KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Algoritmų sudarymas ir analizė (P170B400)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/1 gr. studentas

Vytenis Kriščiūnas

2023 m. vasario 20 d.

Priėmė:

Lekt. Vidmantas Rimavičius

KAUNAS 2023

TURINYS

1. Rekurentinės lygtys

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).

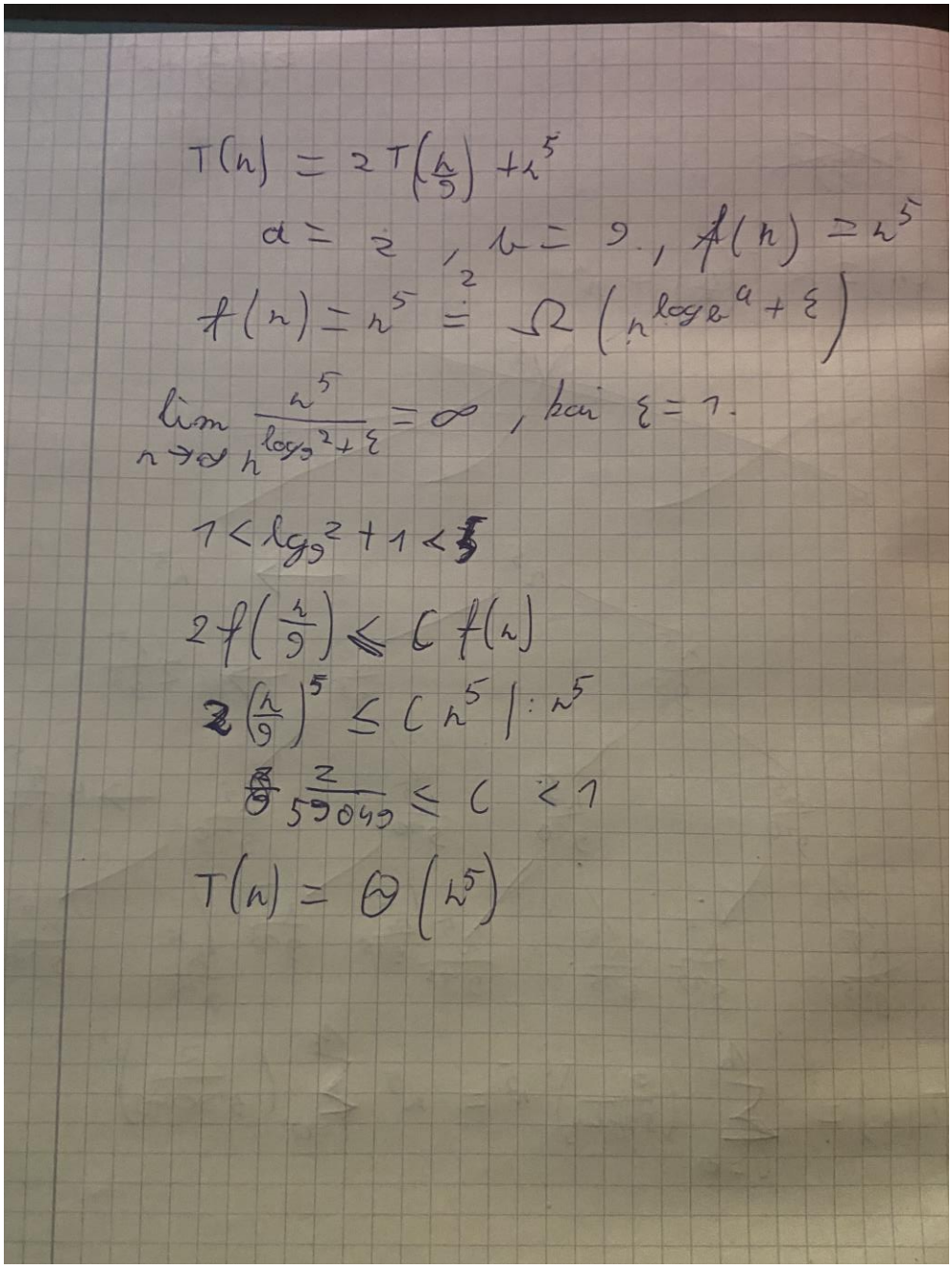
*1.1. Pirmoji rekurentinė lygtis (T(n) = 2*T(n/9)+n^5)*

1.1.1 Kodo analizė

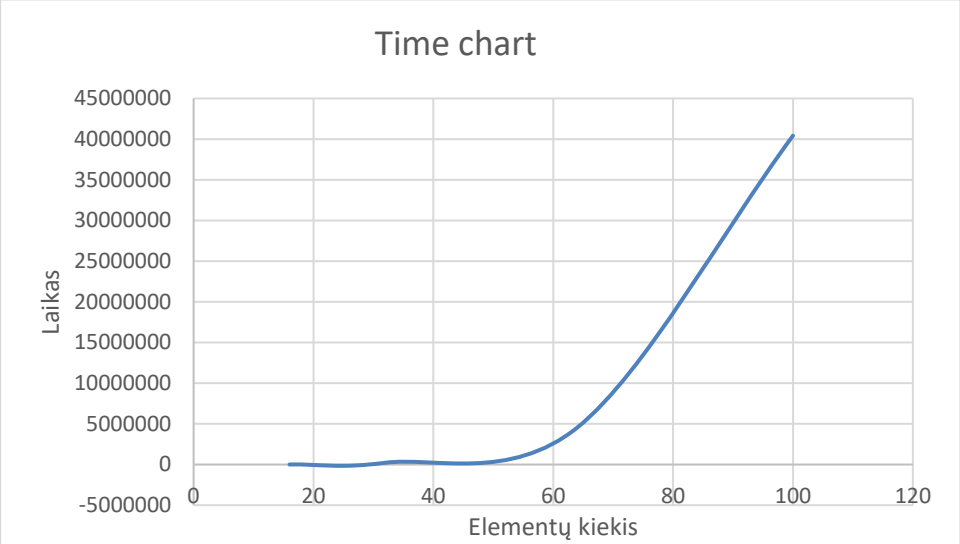
```
//T(n) = 2*T(n/9)+n^5
    public static void R1(int[] a, int n)
                                                        //Laikas | Kartai
    {
        if (n < 9)                                      //c1 | 1
        {
            return;                                     //c2 | 1
        }

        for (int i = 0; i < n; i++)                     //c3 | n
            for (int j = 0; j < n; j++)                 //c4 | n
                for (int z = 0; z < n; z++)             //c5 | n
                    for (int l = 0; l < n; l++)         //c6 | n
                        for (int x = 0; x < n; x++)     //c7 | n
                        {
                            sk++;                       //c8 | n
                        }

        R1(a, n / 9);                                   //T(n/9) | 1
        R1(a, n / 9);                                   //T(n/9) | 1
    }

    // T(n) = c1 + c2, n < 9
    // T(n) = 2*T(n/9) + c1 + c3*n*c4*n*c5*n*c6*n*n(c7+c8), n > 9
```

