai:







Trečiosios rekurentinės lygties kodas:

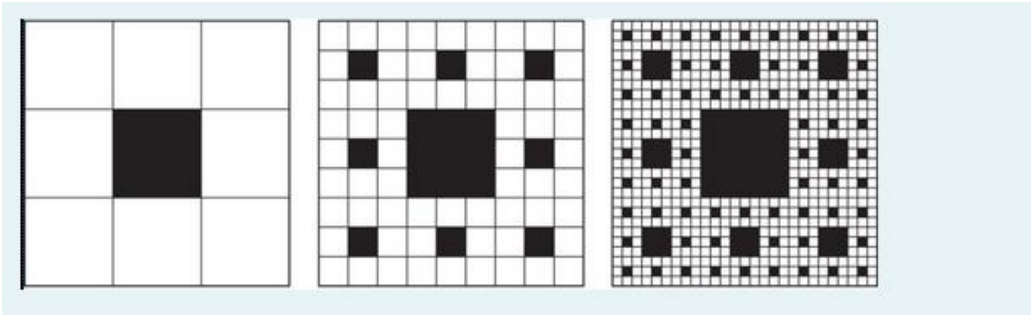
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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):

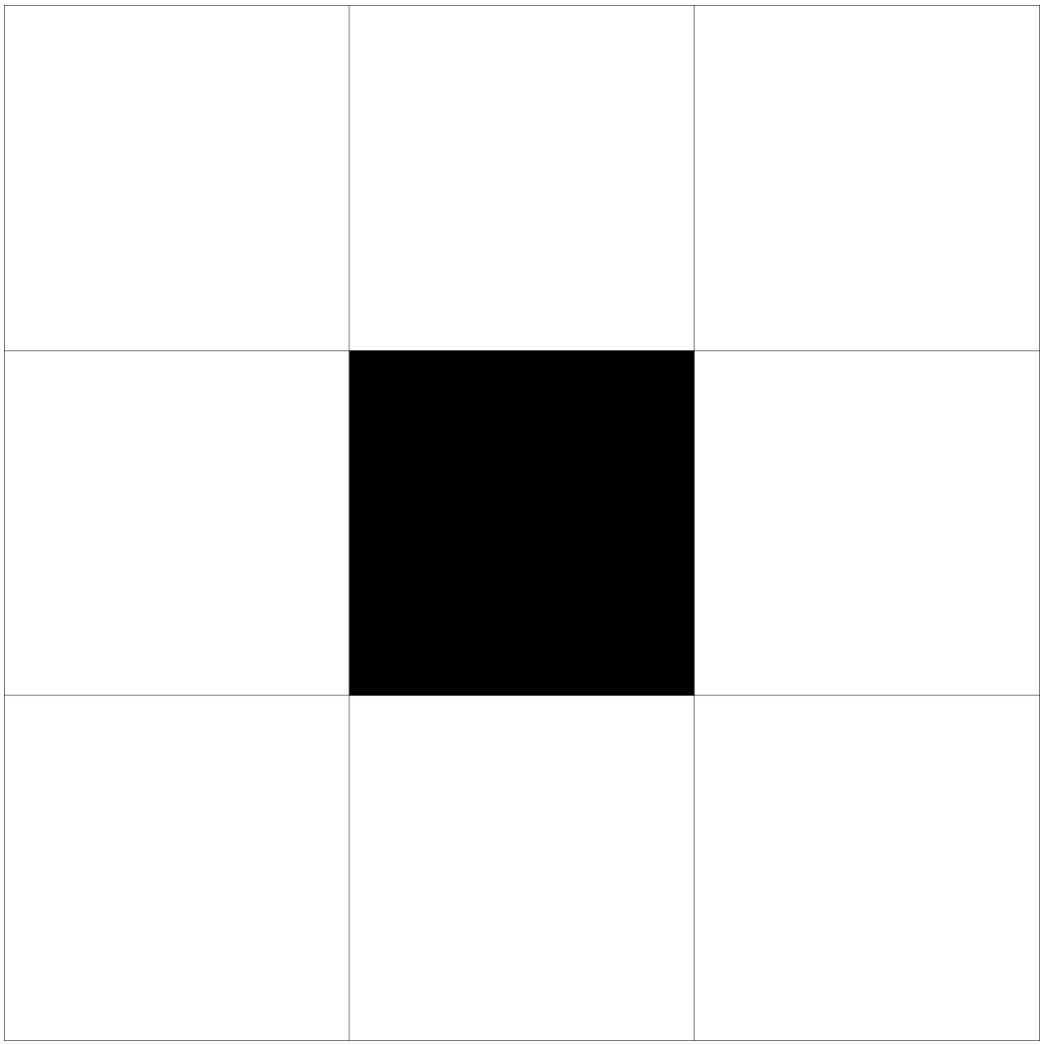
- Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas. (3 balai)
- Eksperimentiškai nustatykite darbo laiko ir veiksmų skaičiaus priklausomybę nuo generuojamo paveikslėlio dydžio (taškų skaičiaus). Gautus rezultatus atvaizduokite grafikais. Grafiką turi sudaryti nemažiau kaip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



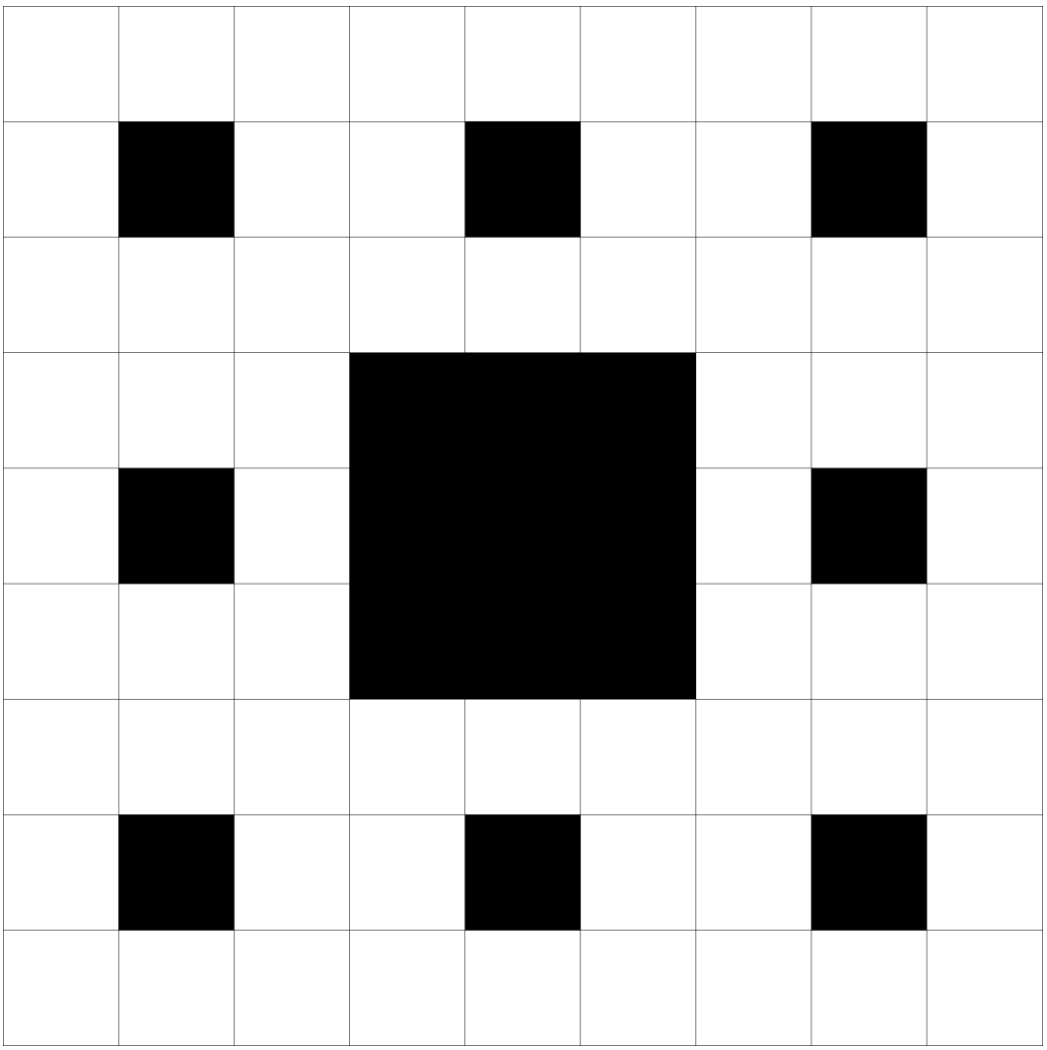
Pav. BMP formato užduotis.

Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas:

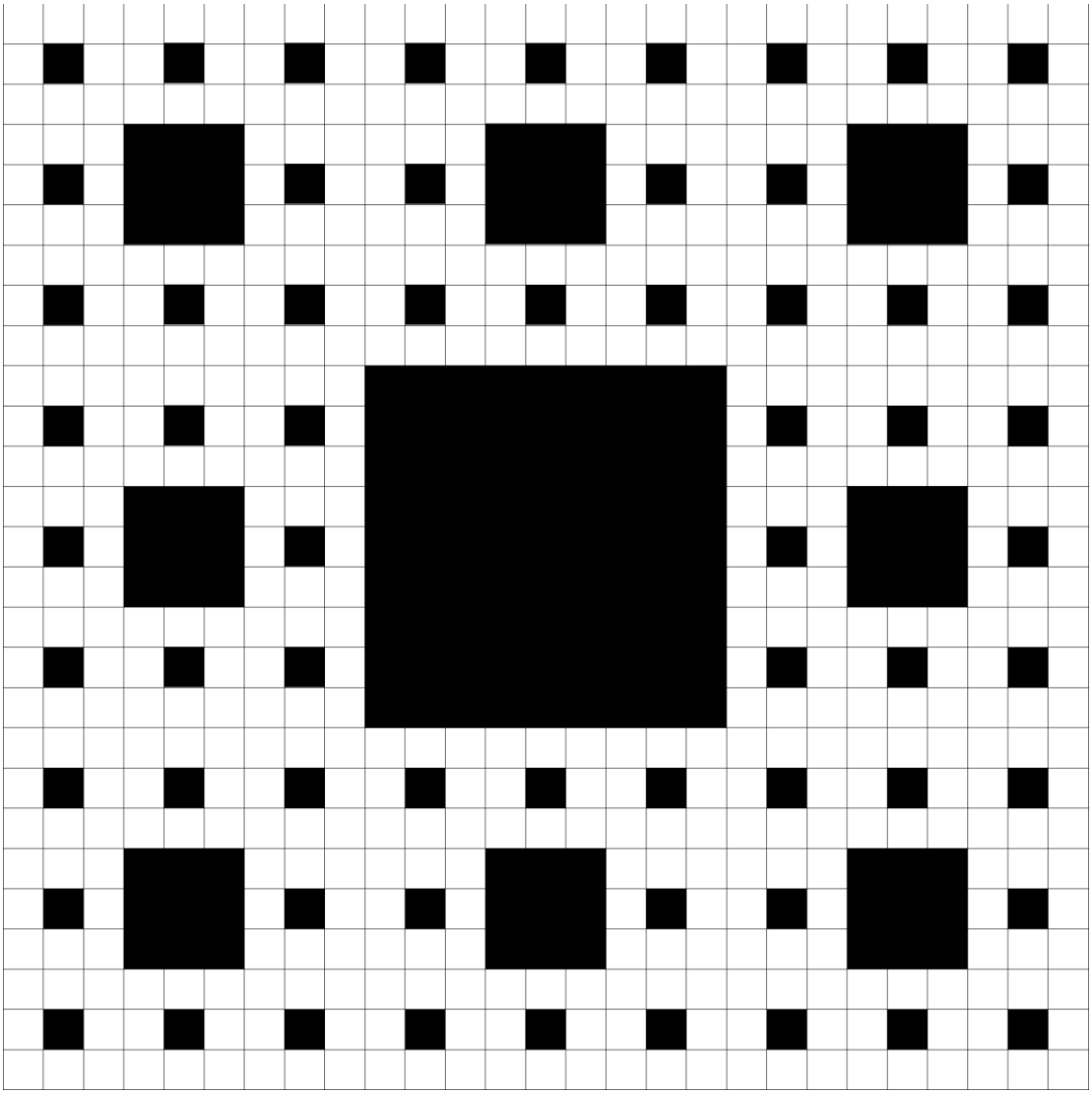
Pirma nuotrauka (image1.bmp)



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

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 lygį. 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;
}
```

8 pav. Pirmos užduoties metodas.

```
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 užduoties metodas.

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.

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→2) ir gauti tiek taškų, kiek yra tame langelyje, ar peršokti du langelius (1→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.

```
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;
}
```

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.

```
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;
}
```

Svoris	Kiekis
c1	1
c2,c3,c4	1;n+1;n
Σ i to n FF4(i)	1
c5	1

12 pav. Antrojo metodo kodo analizė.

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

Po analizės susumuojame visus šiuos svorius ir apskaičiuojame rekurentinę lygtį.

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

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

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

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

Ši sąlyga galioja, kai mūsų x = 1.

