**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIKOS FAKULTETAS**

# Algoritmų sudarymas ir analizė (P170B400)

***Laboratorinių darbų ataskaita***

Atliko:

IFF-1/4 gr. studentas

Mildaras Karvelis

2023 m. gegužės 17 d.

Priėmė:

Doc. Pilkauskas Vytautas

**KAUNAS 2023**

# TURINYS

# 1. LD1

## 1.1. Pirma darbo 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: vykdymo laiką ir veiksmų skaičių) ir patikrinkite ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus (1 balas).

$$T(n) = 3 * T\left(\frac{n}{6}\right) + n^2$$

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{n}{3}\right) + n$$

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

### 1.1.1 Programos kodas

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Threading;


namespace LD1
{
    class Program
    {
            //T(n)=3*T(n/6)+n^2
            static void First(int[] array, int n, int m)
            {

                if (m - n < 2)
                {
                    return;
                }

                First(array, n, n + (m - n) / 6);
                First(array, m - (m - n) / 6, m);
                First(array, m - (n - m) / 6, m);

                for (int i = n; i < m; i++)
```

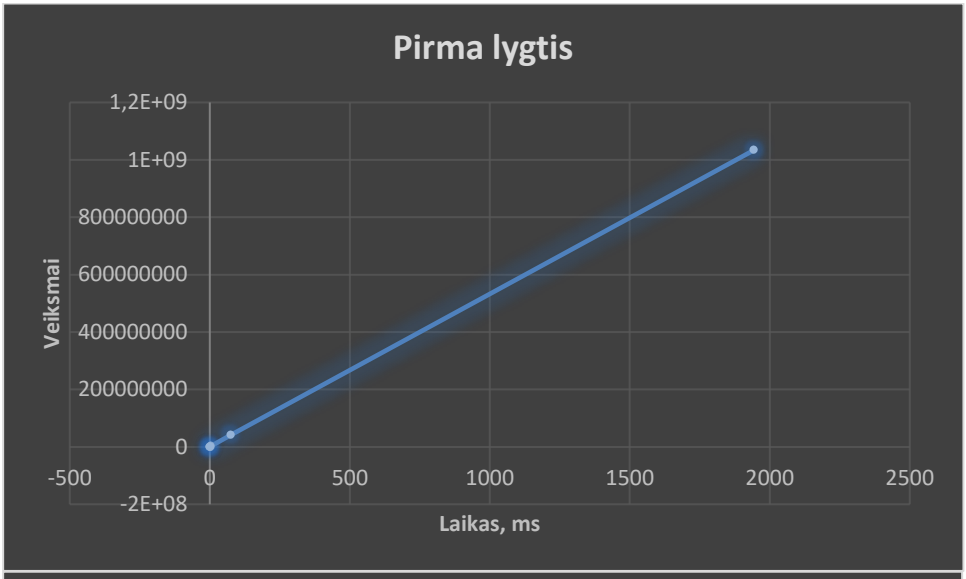
```
        {
            for (int j = n; j < m; j++)
            {
                array[0]++;
            }
        }
    }

    //T(n)=T(n/10)+ T(n/3)+ n
    static void Second(int[] array, int n, int m)
    {
        if (m - n < 9)
        {
            return;
        }

        Second(array, n, n + (m - n) / 3);
        Second(array, m - (m - n) / 10, m);

        for (int i = 0; i < n; i++)
        {
            array[0]++;
        }
    }

    //T(n)=T(n–9)+ T(n–1)+1
    static void Third(int[] array, int n)
    {

        if (n <= 9)
        {
            return;
        }

        Third(array, n - 9);
        Third(array, n - 1);

        array[0]++;
    }

    static void Main(string[] args)
    {

        int m = 70;
        for (int i = 0; i < 5; i++)
        {
            int[] array = new int[m];
            var watch = System.Diagnostics.Stopwatch.StartNew();
            //m = 1000000; m *= 2;
            //First(array, 0, array.Length);