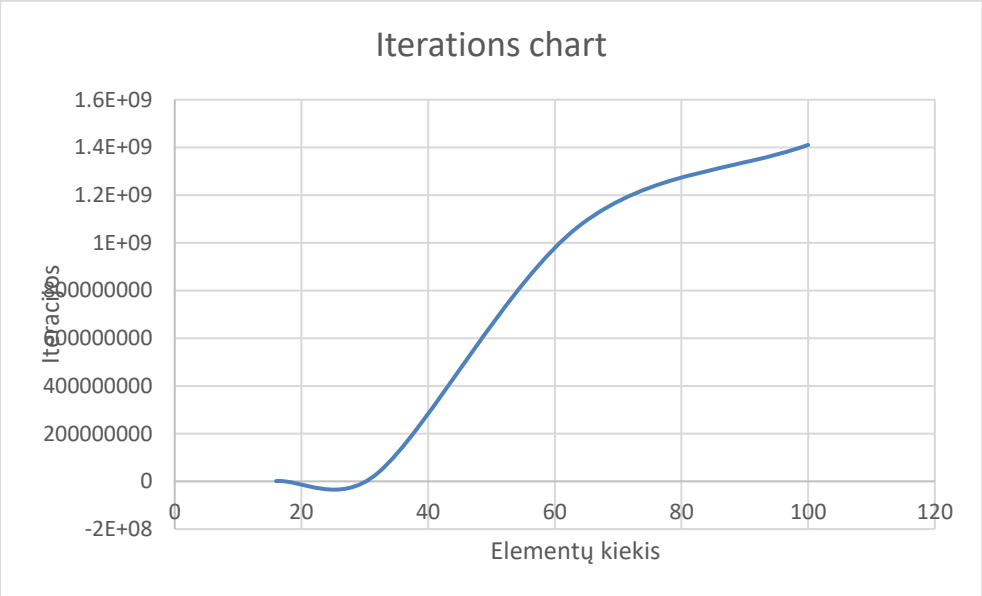
1.1.2 Rekurentinės lygties sprendimas

Gaunamas trečiasis atvejis:



1.1.3 Eksperimentinis tyrimas





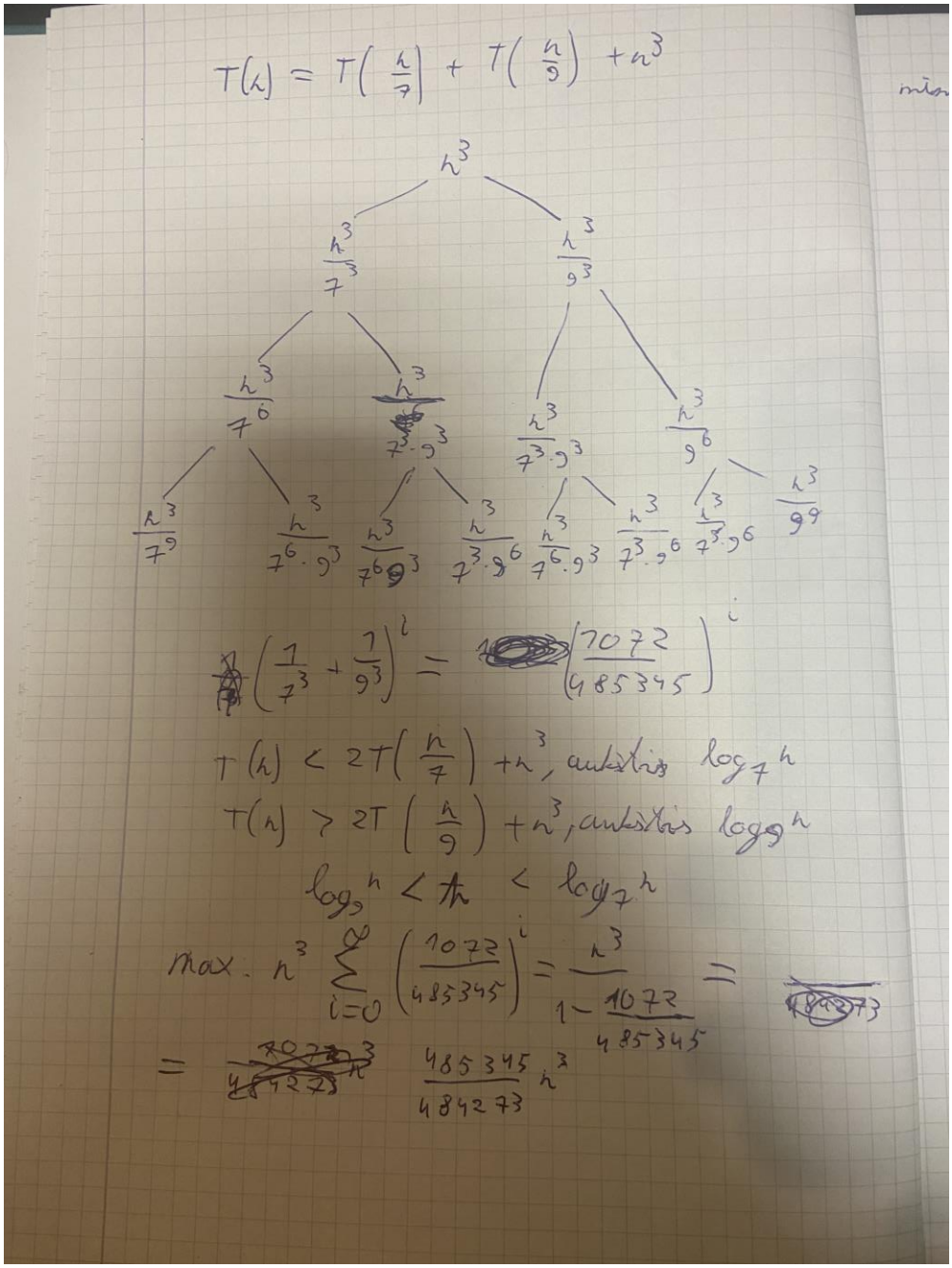
1.2. *Antroji rekurentinė lygtis (T(n) = T(n/7) + T(n/9) + n^3)*