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

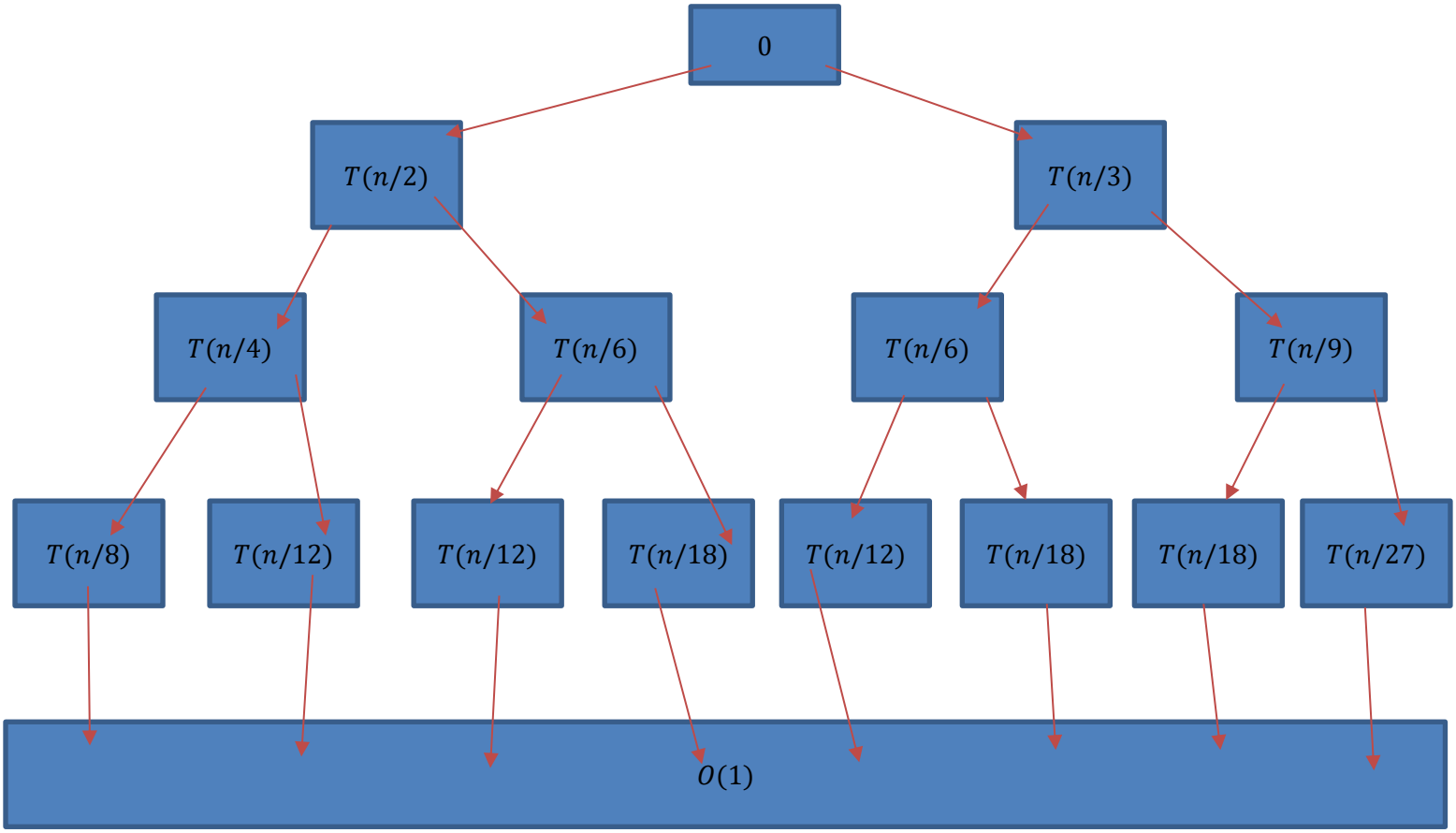
Ši sąlyga galioja, kai mūsų x = 0.

$$T(n) = n;$$

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).

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

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

Nagrinėjame pati blogiausią atvejį:

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

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

Toliau nagrinėjame geriausią atvejį.

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

Sprendžiame uždavinį:

$$FF4(n) = \sum_{i=0}^{h} (\frac{5}{6})^i = \frac{(\frac{5}{6})^{[log_7 n]+1} - 1}{\frac{5}{6} - 1} = \frac{6}{1}\left(1 - (\frac{5}{6})^{[log_7 n]+1}\right) \geq n;$$

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

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

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

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

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

Įstatome šiuos rezultatus į pagrindinę lygtį

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

$$T(n) = (n) * n + n = n^2 + \text{n}$$

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.

Turimę rekurentinę lygtį:

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

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

$$a = 2, nes\ turime\ du\ rekursinius\ kreipinius$$

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

$$f(n) = 0$$

Surandame pagalbines reikšmes, kurias naudosime teoremoje

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

Pagal pagrindine teorema, turime palyginti šią reikšmę su mūsų f(n),

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

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

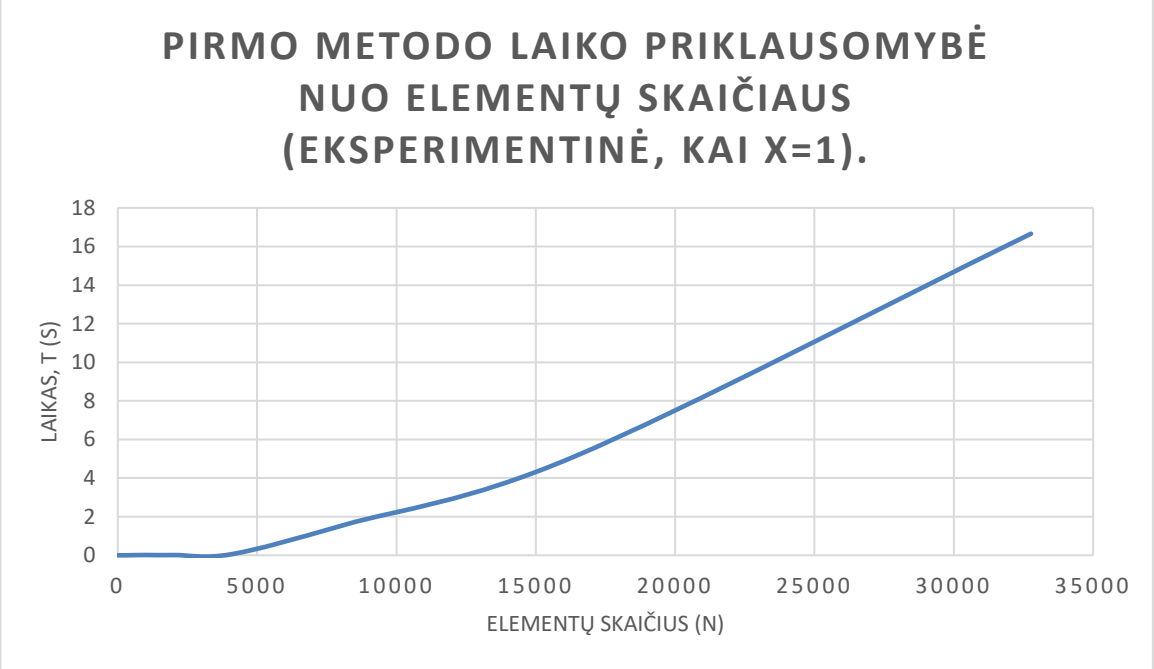
Šis atvejis mums tinka ir galime nustatyti, kad mūsų rekursinės lygties asimptotinis sudėtingumas yra lygus $FF4(n) \approx n^{1{,}7}$. Blogiausias šio metodo asimptotinis sudėtingumas. Tada mūsų bendras asimptotinis sudėtingumas būtų

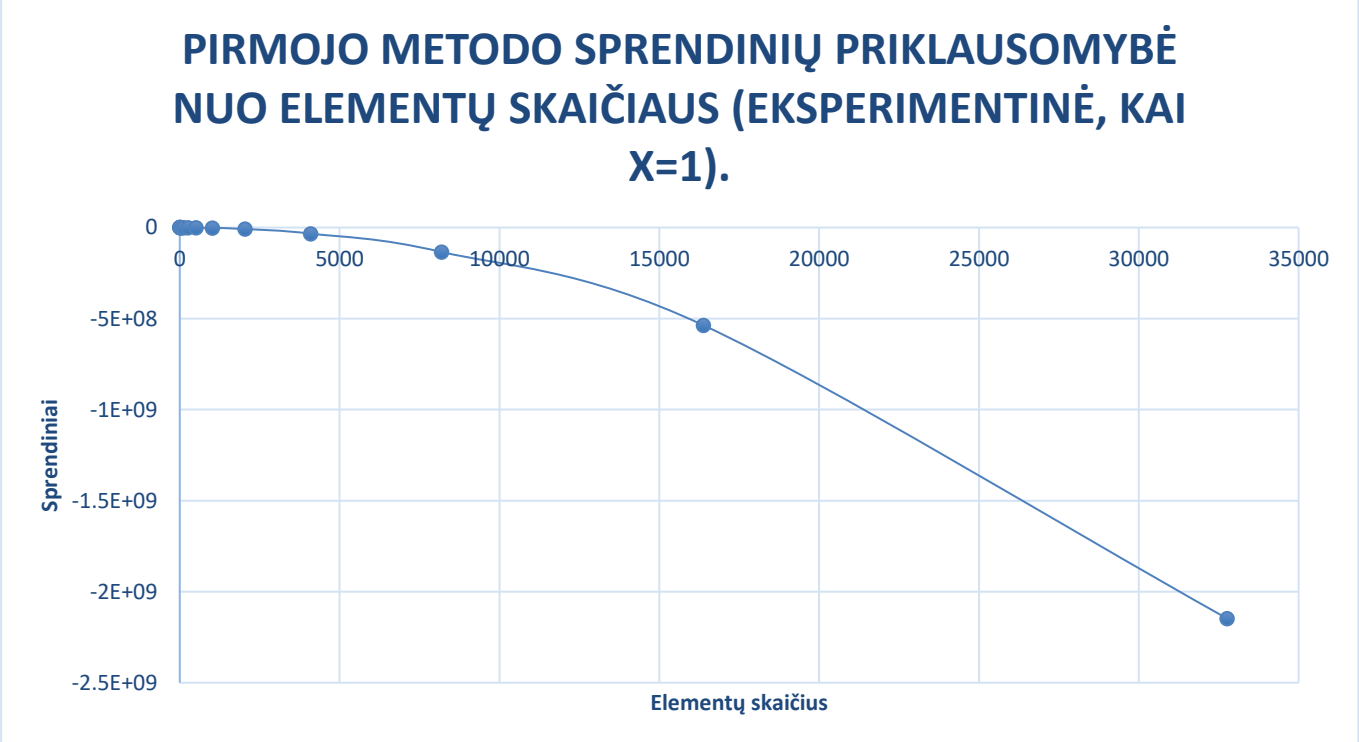
$T(n) = (n^{1{,}7} * n) + n$, *jeigu x = 1*

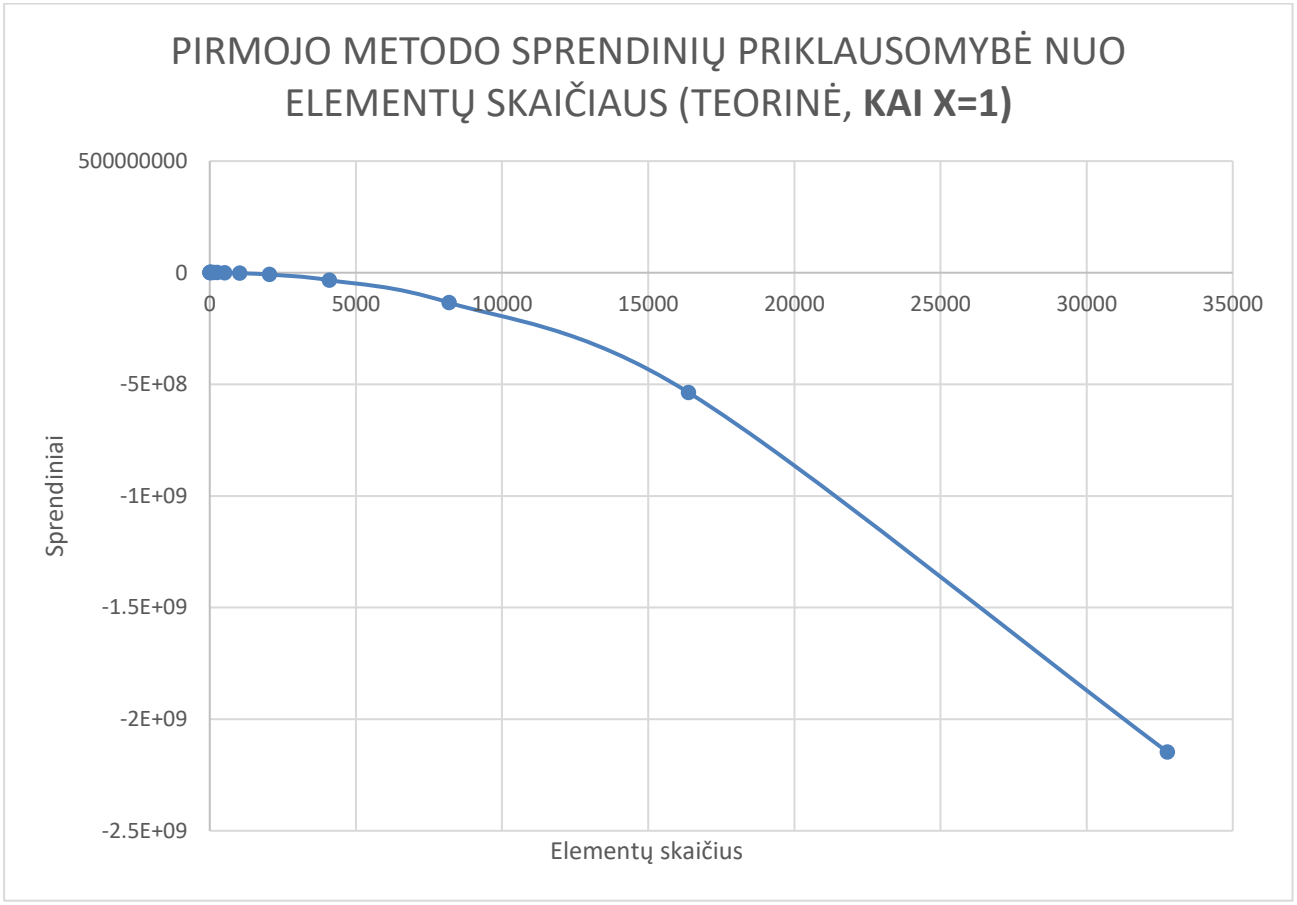
- Jeigu $log_b(a) = f(n)$, t.y. antrasis atvejis, šis atvejis mums netinka kadangi f(n) nėra lygus $log_b(a)$

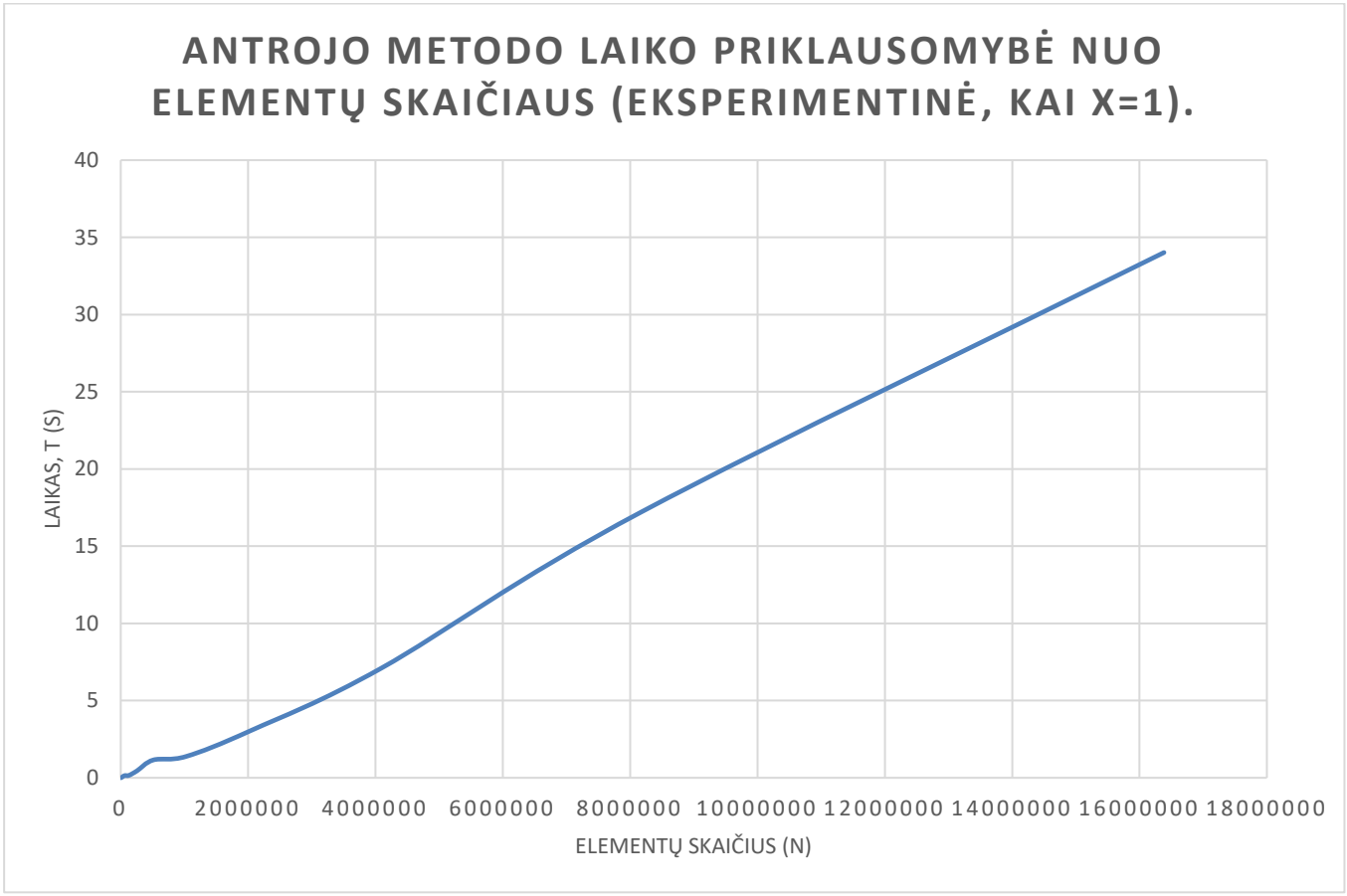
- Jeigu $log_b(a) < f(n)$, t.y. trečiasis atvejis, šis atvejis taipat mums netinka, kadangi f(n) nėra didesnis už $log_b(a)$

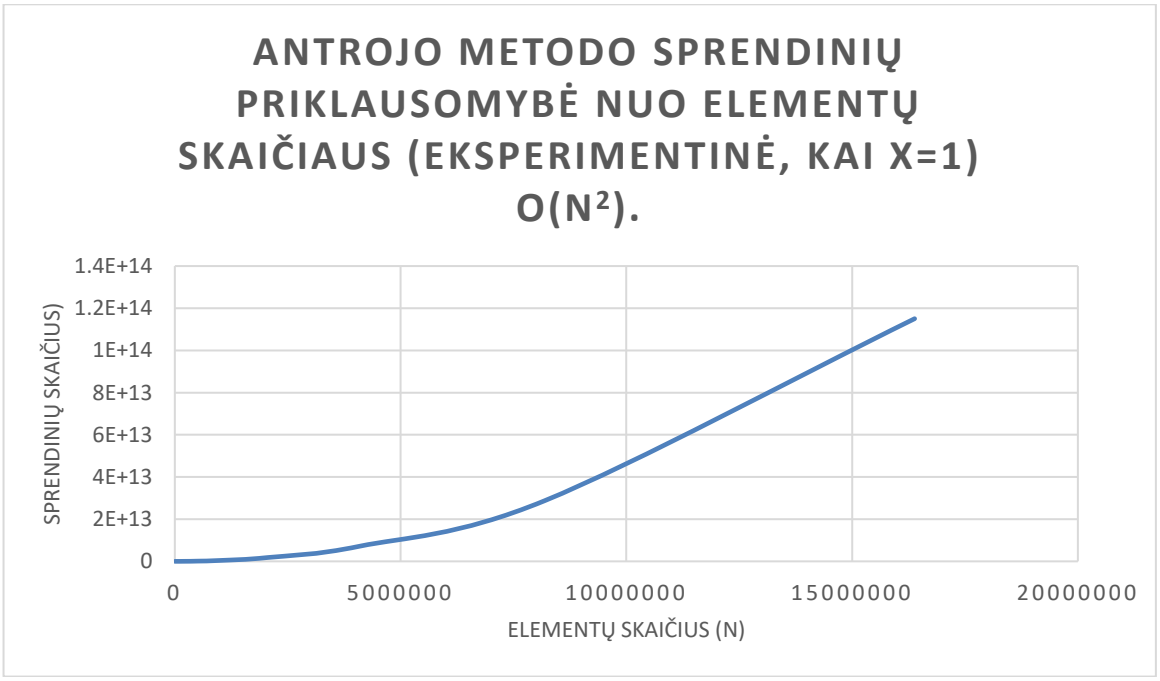
Atliekame našumo testus (pirmajam ir antrajam metodams).

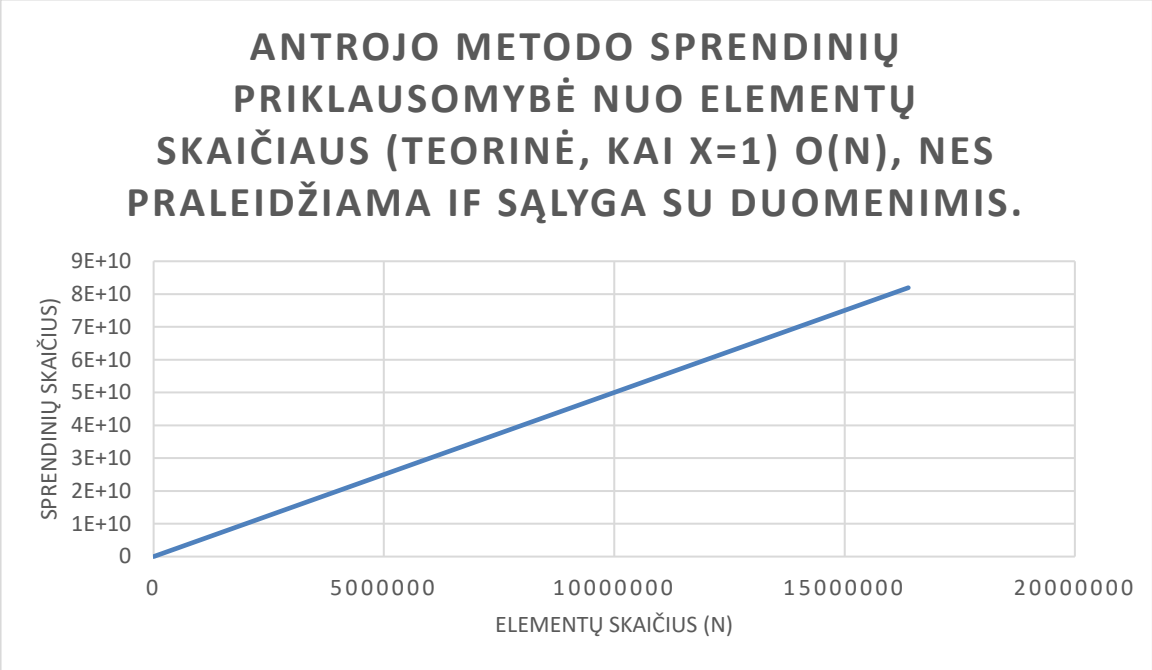


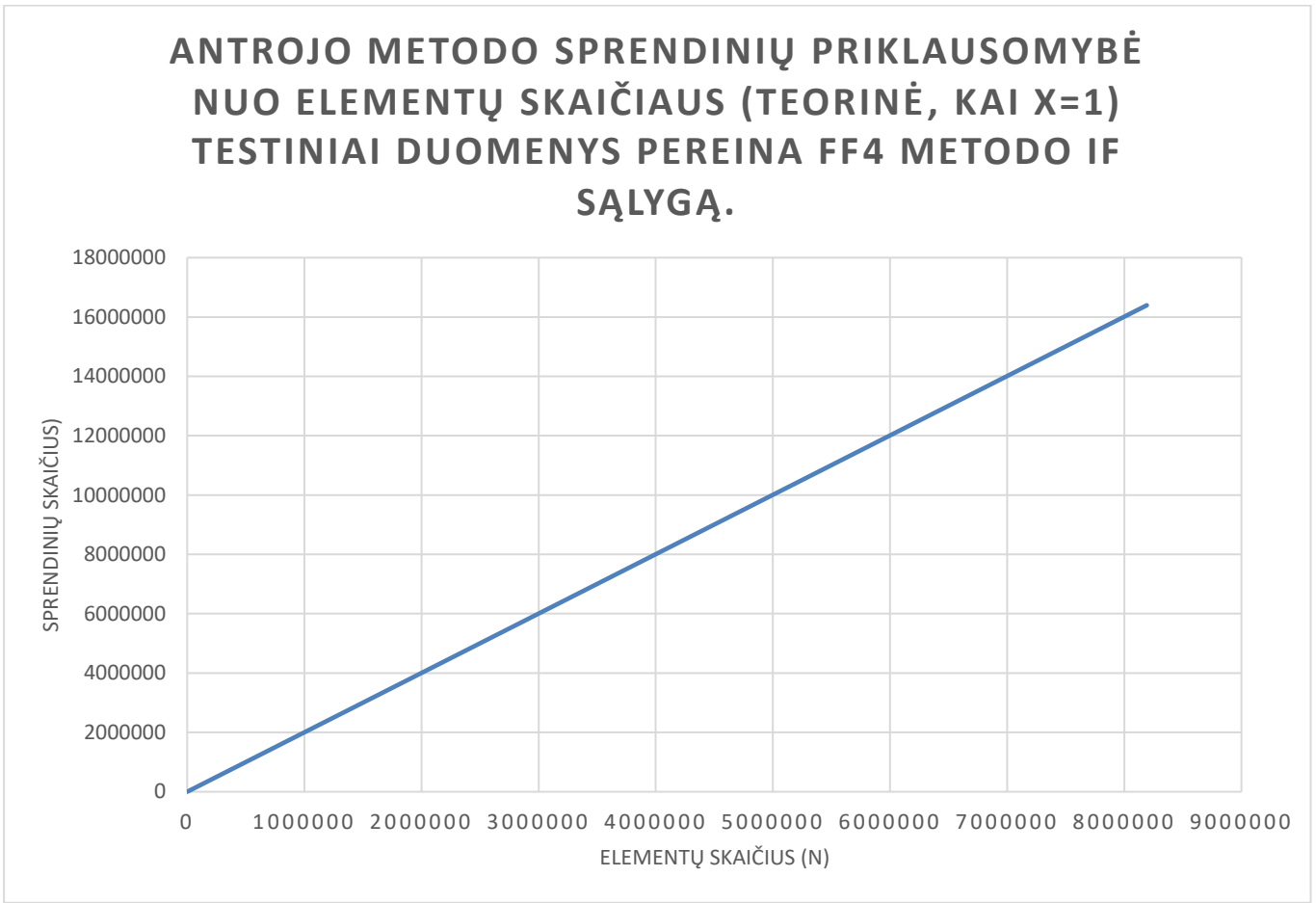



PIRMOJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ, **KAI X=1)**
500000000
0
-5E+08
-1E+09
-1.5E+09
-2E+09
-2.5E+09
Sprendiniai
0
5000
10000
15000
20000
25000
30000
35000
Elementų skaičius


ANTROJO METODO LAIKO PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (EKSPERIMENTINĖ, KAI X=1).
LAIKAS, T (S)
40
35
30
25
20
15
10
5
0
0
2000000
4000000
6000000
8000000
10000000
12000000
14000000
16000000
18000000
ELEMENTŲ SKAIČIUS (N)


ANTROJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (EKSPERIMENTINĖ, KAI X=1) O(N²).
SPRENDINIŲ SKAIČIUS)
1.4E+14
1.2E+14
1E+14
8E+13
6E+13
4E+13
2E+13
0
0
5000000
10000000
15000000
20000000
ELEMENTŲ SKAIČIUS (N)


ANTROJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ, KAI X=1) O(N), NES PRALEIDŽIAMA IF SĄLYGA SU DUOMENIMIS.
SPRENDINIŲ SKAIČIUS)
9E+10
8E+10
7E+10
6E+10
5E+10
4E+10
3E+10
2E+10
1E+10
0
0
5000000
10000000
15000000
20000000
ELEMENTŲ SKAIČIUS (N)


ANTROJO METODO SPRENDINIŲ PRIKLAUSOMYBĖ NUO ELEMENTŲ SKAIČIAUS (TEORINĖ, KAI X=1) TESTINIAI DUOMENYS PEREINA FF4 METODO IF SĄLYGĄ.
SPRENDINIŲ SKAIČIUS)
18000000
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
ELEMENTŲ SKAIČIUS (N)

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 peršokti 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 peršokti 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.