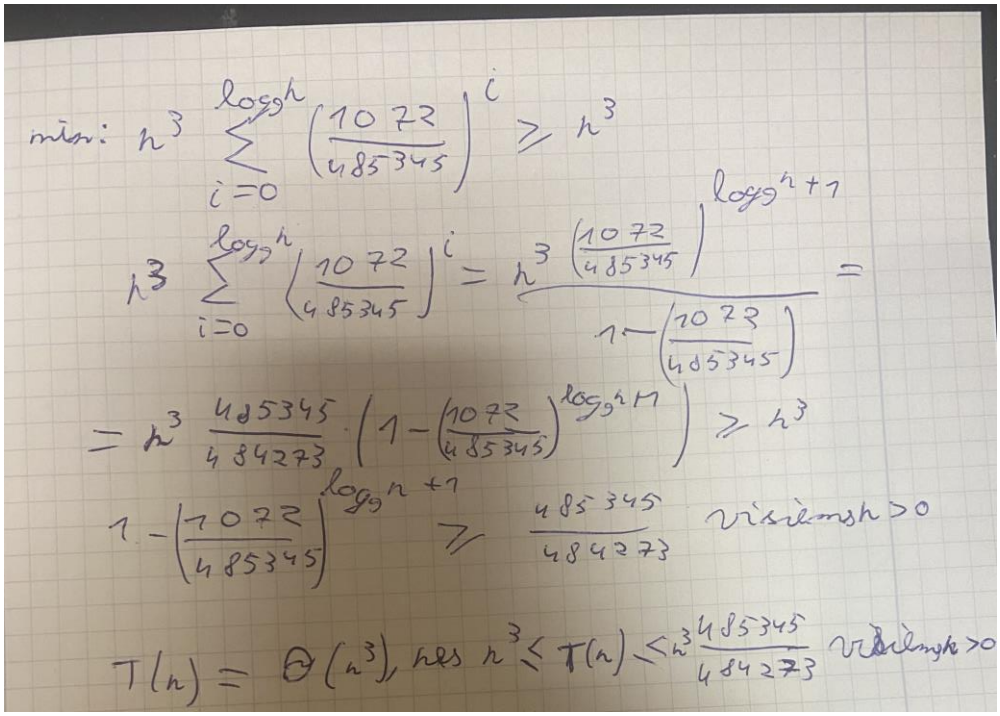
1.2.1 Kodo analizė

```
//T(n) = T(n/7) + T(n/9) + n^3
public static void R2(int[] a, int n)
                                                    //Laikas | Kartai
{
    if (n < 9)                                      //c1 | 1
    {
        return;                                     //c2 | 1
    }

    for (int i = 0; i < n; i++)                     //c3 | n
        for (int j = 0; j < n; j++)                 //c4 | n
            for (int z = 0; z < n; z++)             //c5 | n
            {
                sk++;                               //c6 | n
            }

    R2(a, n / 7);                                   //T(n/7) | 1
    R2(a, n / 9);                                   //T(n/9) | 1
}
```

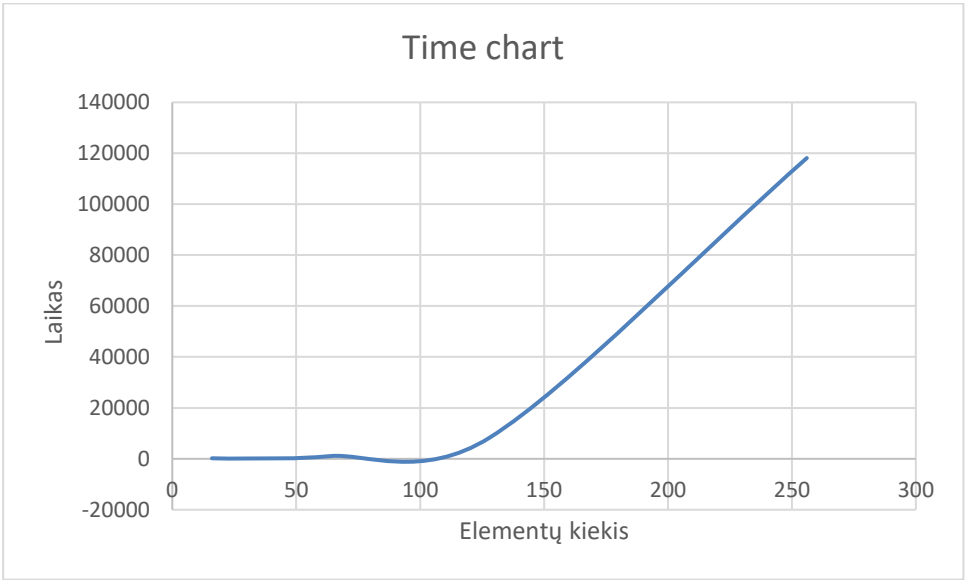
```
// T(n) = c1 + c2, n < 9
// T(n) = T(n/7) + T(n/9) + c1 + c3*n*c4*n*n(c5+c6), n > 9
```

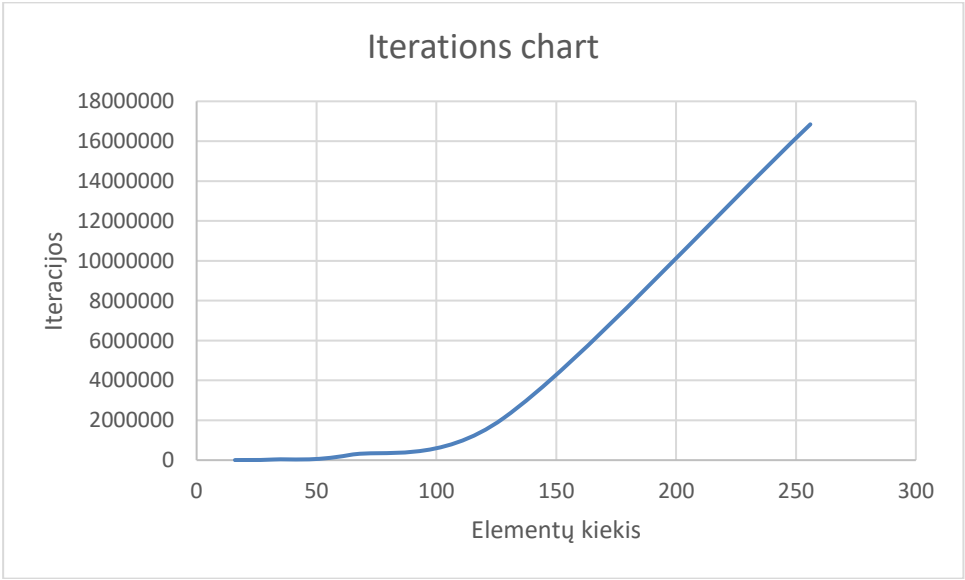
1.2.2 Rekurentinės lygties sprendimas





1.2.3 Eksperimentinis tyrimas



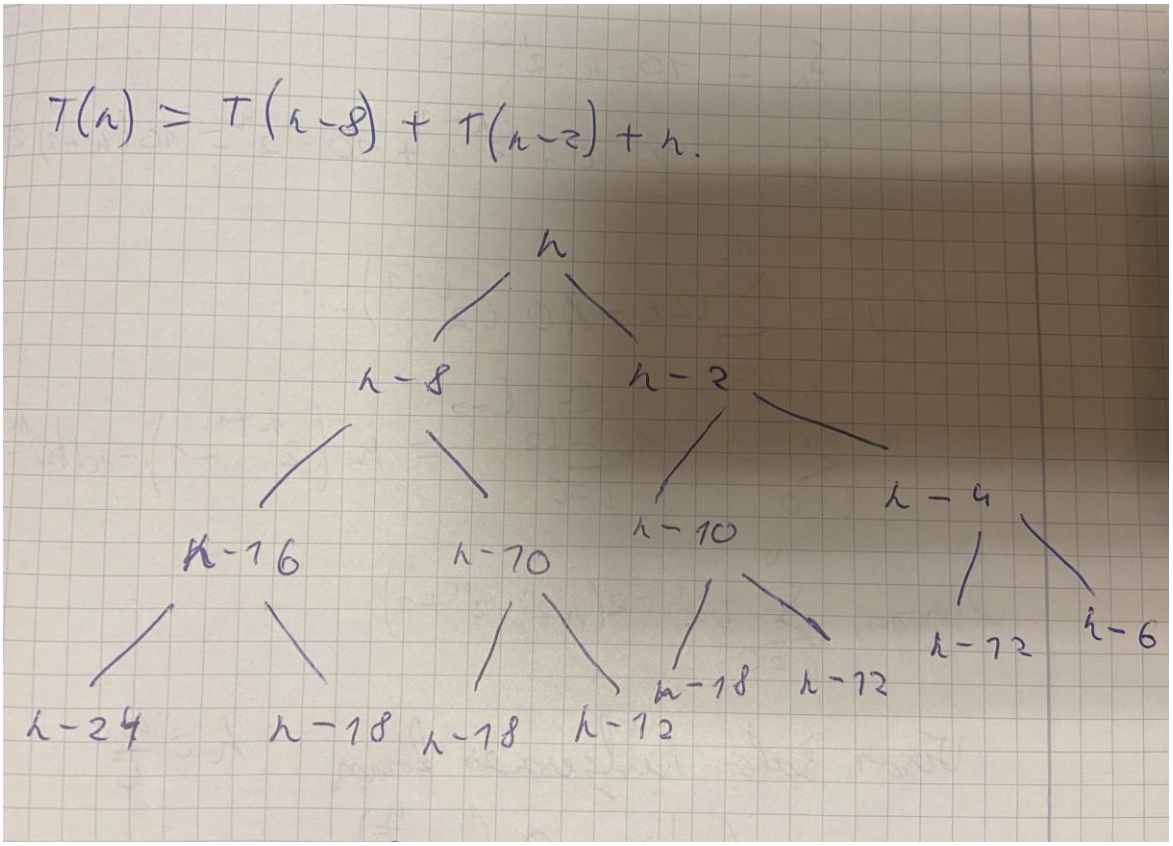


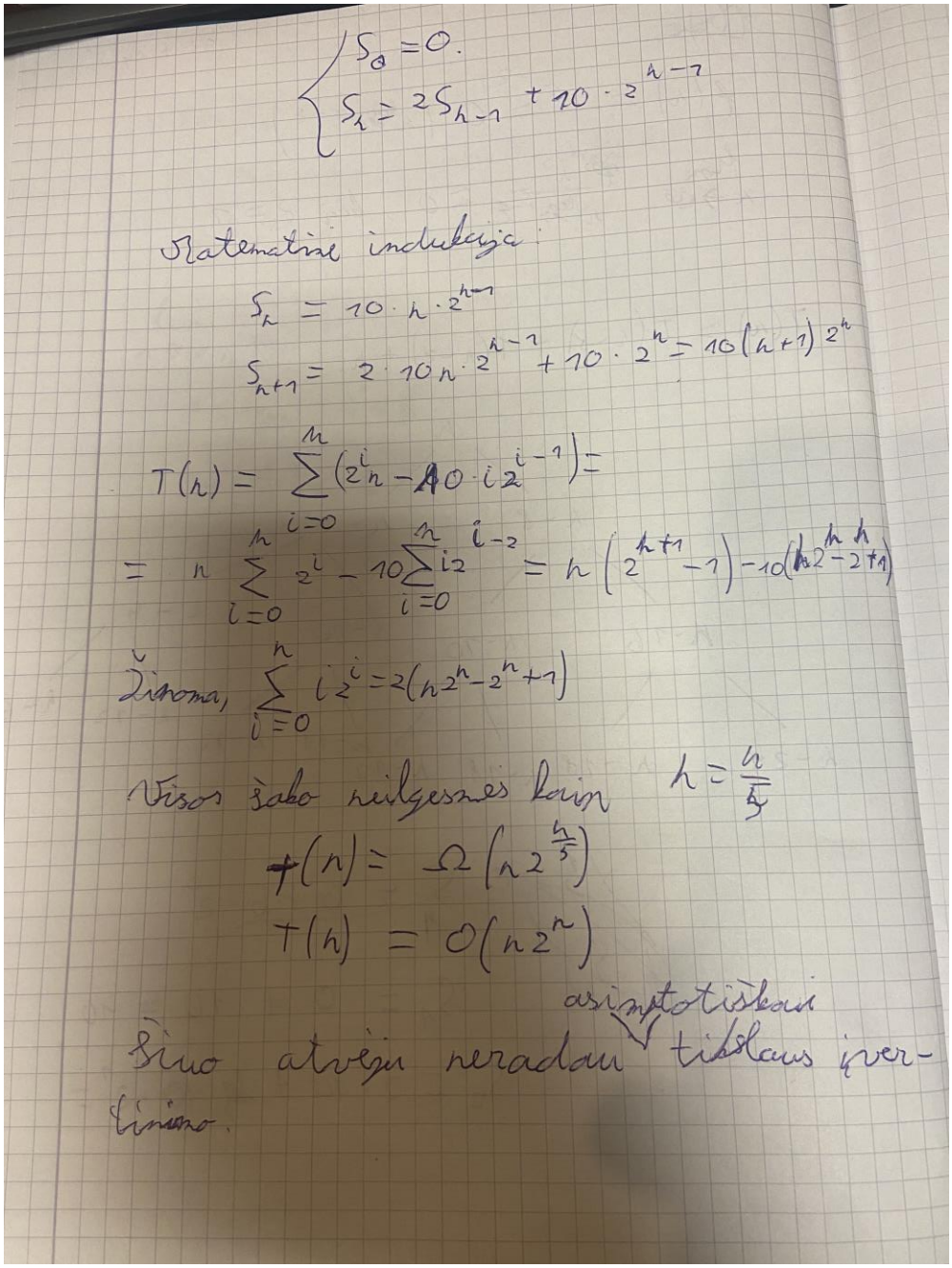
1.3. Trečioji rekurentinė lygtis (T(n) = T(n-8) + T(n-2) + n)