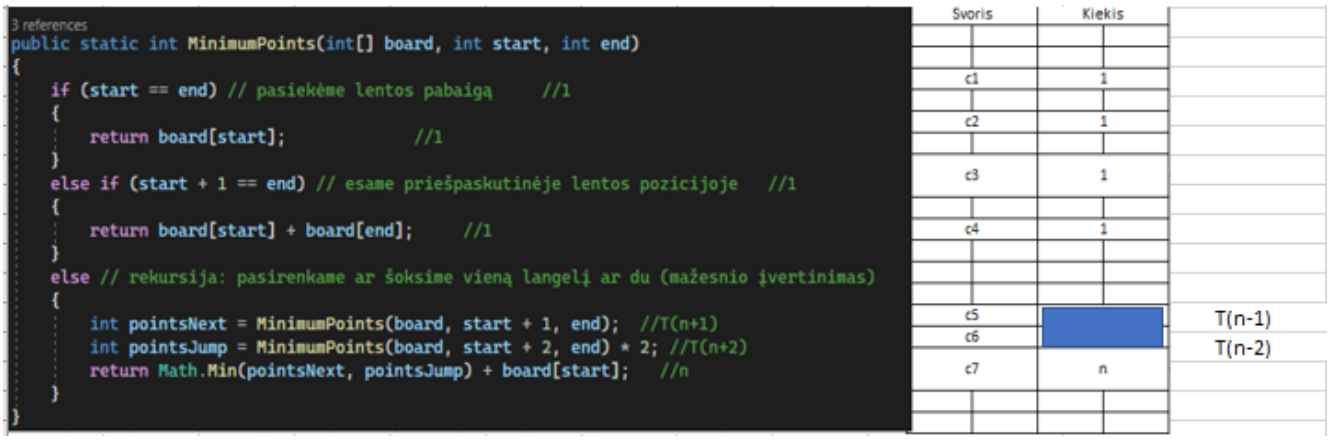
Dinaminio programavimo uždavinio sprendimas (dinaminio programavimo metodologija) :

```
public static int MinimumPoints(int[] board, int start, int end)
        {
           int[] minPoints = new int[end + 1]; //sukuriame masyva taskams kaupti
           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]; //gražiname mažiausią taškų sumą
}
```

Kitaip negu naudojant rekursiją, šiuos problemos sprendimui naudojame dinaminio programavimo metodologija, t.y. vengiame pasikartojančių veiksmų, rezultatų skaičiavimo. Todėl įsivedame masyvą minPoints (mažiausiai taškų sumai kaupti) 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:



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 :

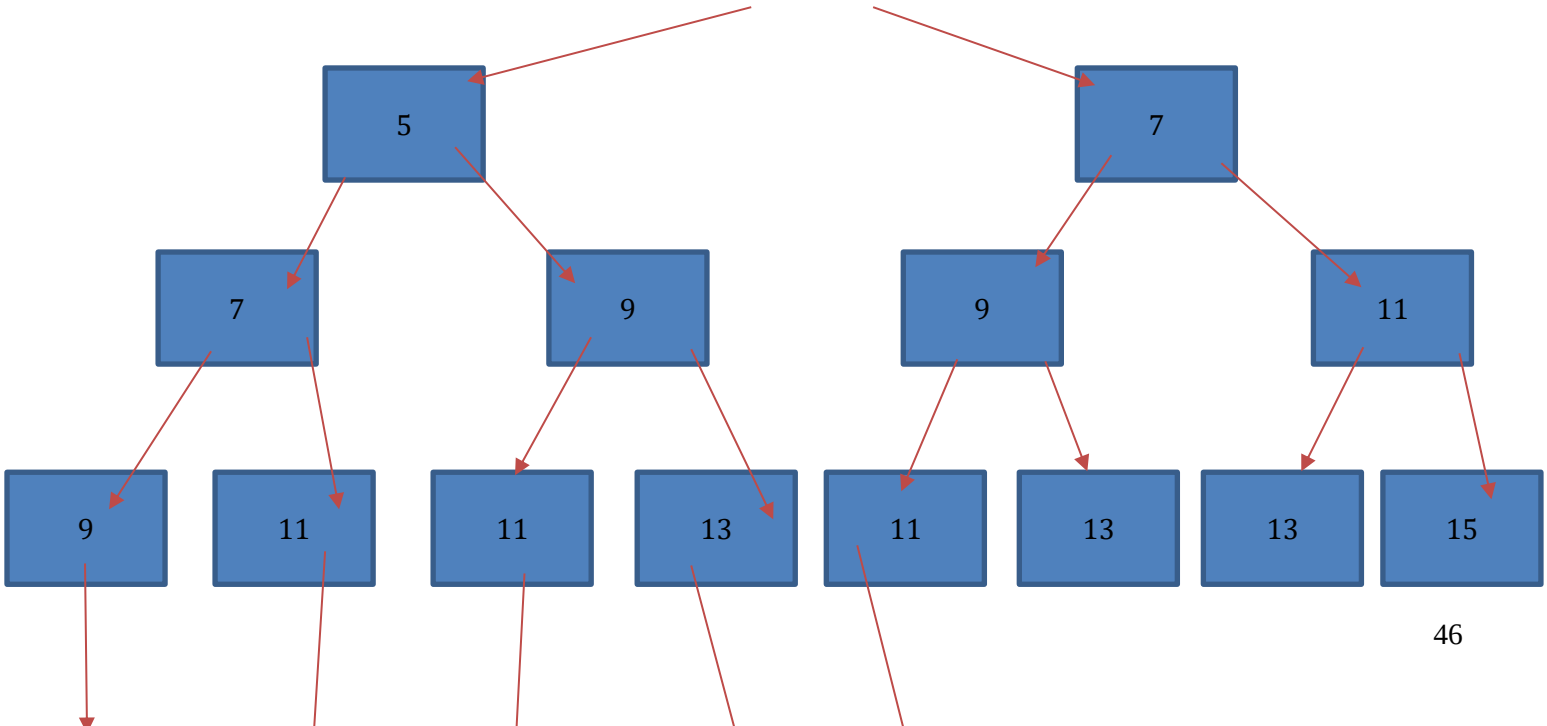
$T(n) = (n-1) + T(n-2) + n$ *t.y. blogiausias mūsų metodo asimptotinis sudėtingumas*

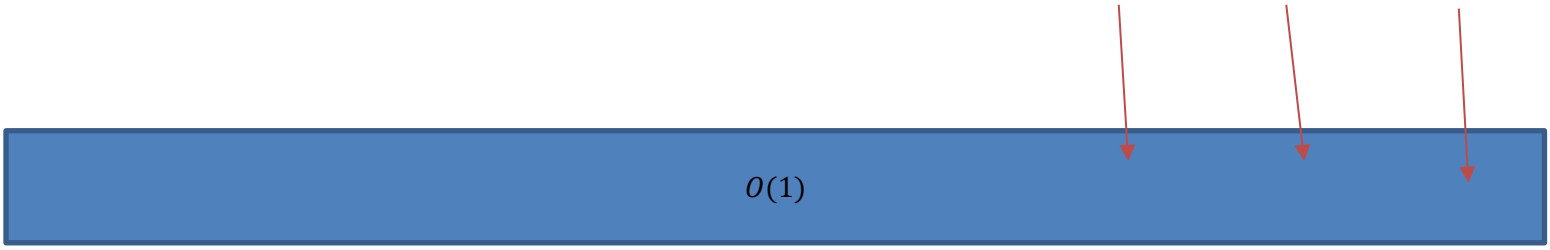
Pats geriausias metodo asimptotinis sudėtingumas bus :

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

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 $h = [\frac{n}{3}]$;

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

Dinaminio programavimo algoritmo kodo analizė:

```
public static int MinimumPoints(int[] board, int start, int end)
{
    int[] minPoints = new int[end + 1]; //sukuriame masyva taskams kaupti
    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]; //gražiname mažiausią taškų sumą
}
```

Svoris	Kiekis
c1	n
c2	1
c3;c4;c5	1;n+1;n
c6	1
c7	1
c8	n
c9	n
c10	n
c11	1

13 pav. Dinaminio kodo analizė.

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

$$c_i - konstantos, kurios\ yra\ O(1);$$

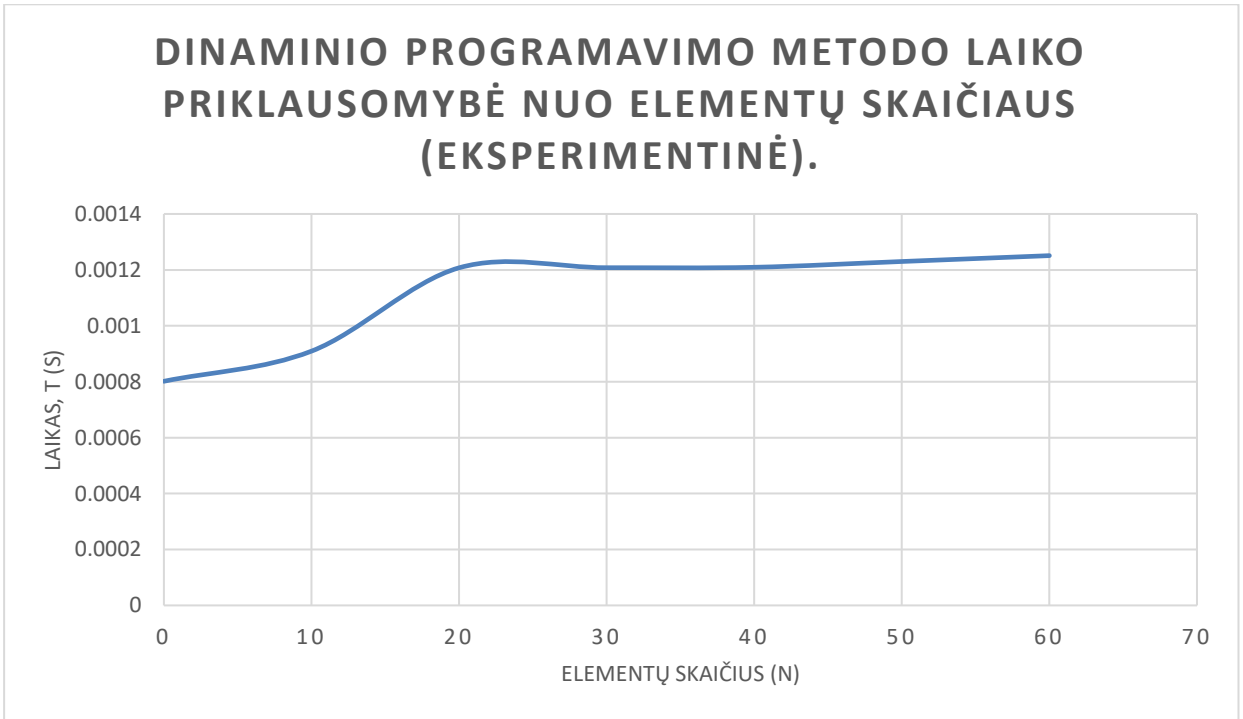
$$T(n) = c_1 n + c_2 + c_3 + c_4(n+1) + c_5(n) + c_6(n) + c_7(n) + c_8(n) + c_9(n) + c_{10}(n) + c_{11}$$