1.3.1 Kodo analizė

```
//T(n) = T(n - 8) + T(n - 2) + n
public static void R3(int[] a, int n)
                                                    //Laikas | Kartai
{
    if (n < 0)                                      //c1 | 1
    {
        return;                                     //c2 | 1
    }

    for (int i = 0; i < n; i++)                     //c3 | n
    {
        sk++;                                       //c4 | n
    }

    R3(a, n - 8);                                   //T(n - 8) | 1
    R3(a, n - 2);                                   //T(n - 2) | 1

}

// T(n) = c1 + c2, n < 0
// T(n) = T(n - 8) + T(n - 2) + c1 + n(c3+c4), n > 0
```

1.3.2 Rekurentinės lygties sprendimas





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

Matematine indukcija:

$$S_n = 10 \cdot n \cdot 2^{n-1}$$

$$S_{n+1} = 2 \cdot 10n \cdot 2^{n-1} + 10 \cdot 2^n = 10(n+1)2^n$$

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

$$= n \sum_{i=0}^{n} 2^i - 10 \sum_{i=0}^{n} i 2^{i-1} = n\left(2^{n+1} - 1\right) - 10\left(n2^n - 2^n + 1\right)$$

Žinoma, $\sum_{i=0}^{n} i 2^i = 2(n2^n - 2^n + 1)$

Visos šako neilgesnės kainos $n = \frac{n}{5}$

$$T(n) = \Omega\left(n 2^{\frac{n}{5}}\right)$$

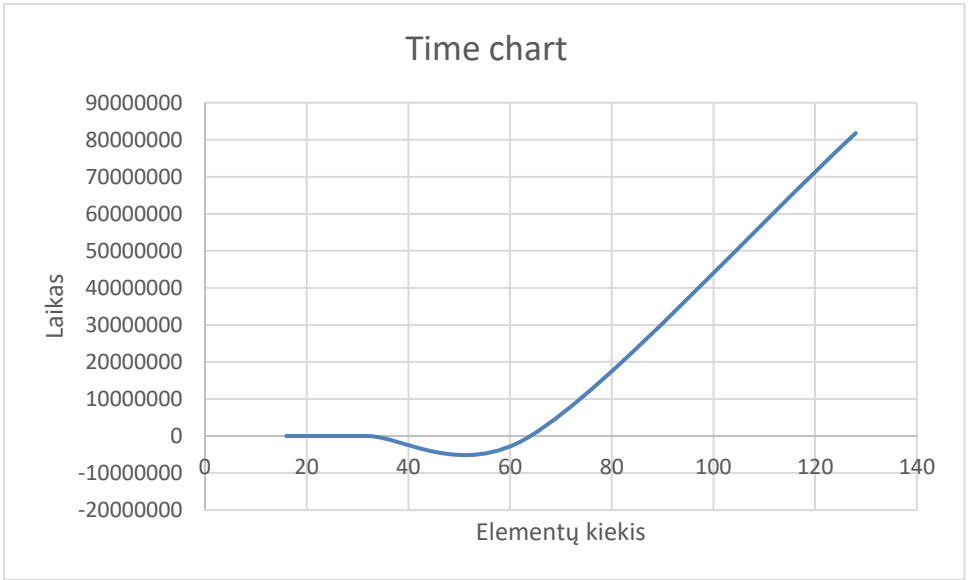
$$T(n) = O(n 2^n)$$

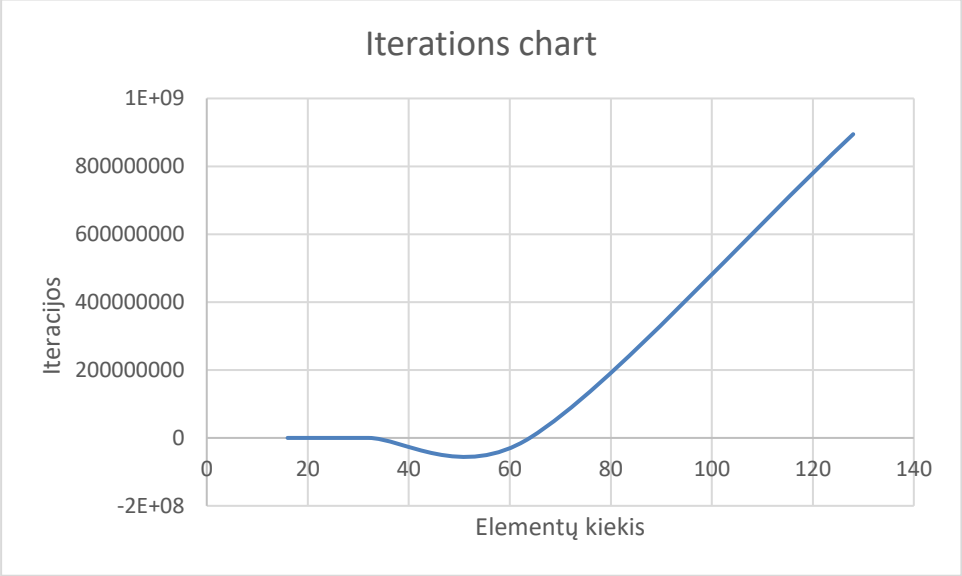
asimptotiškai

Šiuo atveju neradau tikslaus įver-

tinimo.

1.3.3 Eksperimentinis tyrimas





1.4. Programos kodas

```
using BenchmarkDotNet.Running;
using System.Diagnostics;

internal class Program
{
    private static int sk;
    private static void Main(string[] args)
    {
        int[] a = new int[0];
        Console.WriteLine("Pirmoji rekurentine lygtis: ");
        a = new int[16];
        Test1(a);
        a = new int[32];
        Test1(a);
        a = new int[64];
        Test1(a);
        a = new int[100];
        Test1(a);
        a = new int[256];
        Test1(a);
        Console.WriteLine();

        //Console.WriteLine("Antroji rekurentine lygtis: ");
        //a = new int[16];
        //Test2(a);
        //a = new int[32];
        //Test2(a);
        //a = new int[64];
        //Test2(a);
        //a = new int[128];
        //Test2(a);
        //a = new int[256];
        //Test2(a);
        Console.WriteLine();

        //Console.WriteLine("Trecioji rekurentine lygtis: ");
        //a = new int[16];
        //Test3(a);
        //a = new int[32];
        //Test3(a);
        //a = new int[64];
        //Test3(a);
        //a = new int[128];
```

```
        //Test3(a);

    }

    public static void Test1(int[] a)
    {
        Stopwatch time = new Stopwatch();
        sk = 0;
        time.Start();
        R1(a, a.Length);
        time.Stop();
        Console.WriteLine("Duomenu kiekis: {0}", a.Length);
        Console.WriteLine("Elapsed time: {0} microsec.",
time.Elapsed.TotalMicroseconds);
        Console.WriteLine("Iterations: {0}", sk);

    }
    public static void Test2(int[] a)
    {
        Stopwatch time = new Stopwatch();
        sk = 0;
        time.Start();
        R2(a, a.Length);
        time.Stop();
        Console.WriteLine("Duomenu kiekis: {0}", a.Length);
        Console.WriteLine("Elapsed time: {0} microsec.",
time.Elapsed.TotalMicroseconds);
        Console.WriteLine("Iterations: {0}", sk);
    }
    public static void Test3(int[] a)
    {
        Stopwatch time = new Stopwatch();
        sk = 0;
        time.Start();
        R3(a, a.Length);
        time.Stop();
        Console.WriteLine("Duomenu kiekis: {0}", a.Length);
        Console.WriteLine("Elapsed time: {0} microsec.",
time.Elapsed.TotalMicroseconds);
        Console.WriteLine("Iterations: {0}", sk);
    }

    //T(n) = 2*T(n/9)+n^5
    public static void R1(int[] a, int n)
                                                    //Laikas | Kartai
    {
        if (n < 9)                                  //c1 | 1
        {
            return;                                 //c2 | 1
        }

        for (int i = 0; i < n; i++)                 //c3 | n
            for (int j = 0; j < n; j++)             //c4 | n
                for (int z = 0; z < n; z++)         //c5 | n
                    for (int l = 0; l < n; l++)     //c6 | n
                        for (int x = 0; x < n; x++) //c7 | n
                        {
                            sk++;                   //c8 | n
                        }

        R1(a, n / 9);                               //T(n/9) | 1
        R1(a, n / 9);                               //T(n/9) | 1
    }

    // T(n) = c1 + c2, n < 9
```

```
// T(n) = 2*T(n/9) + c1 + c3*n*c4*n*c5*n*c6*n*n(c7+c8), n > 9


//T(n) = T(n/7) + T(n/9) + n^3
public static void R2(int[] a, int n)
                                                    //Laikas | Kartai
{
    if (n < 9)                                      //c1 | 1
    {
        return;                                     //c2 | 1
    }

    for (int i = 0; i < n; i++)                     //c3 | n
        for (int j = 0; j < n; j++)                 //c4 | n
            for (int z = 0; z < n; z++)             //c5 | n
            {
                sk++;                               //c6 | n
            }

    R2(a, n / 7);                                   //T(n/7) | 1
    R2(a, n / 9);                                   //T(n/9) | 1
}

// T(n) = c1 + c2, n < 9
// T(n) = T(n/7) + T(n/9) + c1 + c3*n*c4*n*n(c5+c6), n > 9


//T(n) = T(n - 8) + T(n - 2) + n
public static void R3(int[] a, int n)
                                                    //Laikas | Kartai
{
    if (n < 0)                                      //c1 | 1
    {
        return;                                     //c2 | 1
    }

    for (int i = 0; i < n; i++)                     //c3 | n
    {
        sk++;                                       //c4 | n
    }

    R3(a, n - 8);                                   //T(n - 8) | 1
    R3(a, n - 2);                                   //T(n - 2) | 1

}

// T(n) = c1 + c2, n < 0
// T(n) = T(n - 8) + T(n - 2) + c1 + n(c3+c4), n > 0


  }
```