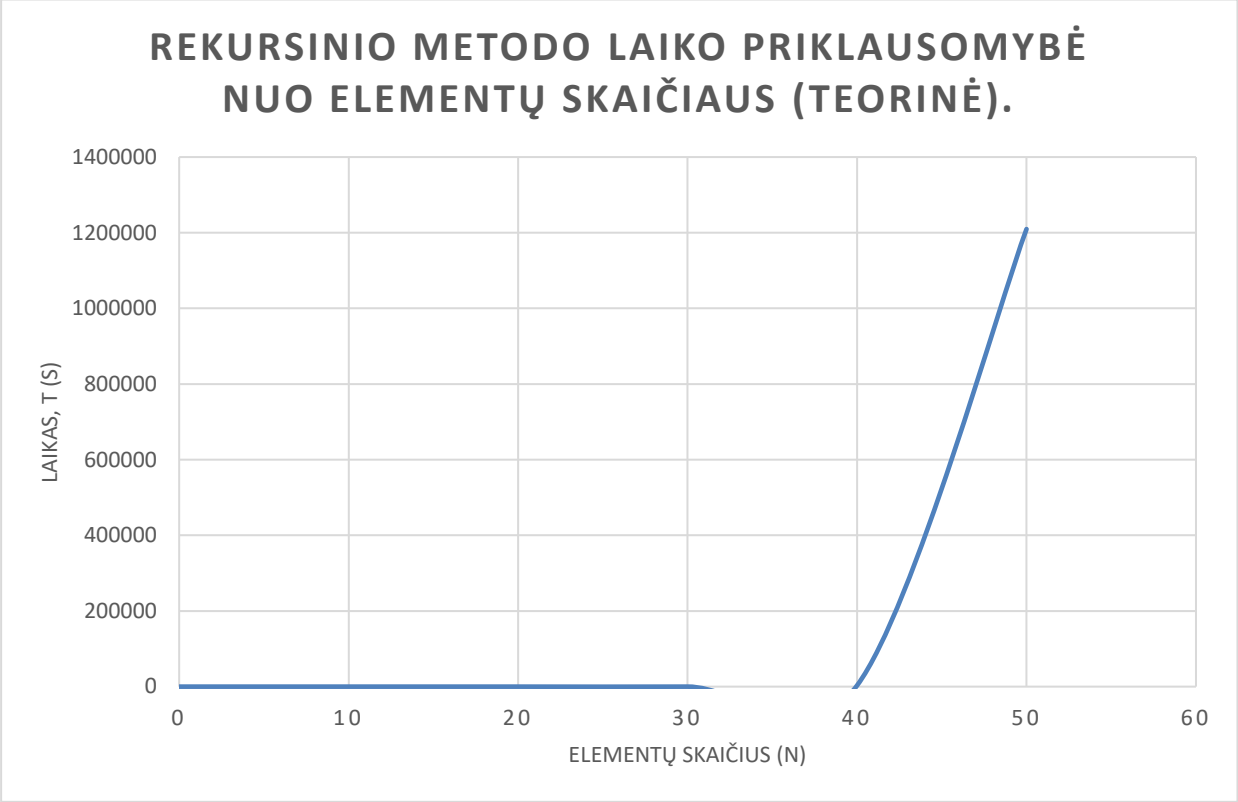
Suprastiname konstantas ir gauname, kad blogiausiu ir geriausiu atveju mūsų metodo asimptotinis sudėtingumas yra lygus (jeigu duomenys yra teisingi)

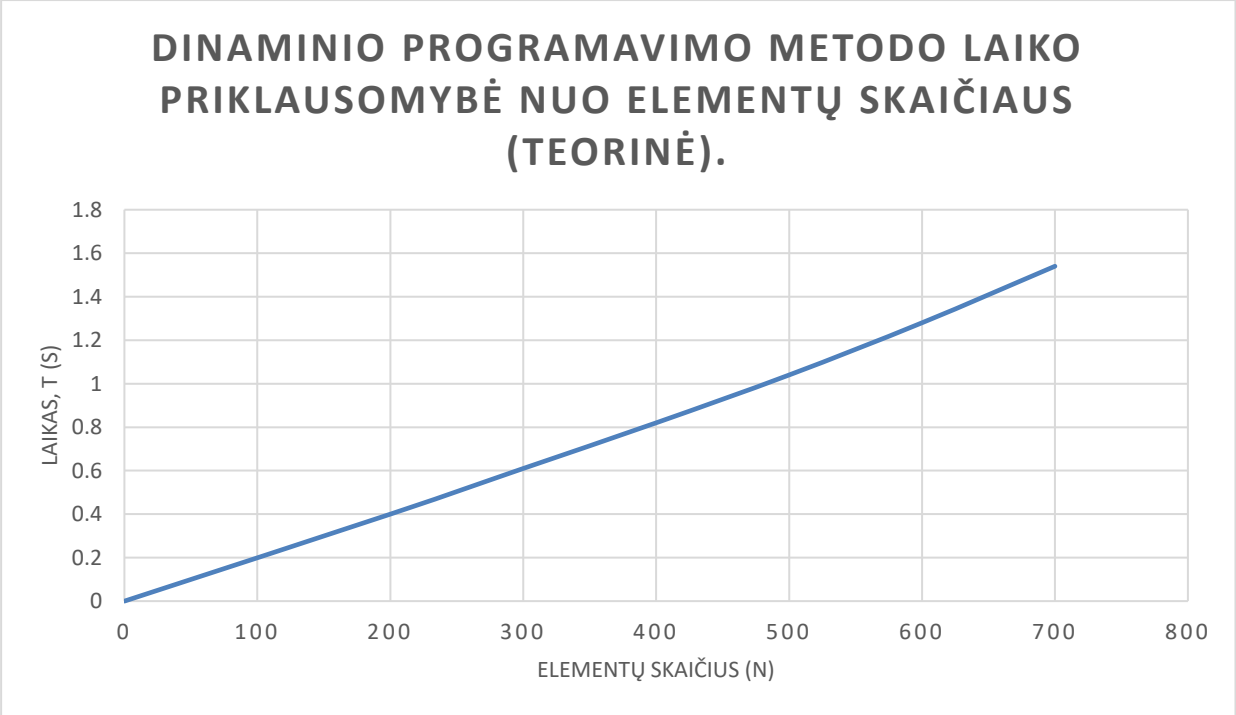
$$T(n) = n$$

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

Našumo analizė :







2.3. Šaltiniai

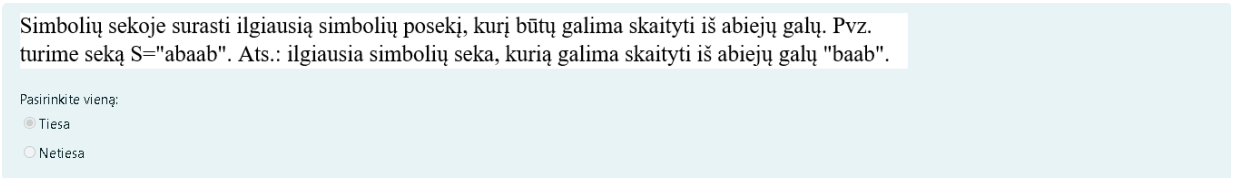
https://moodle.ktu.edu/course/view.php?id=2470

3. 3 LD laboratorinis darbas

3.1. Pradinė užduotis

1 užduoties dalis (4 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.



7 pav. Pirma uždavinio dalis

2 užduotis (6 balai):

- Atlikite pateiktų procedūrų lygiagretinimą.
- Įvertinkite teorinį nelygiagretintų ir lygiagretintų procedūrų sudėtingumą.
- 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.
- *2 uždavinys - 3 balai / 3 uždavinys - 3 balai*

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

Pasirinkite vieną:

- Tiesa
- Netiesa

```
public static long methodToAnalysis (int n, int[] arr)
{
    long k = 0;
    for (int i = 0; i < n; i++)
    {
        k += k;
        k += FF7(i, arr);
    }
    k += FF7(n, arr);
    return k;
}

public static long FF7(int n, int[] arr)
{
    if (n > 0 && arr.Length > n && arr[0] > 0)
    {
        return FF7(n - 2, arr) + FF7(n - 1, arr);
    }
    return n;
}
```

Pasirinkite vieną:

- Tiesa
- Netiesa

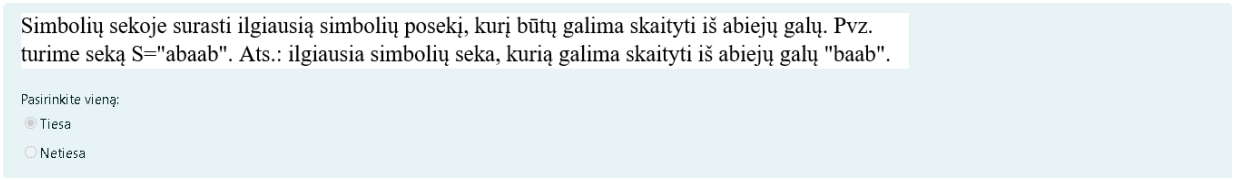
8 pav. Antra uždavinio dalis.

3.2. Užduoties sprendimas ir rezultatai

3.2.1 Pirmos dalies sprendimas

1 užduoties dalis (4 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.



7 pav. Pirma uždavinio dalis

Uždavinio sprendimas

Ieškome ilgiausios simbolių sekos, kurią galime skaityti iš abiejų galų. Pradžioje taikome rekursinių sąryšių sprendimą ir vėliau dinaminio programavimo sprendimą, palyginame abiejų sprendimų rezultatus.

Užduoties paaiškinimas:

Naudodami rekursiją, pereiname per mūsų tekstą ir kviesdami rekursiją mažėdama tvarka, gauname didžiausią raidžių seką, kurią galime skaityti iš abiejų pusių.

Naudodami dinaminį programavimą, suskirstome uždavinį į mažesnes dalis ir prisimename praeitus sprendimus, todėl pageriname mūsų uždavinio asimptotinį sudėtingumą.

Rekursinis uždavinio sprendimas (naudojant rekursinius sąryšius, rekursija) :

```
public static string LongestPalindrome(string s)
            {
                if (string.IsNullOrEmpty(s))
                {
                    return "";
                }

                if (s.Length == 1)
                {
                    return s;
                }

                if (s[0] == s[s.Length - 1])
                {
                    string subPalindrome = LongestPalindrome(s.Substring(1, s.Length -
2));
                    return s[0] + subPalindrome + s[s.Length - 1];
                }
                else
                {
                    string palindrome1 = LongestPalindrome(s.Substring(0, s.Length - 1));
                    string palindrome2 = LongestPalindrome(s.Substring(1));
                    return palindrome1.Length > palindrome2.Length ? palindrome1 :
palindrome2;
                }
            }
```

Šio metodo asimptotinis sudėtingumas: $O(2^n)$

Dinaminio programavimo uždavinio sprendimas (dinaminio programavimo metodologija) :

```
public static string LongestPalindrome(string s)
            {
                if (string.IsNullOrEmpty(s))
                {
                    return "";
                }

                int n = s.Length;
                bool[,] dp = new bool[n, n];

                // base case: single character strings are palindromes
                for (int i = 0; i < n; i++)
                {
                    dp[i, i] = true;
                }

                int maxLength = 1;
                int start = 0;

                // check substrings of length 2 or more
                for (int len = 2; len <= n; len++)
                {
                    for (int i = 0; i < n - len + 1; i++)
                    {
                        int j = i + len - 1;

                        if (s[i] == s[j] && (len == 2 || dp[i + 1, j - 1]))
                        {
                            dp[i, j] = true;

                            if (len > maxLength)
                            {
                                maxLength = len;
                                start = i;
                            }
                        }
                    }
                }

                return s.Substring(start, maxLength);
            }
```

Šio metodo asimptotinis sudėtingumas yra $O(n^2)$. Jis yra žymiai geresnis už rekursinį variantą.

Rekursinio algoritmo kodo analizė:

```
public static string LongestPalindrome(string s)
{
    if (string.IsNullOrEmpty(s))
    {
        return "";
    }

    if (s.Length == 1)
    {
        return s;
    }

    if (s[0] == s[s.Length - 1])
    {
        string subPalindrome = LongestPalindrome(s.Substring(1, s.Length - 2));
        return s[0] + subPalindrome + s[s.Length - 1];
    }
    else
    {
        string palindrome1 = LongestPalindrome(s.Substring(0, s.Length - 1));
        string palindrome2 = LongestPalindrome(s.Substring(1));
        return palindrome1.Length > palindrome2.Length ? palindrome1 : palindrome2;
    }
}
```

Laikas	Kartai	T(n), x=n.length - LongestPalindrome procedūros sudėtingumas
c2 + (1-n)c1,		čia, $n = \begin{cases} 0, kai\ x = 0 \\ 1, kai\ x > 0 \end{cases}$
c4+(1-x)c3		čia, $x = \begin{cases} 0, kai\ x = 1 \\ 1, kai\ x > 1 \end{cases}$
c5+(1-g)*T(n-2)+(1-n)c6		čia, $g = \begin{cases} 0, kai\ s[0] \,!= s[x-1] \\ 1, \ kai\ s[0] == s[x-1] \end{cases}$
T(n-1)	1	
T(1)	1	čia, kitais atvejais
c7	1	

12 pav. Rekursinio kodo analizė.

Turime du atvejus (Geriausią ir blogiausią).

Geriausias atvejis, kai mūsų (n = 0, arba x = 0), tada grąžiname mūsų gautą reikšmę.

$$c_i - konstantos, kurios\ yra\ O(1);$$

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

Gauname, kad mūsų asimptotinis sudėtingumas, geriausių atveju yra O(1).

Blogiausias yra, kai mūsų (n = 1, x =1), tada pereiname prie mūsų g, kuris gali būti arba 0 arba 1. Visais atvejais vistiek turėsime spręsti rekurentinę lygtį.