2. BMP programa

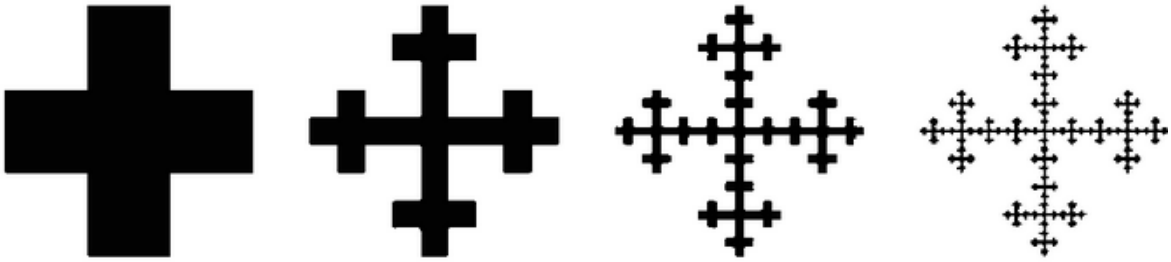
2.1. Užduotis

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

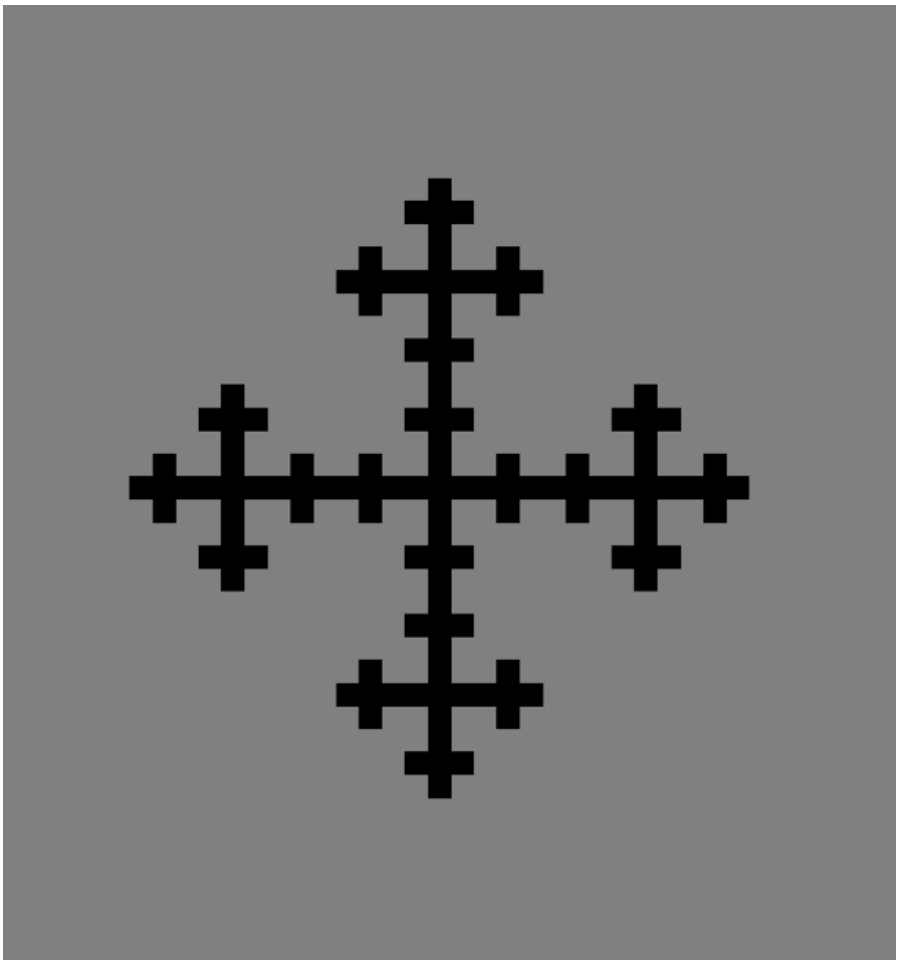
- Programos rezultatas BMP formato bylos demonstruojančios programos rekursijas. (3 balai)
- Eksperimentiškai nustatykite darbo laiko ir veiksmų skaičiaus priklausomybę nuo generuojamo paveikslėlio dydžio (taškų skaičiaus). Gautus rezultatus atvaizduokite

grafikais. Grafiką turi sudaryti nemažiau kaip 5 taškai ir paveikslėlio taškų skaičius turi didėti proporcingai (kartais). (1 balas)

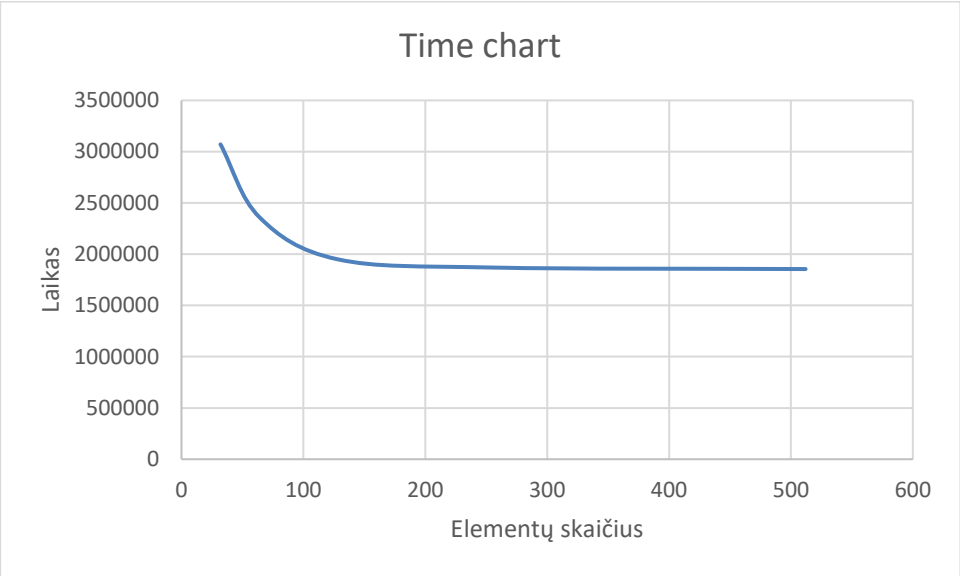
- Analitiškai įvertinkite procedūros, kuri generuoja paveikslėlį, veiksmų skaičių sudarydami rekurentinę lygtį ir ją išspręskite. Gautas rezultatas turi patvirtinti eksperimentinius rezultatus. (1 balas)

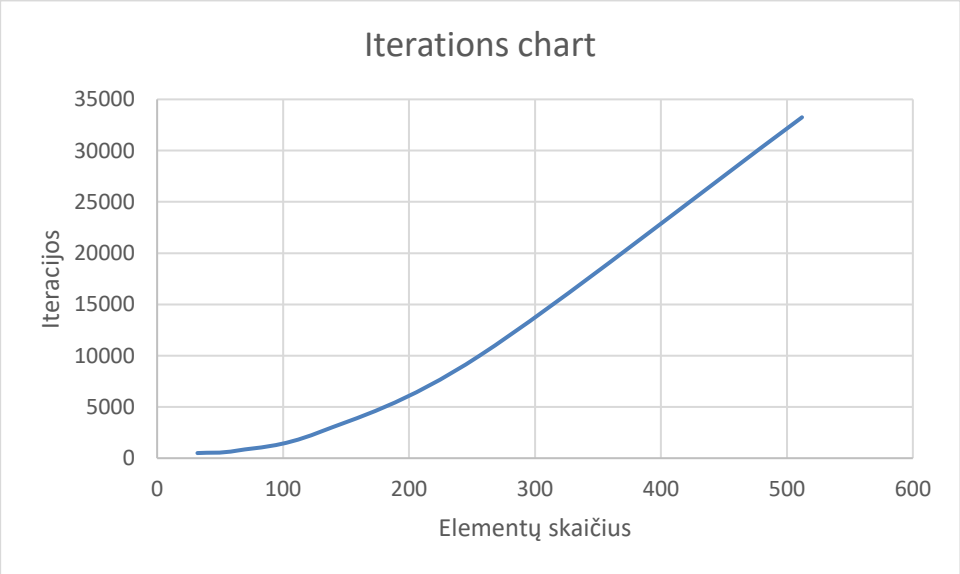


2.2. Rezultatai



2.3. *Eksperimentinis tyrimas*





2.4. *Kodas*

Renderer.cs

```
using ConsoleApp1;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SimpleREngine
{
    internal class Renderer
    {
        public Renderer(string OutputName, ushort Width, ushort Height, uint
FillingColor) // Color format is ARGB (to define recomended hex: 0xAARRGGBB),
coordinates start from bottom left corner, 1 unit is 1 pixel
        {
            this.Width = Width;
            this.Height = Height;
            Buffer = new uint[Width * Height];
```

```
            Array.Fill(Buffer, FillingColor);

            this.OutputName = OutputName;
            if (!OutputName.Contains(".bmp"))
                this.OutputName += ".bmp";
        }


        public void DrawFilledSquare(double X, double Y, double Width, double
Height, uint Color = 0)
        {
            for (double SY = 0; SY < Height; SY++)
            {
                for (double SX = 0; SX < Width; SX++)
                {
                    SetPixel(SX + X, SY + Y, Color);
                }
            }
            Write();
        }


        private void SetPixel(double X, double Y, uint Color)
        {
            int Pixel = GetPixel(X, Y);
            if (Pixel < 0)
                return;

            Buffer[Pixel] = Color;
            Program.sk++;
        }

        private int GetPixel(double X, double Y)
        {
            int Pixel = ((int)Math.Round(Y) * Width) + (int)Math.Round(X);
            if (Pixel > Buffer.Length)
                return -1;

            if (X < 0)
                return -1;
            else if (X > Width)
                return -1;

            return Pixel;
        }

        public void Write()
        {
            using (FileStream File = new FileStream(OutputName, FileMode.Create,
FileAccess.Write))
            {
                File.Write(new byte[] { 0x42, 0x4D }); // BM
                File.Write(BitConverter.GetBytes(Height * Width * sizeof(uint) +
0x1A)); // Size
                File.Write(BitConverter.GetBytes(0)); // Reserved (0s)
                File.Write(BitConverter.GetBytes(0x1A)); // Image Offset (size of
the header)
                File.Write(BitConverter.GetBytes(0x0C)); // Header size (size is 12
bytes)
                File.Write(BitConverter.GetBytes(Width)); // Width
                File.Write(BitConverter.GetBytes(Height)); // Height
                File.Write(BitConverter.GetBytes((ushort)1)); // Color plane
                File.Write(BitConverter.GetBytes((ushort)32)); // bits per pixel

                byte[] Converted = new byte[Buffer.Length * sizeof(uint)];
```

```
                System.Buffer.BlockCopy(Buffer, 0, Converted, 0, Converted.Length);

                File.Write(Converted);
                File.Close();
            }
        }

        private readonly uint[] Buffer;
        private readonly ushort Width;
        private readonly ushort Height;
        private readonly string OutputName;
    }
}
```

Program.cs

```
using SimpleREngine;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using static System.Net.Mime.MediaTypeNames;

namespace ConsoleApp1
{
    class Program
    {
        public static int sk;
        static void Main(string[] args)
        {
            Renderer Render = new Renderer("Result", 1000, 1000, 0xFF808080);
            Test(Render, 500, 500, 32, 3);
            Test(Render, 500, 500, 64, 3);
            Test(Render, 500, 500, 128, 3);
            Test(Render, 500, 500, 256, 3);
            Test(Render, 500, 500, 512, 3);

        }

        static void RecursiveSquare(Renderer Render, double X, double Y, double
Size, uint Iteration)
        {
            if (Iteration == 0)
            {
                Render.DrawFilledSquare(X - (Size * (1d / 2d)), Y - (Size * (1d /
6d)), Size, Size * (1d / 3d));
                Render.DrawFilledSquare(X - (Size * (1d / 6d)), Y - (Size * (1d /
2d)), Size * (1d / 3d), Size);

                return;
            }

            double IterSize = Math.Pow(3, Iteration + 1);
            double NextSize = (Math.Pow(3, Iteration) / IterSize) * Size;

            double Distance = (Size / 2) - (NextSize / 2);

            RecursiveSquare(Render, X, Y, NextSize, Iteration - 1);
            RecursiveSquare(Render, X + Distance, Y, NextSize, Iteration - 1);
            RecursiveSquare(Render, X - Distance, Y, NextSize, Iteration - 1);
```

```
            RecursiveSquare(Render, X, Y + Distance, NextSize, Iteration - 1);
            RecursiveSquare(Render, X, Y - Distance, NextSize, Iteration - 1);
        }

        public static void Test(Renderer Render, double X, double Y, double Size,
uint Iteration)
        {
            Stopwatch time = new Stopwatch();
            sk = 0;
            time.Start();
            RecursiveSquare(Render, X, Y, Size, Iteration);
            time.Stop();
            Console.WriteLine("Tasku skaicius: {0}", Size);
            Console.WriteLine("Elapsed time: {0} microsec.",
time.Elapsed.TotalMicroseconds);
            Console.WriteLine("Iterations: {0}", sk);

        }
    }
}
```