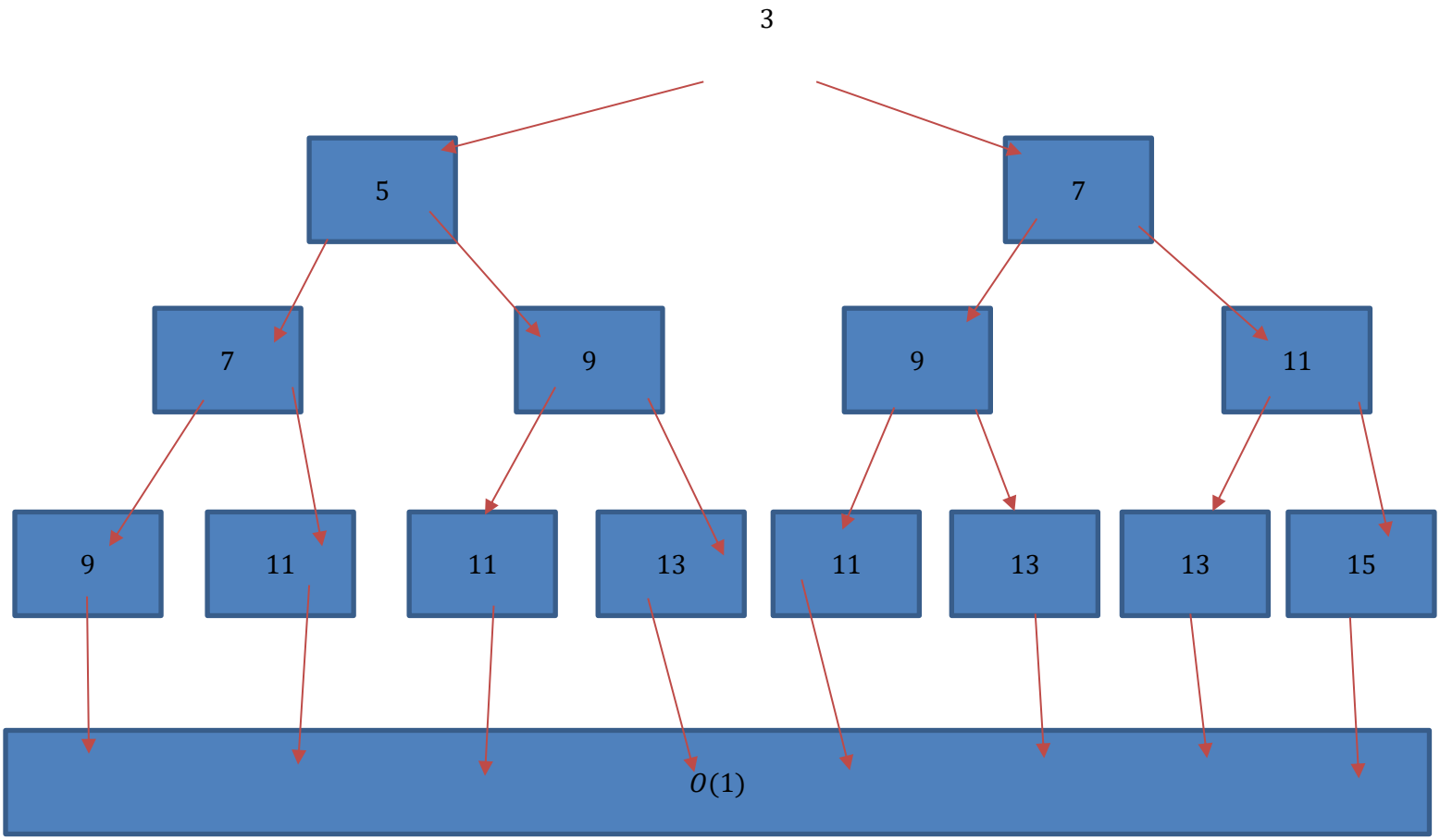
$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + T(n-2) + c_6 + T(n-1) + T(1) + c_7$$

Suprastiname visas konstantas ir gauname rekursinių sąryšių lygtį :

$T(n) = (n-2) + T(n-1) + T(1)$ *t.y. blogiausias mūsų metodo asimptotinis sudėtingumas, T(1) smulki konstanta, todėl galime ją atmesti.*

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 $h = [\frac{n}{3}]$;

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

Negavome asimptotiškai tikslaus įvertinimo. Tačiau blogiausias šios lygties asimptotinis sudėtingumas yra $O(2^n)$

Dinaminio programavimo algoritmo kodo analizė:

```
1 reference
public static string LongestPalindrome(string s)
{
    if (string.IsNullOrEmpty(s))
    {
        return "";
    }
    int n = s.Length; bool[,] dp = new bool[n, n];
    // base case: single character strings are palindromes
    for (int i = 0; i < n; i++)
    {
        dp[i, i] = true;
    }
    int maxLength = 1;int start = 0;
    // check substrings of length 2 or more
    for (int len = 2; len <= n; len++)
    {
        for (int i = 0; i < n - len + 1; i++)
        {
            int j = i + len - 1;
            if (s[i] == s[j] && (len == 2 || dp[i + 1, j - 1]))
            {
                dp[i, j] = true;
                if (len > maxLength)
                {
                    maxLength = len;
                    start = i;
                }
            }
        }
    }
    return s.Substring(start, maxLength);
```

Kaina	Kiekis	T(n), x=n.length - LongestPalindrome procedūros sudėtingumas
		len = 2; maxLength = 1; i = 0; j = i + len - 1;
c2+(1-n)c1		čia, $n = \begin{cases} 0, jei\ x = 0 \\ 1, jei\ x > 0 \end{cases}$
c3;c4	1;1	
c5;c6;c7	1;n+1;n	
c8	1	
c9	1	
c10;c11;c12	1;n+1;n	
c13;c14;c15	n;n^2+1;n^2	
c16	n^2	
(1-m)*(c17*n^2)		čia, $m = \begin{cases} 0, jei\ n[i] = n[j]\ ir\ len = 2\ arba\ dp[i+1, j-1] \\ 1, kitais\ atvejais \end{cases}$
c18	(1-m)n^2	
(1-l)*(c19*n^2)		čia, $l = \begin{cases} 0, jei\ len > maxLength \\ 1, kitais\ atvejais \end{cases}$
c20	(2-m-l)n^2	
c21	(2-m-l)n^2	
c22	1	

13 pav. Dinaminio kodo analizė.

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

$$c_i - konstantos, kurios\ yra\ O(1);$$

Geriausias atvejis, kai mūsų n=0, tada

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

T.y. geriausiu atveju mūsų metodo asimptotinis sudėtingumas yra O(1).

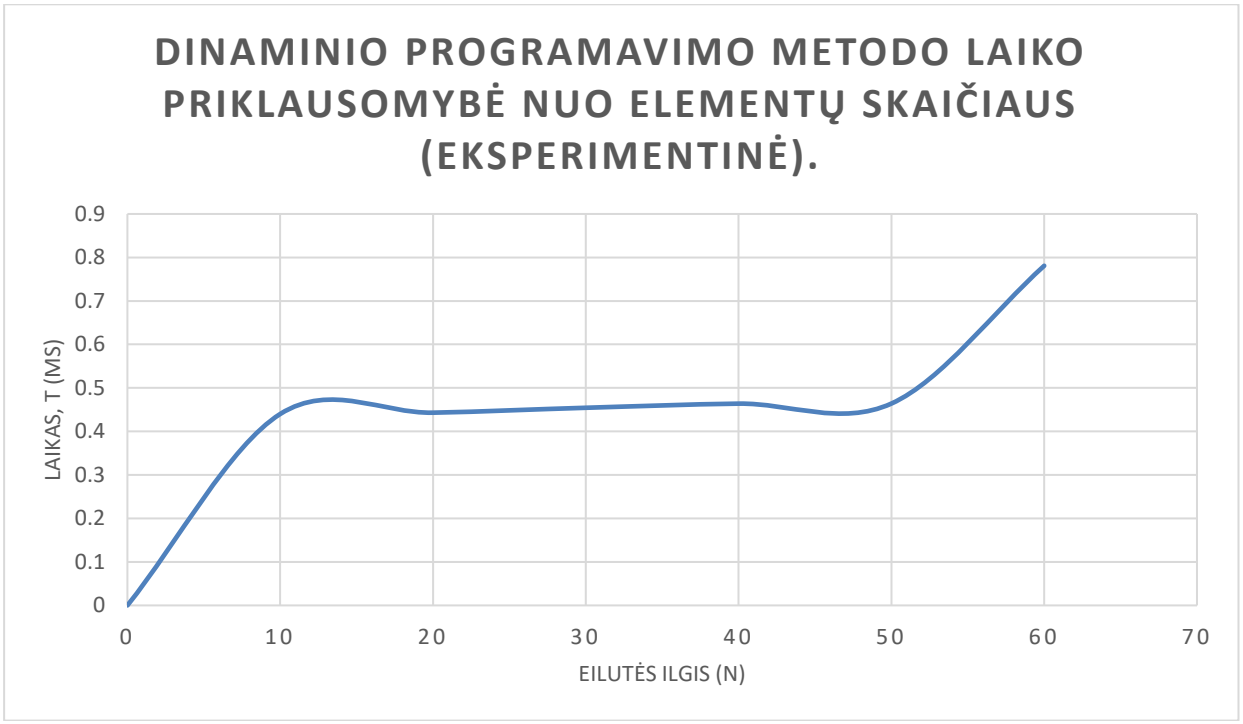
Blogiausias atvejis, kai mūsų n=1, m=0, l=0, tada gauname

$$\begin{aligned} T(n) = {} & c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7(n) + c_8 + c_9 + c_{10} + c_{11}(n+1) + c_{12}(n) \\ & + c_{13}(n) + c_{14}(n^2+1) + c_{15}(n^2) + c_{16}(n^2) + c_{17}(n^2) + c_{18}(n^2) + c_{19}(n^2) \\ & + c_{20}(n^2) + c_{21}(n^2) + c_{22} \end{aligned}$$

Suprastiname konstantas ir gauname, kad blogiausiu mūsų metodo asimptotinis sudėtingumas yra lygus (jeigu duomenys yra teisingi). (Geriausiu atveju mūsų metodo asimptotinis sudėtingumas yra O(1), jeigu yra paduotas tuščias duomenų stringas arba iš vienos raidės).

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

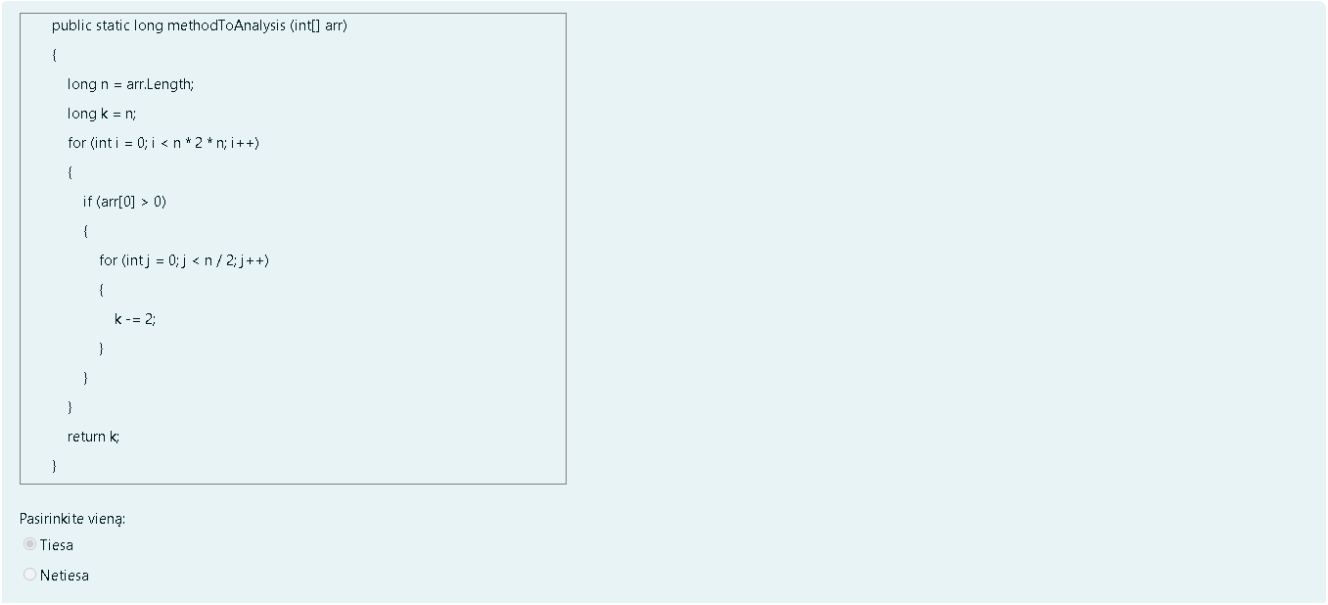
Našumo analizė :



3.2.2 Antros dalies sprendimas

- Atlikite pateiktų procedūrų lygiagretinimą.
- Įvertinkite teorinį nelygiagretintų ir lygiagretintų procedūrų sudėtingumą.
- 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.
- *2 uždavinys - 3 balai / 3 uždavinys - 3 balai*

Atliekame duotų procedūrų lygiagretinimą (methodToAnalysis ir methodToAnalysis1)



13 pav. Pirmosios procedūros pavyzdys.

```
public static long methodToAnalysisParallel(int[] arr)
        {
            long n = arr.Length;
            long k = n;
            object monitor = new object();
            if (arr[0] > 0)
            {
                Parallel.For(0, n * 2 * n, () => 0,
                    (i, state, local_sum_of_minus_two) =>
            {

                for (int j = 0; j < n / 2; j++)
                {
                    local_sum_of_minus_two -= 2;
                }
                return local_sum_of_minus_two;

            }, (total_sum) => { lock (monitor) k += total_sum; });
            }
            return k;
        }
```

14 pav. Antrosios procedūros pavyzdys.

```
public static long methodToAnalysis1Parallel(int n, int[] arr)
        {
            long k = 0;
            object lockObject = new object();
            Parallel.For(0, n, i => {
                long localK = k;
                localK += localK;
                localK += FF7(i, arr);
                lock (lockObject)
                {
                    k = localK;
                }
            });
            k += FF7(n, arr);
            return k;
        }
        public static long FF7(int n, int[] arr)

        {

            if (n > 0 && arr.Length > n && arr[0] > 0)

            {

                return FF7(n - 2, arr) + FF7(n - 1, arr);

            }

            return n;

        }
```

Atliekame duotų metodų kodų analizę

```
public static long methodToAnalysis(int[] arr)
{
    long n = arr.Length;

    long k = n;

    for (int i = 0; i < n * 2 * n; i++)

    {

        if (arr[0] > 0)

        {

            for (int j = 0; j < n / 2; j++)

            {

                k -= 2;

            }

        }

    }

    return k;
}
```

Kaina	Kiekis	T(n) - methodToAnalysis procedūros sudėtingumas
c1	1	
c2	1	
c3;c4;c5	1;2n^2+1;n^2	
	(1-n)*c6	čia, $n = \begin{cases} 0, jei\ arr[0] > 0 \\ 1, kitais\ atvejais \end{cases}$
c7;c8;c9	(1-n)*n^2;(1-n)*n^3/2;(1-n)*n^3	
c10	(1-n)n^3	
c11	1	

15 pav. Pirmojo metodo (nelygiagretaus) kodo analizė.

Turime du atvejus, blogiausias ir geriausias. Blogiausias atvejis yra tada, kai mūsų arr[0] > 0, (t.y. n = 0), geriausias atvejis yra tada, kai mūsų (n=1).

$$c_i - konstantos, kurios\ yra\ O(1);$$

Apskaičiuojame šiuos svorius ir įvertiname asimptotinį sudėtingumą.

Geriausiu atveju nepereiname if sąlygos ir einame tik per vieną ciklą.

Geriausias atvejis (n=1) :

$$T(n) = c_1 + c_2 + c_3 + \ c_4 * (2n^2) + \ c_5 * (n^2) + \ (1-1)c_6 + c_{11} = O(n^2);$$

Tai geriausiu atveju mūsų nelygiagretaus metodo asimptotinis sudėtingumas yra lygus

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

Blogiausias šio nelygiagretaus metodo asimptotinis sudėtingumas, kai mūsų (n=0, arr[0] > 0).

$$T(n) = c_1 + c_2 + c_3 + c_4 * (2n^2) + c_5 * (n^2) + (1-0)c_6 + (1-0)c_7 + (1-0) * (n^2 * c_8) + (1-0) * (\frac{n^3}{2} c_9) + (1-0) * (n^3 * c_{10}) + c_{11} = O(n^3);$$

Suprastiname visą funkciją ir gauname:

$$T(n) = n^3;$$

Tai, blogiausias mūsų nelygiagretaus metodo asimptotinis sudėtingumas yra:

$$T(n) = O(n^3);$$

Atliekame kodo analizę lygiagrečiam šio metodo pritaikymui.

```
public static long methodToAnalysisParallel(int[] arr)
{
    long n = arr.Length;
    long k = n;
    object monitor = new object();
    if (arr[0] > 0)
    {
        Parallel.For(0, n * 2 * n, () => 0,
            (i, state, local_sum_of_minus_two) =>
    {

        for (int j = 0; j < n / 2; j++)
        {
            local_sum_of_minus_two -= 2;
        }
        return local_sum_of_minus_two;

    }, (total_sum) => { lock (monitor) k += total_sum; });
    }
    return k;
}
```

Kaina	Kiekis	T(n) x-kintamasis - methodToAnalysis procedūros sudėtingumas
c1;c2;c3	1	
(1-x)*c4		čia, $x = \begin{cases} 0, jei\ arr[0] > 0 \\ 1, kitais\ atvejais \end{cases}$
(1-x)*(c5*1+c6*2*n^2+c7*n^2)		
c8;c9;c10	(1-x)*n^2;(1-x)*(n^3/2)+1;(1-x)*n^3	
c11	(1-x)*n^3	
c12	(1-x)*n^2	
c13	1	
c14	1	

16 pav. Pirmojo metodo (lygiagretaus) kodo analizė.

Turime du atvejus, blogiausias ir geriausias. Blogiausias atvejis yra tada, kai mūsų arr[0] > 0, (t.y. x = 0), geriausias atvejis yra tada, kai mūsų (x=1).

$$c_i - konstantos, kurios\ yra\ O(1);$$

Apskaičiuojame šiuos svorius ir įvertiname asimptotinį sudėtingumą.

Geriausiu atveju nepereiname if sąlygos.

Geriausias atvejis (x=1) :

$$T(n) = c_1 + c_2 + c_3 + (1-1) * c_4 + (1-1) * (c_5) + (1-1) * (2n^2 * c_6) + (1-1) * (n^2 * c_7) + (1-1) * (n^2 * c_8) + (1-1) * (\frac{n^3}{2} * c_9) + (1-1) * (n^3 * c_{10}) + (1-1) * (n^3 * c_{11}) + (1-1) * (n^2 * c_{12}) + (1-1) * (c_{13}) + c_{14} = O(1);$$

Tai geriausiu atveju mūsų lygiagretaus metodo asimptotinis sudėtingumas yra lygus

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

Blogiausias šio nelygiagretaus metodo asimptotinis sudėtingumas, kai mūsų (x=0, arr[0] > 0).

$$T(n) = c_1 + c_2 + c_3 + (1-0) * c_4 + (1-0) * (c_5) + (1-0) * (2n^2 * c_6) + (1-0) * (n^2 * c_7) + (1-0) * (n^2 * c_8) + (1-0) * (\frac{n^3}{2} * c_9) + (1-0) * (n^3 * c_{10}) + (1-0) * (n^3 * c_{11}) + (1-0) * (n^2 * c_{12}) + (1-0) * (c_{13}) + c_{14} = O(1);$$

Suprastiname visą funkciją ir gauname:

$$T(n) = n^3;$$

Tai, blogiausias mūsų lygiagretaus metodo asimptotinis sudėtingumas yra (tačiau bendra užduotis atliekama greičiau, nes naudojamos gyjos ir darbas paskirtomas per kelias gyjas, todėl užduoties atlikimo laikas patrumpėja – priklausomai nuo kompiuterio bendrų resursų).

$$T(n) = O(n^3);$$

Atliekame antrojo metodo methodToAnalysis1 kodo analizę.

```
public static long methodToAnalysis1(int n, int[] arr)
{
    long k = 0;
    for (int i = 0; i < n; i++)

    {
        k += k;
        k += FF7(i, arr);
    }
    k += FF7(n, arr);
    return k;
}

5 references
public static long FF7(int n, int[] arr)
{
    if (n > 0 && arr.Length > n && arr[0] > 0)

    {
        return FF7(n - 2, arr) + FF7(n - 1, arr);
    }

    return n;
}
```

Kaina	Kiekis	T(n) x-kintamasis - methodToAnalysis1 procedūros sudėtingumas
c1	1	
c2;c3;c4	1;n+1;n	
c5	n	
$\sum_{i=0}^{n} FF7(i)$	n	
FF7(n);c6	1;1	
	(1-x)*(c7+c8+c9)	čia, $x = \begin{cases} 0, jei\ n > 0\ ir\ n.length > n\ ir\ arr[0] > 0 \\ 1, kitais\ atvejais \end{cases}$
c10	(1-x)*(FF7(n-2)+FF7(n-1))	
c11	1	

16 pav. Antrojo metodo (nelygiagretaus) kodo analizė.

Įsivedame kintamąjį x, kuris parodo ar mūsų if sąlyga buvo įvykdyta FF7 metode.

Po analizės susumuojame visus šiuos svorius ir apskaičiuojame rekurentinę lygtį.

Susumuojame visus methodToAnalysis1 procedūros kiekius ir sumas. Tada sprendžiame FF7 lygtį ir įsistatome gautas reikšmes į mūsų T(n) lygtį, taip surasdami šios funkcijos asimptotinį sudėtingumą.

$$T(n) = c_1 + c_2 + c_3(n+1) + c_4(n) + c_5(n) + \sum_i^n FF7(i) * n + \sum_n FF7(n) + c_6;$$

Blogiausias atvejis FF7

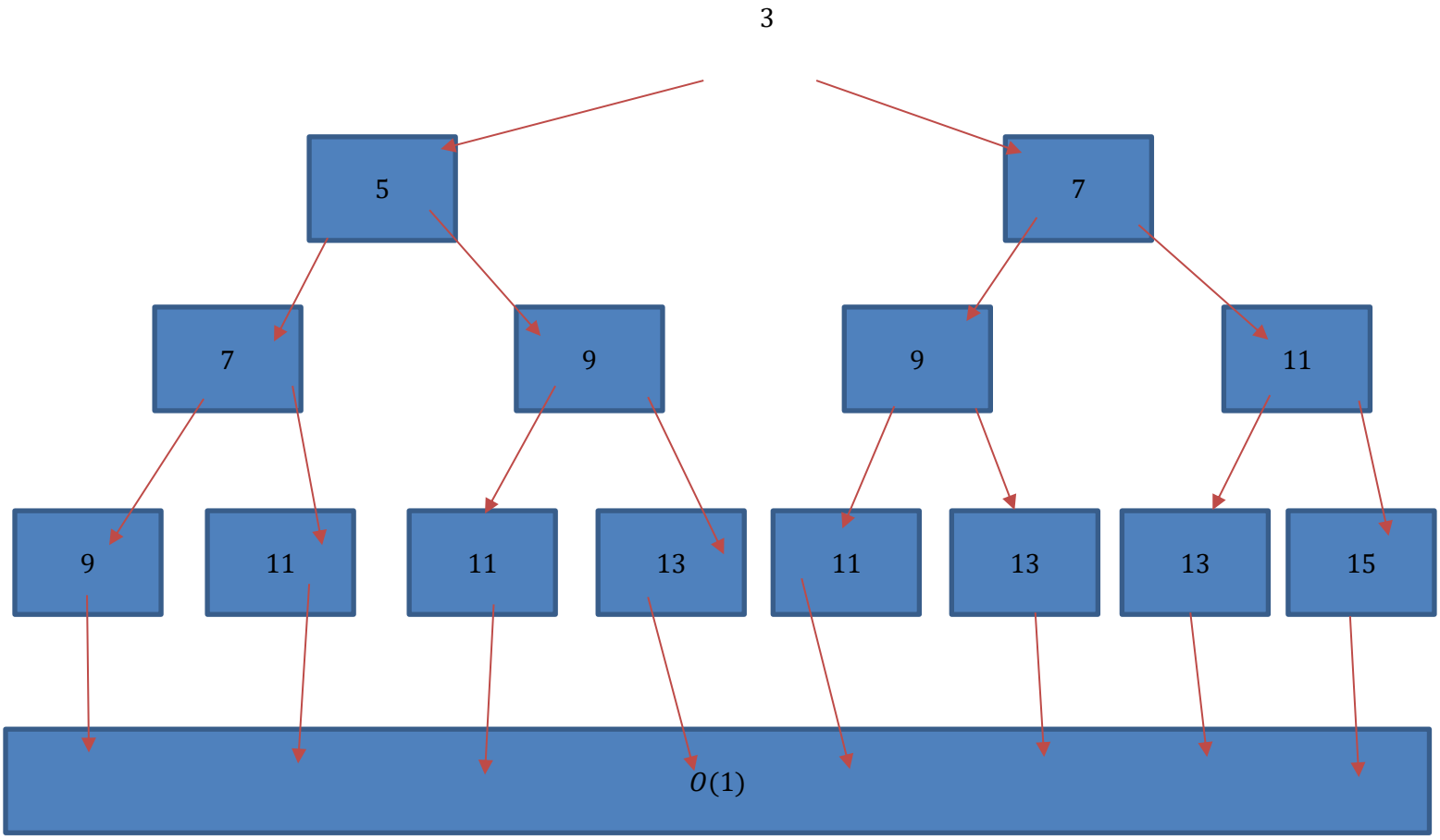
$$FF7(n) = (1-x)c_7 + (1-x)c_8 + (1-x)c_9 + \big((1-x)FF7(n-2) + (1-x)FF7(n-1)\big) + (1-x)c_{10} + c_{11}$$ *, kai mūsų x = 1, t.y. praeinama if sąlyga.*

Suprastinę viską, gauname rekurentinę lygtį

$$FF7(n) = FF7(n-2) + FF7(n-1)$$

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 $h = [\frac{n}{3}]$;

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

Negavome asimptotiškai tikslaus įvertinimo. Tačiau blogiausias šios lygties asimptotinis sudėtingumas yra $O(2^n)$

Taigi įstatome šia lygtį į mūsų T(n) ir gauname, kad blogiausiu atveju mūsų procedūros asimptotinis sudėtingumas

$$T(n) = c_1 + c_2 + c_3(n+1) + c_4(n) + c_5(n) + \sum_{i}^{n} FF7(i) * n + \sum_{n} FF7(n) + c_6 = 2^n * n + n;$$

Blogiausiu atveju, mūsų procedūros asimptotinis sudėtingumas yra

$$O(n2^n + n)$$

Geriausias atvejis FF7

$$FF7(n) = (1-x)c_7 + (1-x)c_8 + (1-x)c_9 + \big((1-x)FF7(n-2) + (1-x)FF7(n-1)\big) + (1-x)c_{10} + c_{11} = O(1), kai\ mūsų\ x = 0, t.y. nepraeinama\ if\ sąlyga$$

.

Taigi įstatome šia lygtį į mūsų T(n) ir gauname, kad geriausiu atveju mūsų procedūros asimptotinis sudėtingumas

$$T(n) = c_1 + c_2 + c_3(n+1) + c_4(n) + c_5(n) + \sum_{i}^{n} FF7(i) * n + \sum_{n} FF7(n) + c_6 = n + O(1) * n = n;$$

Geriausiu atveju, mūsų procedūros asimptotinis sudėtingumas yra

$$O(n)$$

Atliekame antrojo methodToAnalysis1Parallel procedūros kodo analizę.

ANTRASIS METODAS LYGIAGRETUS

```
public static long methodToAnalysis1Parallel(int n, int[] arr)
{
    long k = 0;
    object lockObject = new object();
    Parallel.For(0, n, i => {
        long localK = k;
        localK += localK;
        localK += FF7(i, arr);
        lock (lockObject)
        {
            k = localK;
        }
    });
    k += FF7(n, arr);
    return k;
}
public static long FF7(int n, int[] arr)
{
    if (n > 0 && arr.Length > n && arr[0] > 0)

    {
        return FF7(n - 2, arr) + FF7(n - 1, arr);
    }

    return n;
}
```

	Kaina	Kiekis	T(n) x-kintamasis - methodToAnalysis1 procedūros sudėtingumas
	c1;c2	1;1	
	c3;c4;c5	1;n;n	
	$\sum_{i=0}^{n} k$ $\sum_{i=0}^{n} FF7(i)$	n;n	
	c6	n	
	$\sum^{n} FF7(n)$;c7	1;1	
	(1-x)*(c7+c8+c9)	čia, $x = \begin{cases} 0, jei\ n > 0\ ir\ n.length > n\ ir\ arr[0] > 0 \\ 1, kitais\ atvejais \end{cases}$	
c10	(1-x)*(FF7(n-2)+FF7(n-1))		
c11	1		

17 pav. Antrojo metodo (lygiagretaus) kodo analizė.

FF7 procedūros metodas, toks pats, kaip ir nelygiagretaus.

Turime du atvejus, geriausias ir blogiausias. Geriausias, tada kai mūsų x = 0 (praeinama if sąlyga), o blogiausias, kai mūsų x=1 (nepraeinama if sąlyga).

$$T(n) = c_1 + c_2 + c_3(1) + c_4(n) + c_5(n) + \sum_i^n FF7(i) * n + \sum_i^n k * n + c_6 * n + \sum^n FF7(n) + c_7 ;$$

Geriausias

$FF7(n) = (1-x)c_7 + (1-x)c_8 + (1-x)c_9 + \big((1-x)FF7(n-2) + (1-x)FF7(n-1)\big) + (1-x)c_{10} + c_{11} = O(1)$, *kai mūsų x = 1, t.y. nepraeinama if sąlyga*

Įstatome į mūsų T(n),
$T(n) = c_1 + c_2 + c_3(1) + c_4(n) + c_5(n) + O(1) * n + c_6 * n + \sum^n FF7(n) + c_7$ = O(n);

Geriausiu atveju, mūsų procedūros asimptotinis sudėtingumas yra O(n).

Skaičiuojame blogiausią atvejį:

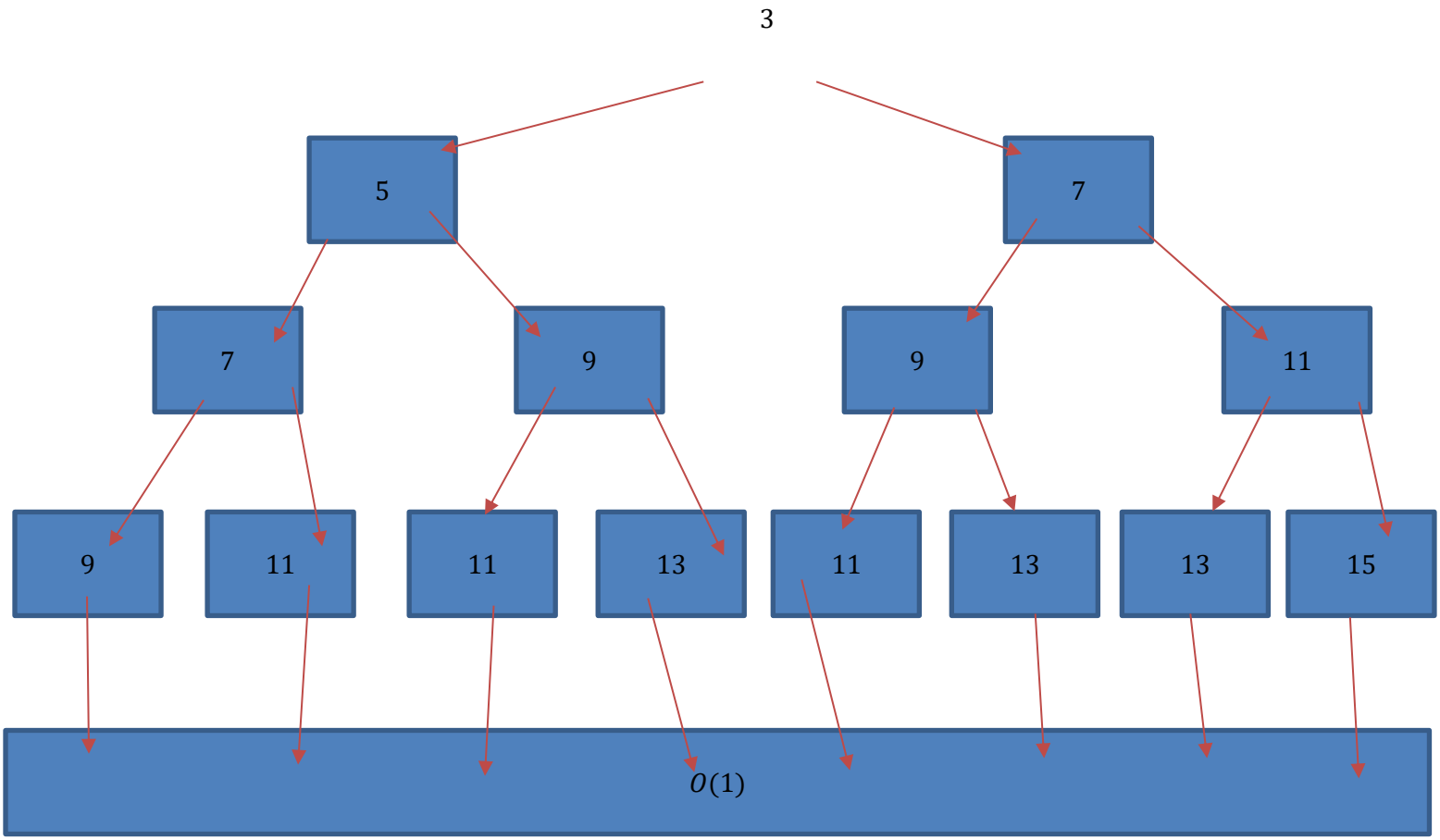
$$FF7(n) = (1-x)c_7 + (1-x)c_8 + (1-x)c_9 + \big((1-x)FF7(n-2) + (1-x)FF7(n-1)\big) + (1-x)c_{10} + c_{11}$$, *kai mūsų x = 0, t.y. praeinama if sąlyga.*

Suprastinę viską, gauname rekurentinę lygtį

$$FF7(n) = FF7(n-2) + FF7(n-1)$$

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 $h = [\frac{n}{3}]$;

$T(n) = \lambda\left(2^{\frac{n}{3}}\right) \approx O(2^n) \approx O(2^n)$
Negavome asimptotiškai tikslaus įvertinimo. Tačiau blogiausias šios lygties asimptotinis sudėtingumas yra $O(2^n)$

Surandame mūsų T(n).

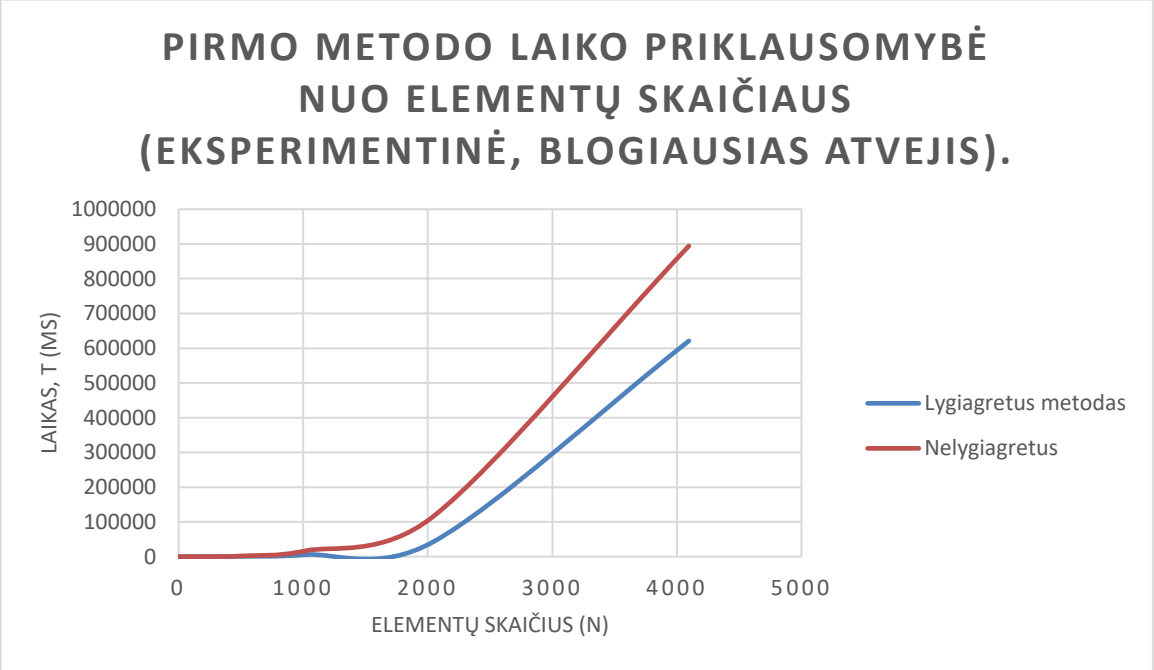
Įstatome į mūsų T(n),
$T(n) = c_1 + c_2 + c_3(1) + c_4(n) + c_5(n) + 2^n * n + c_6 * n + \sum^n FF7(n) + c_7$ = O(2^n*n+n);

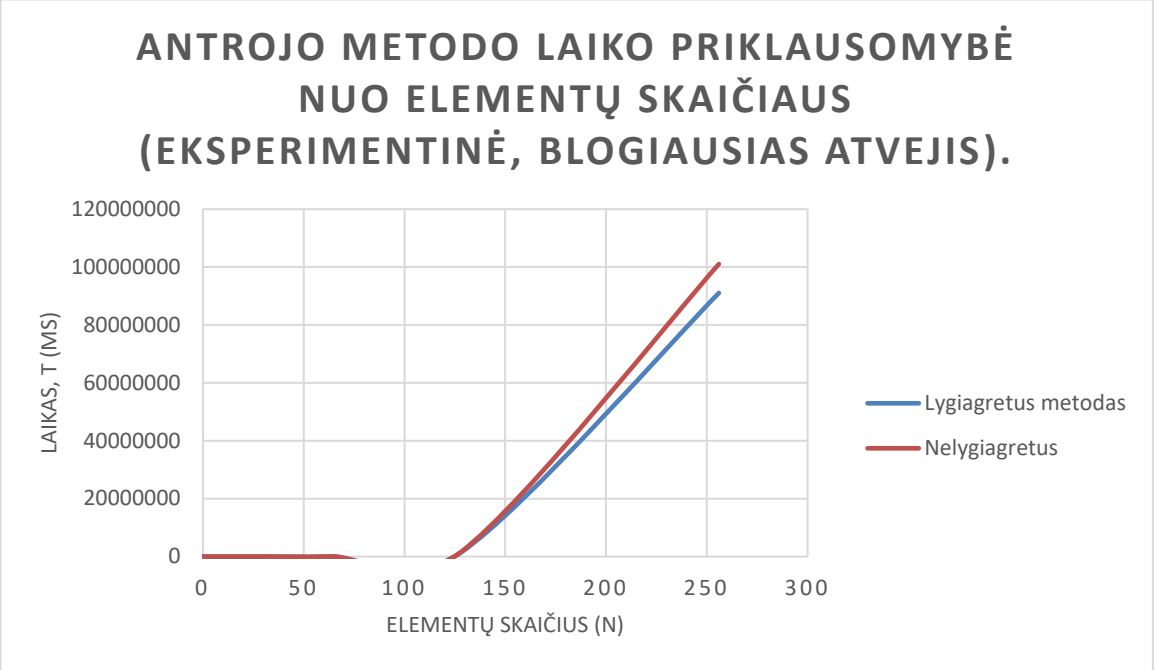
Gauname, kad mūsų blogiausias šio metodo asimptotinis sudėtingumas yra

T(n) = O($n2^n + n$);

Tačiau, kadangi mūsų procedūrą yra lygiagreti, todėl pats vykdymo laikas bus trumpesnis (priklausant nuo kompiuterio resursų, procesoriaus gijų skaičiaus) negu paprastos nelygiagrečios procedūros, tačiau lygiagretinimas šiuo atveju mūsų asimptotinio sudėtingumo nesumažino.

Atliekame našumo testus (lygiagretiem ir nelygiagretiem) pirmajam ir antrajam metodams.





3.3. Šaltiniai

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

4. Inžinerinis projektas