            //m = 100; m*=2
            //Second(array, 0, array.Length);
```

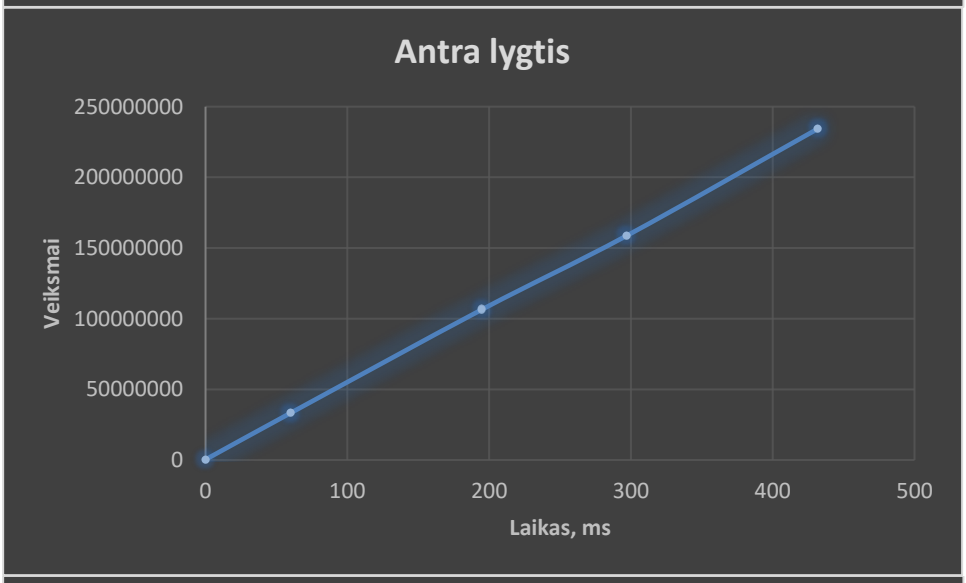
```
                    //m = 70; m += 10;
                    Third(array, array.Length);

                    watch.Stop();
                    var elapsedMs = watch.ElapsedMilliseconds;
                    Console.WriteLine(m + " iteraciju skaicius, " + array[0] + " veiksmų skaicius =
" + elapsedMs + " ms");
                    m += 10;
                }
            }
        }

    }
}
```

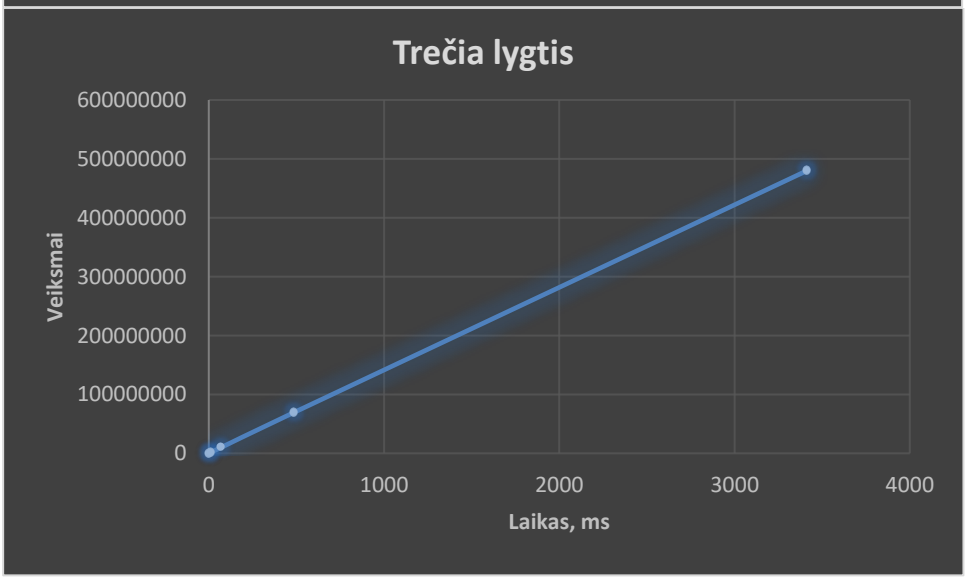
### *1.1.2 Rezultatai*

```
Pirma
50 iteraciju skaicius, 2628 veiksmu skaicius = 0 ms
250 iteraciju skaicius, 66006 veiksmu skaicius = 0 ms
1250 iteraciju skaicius, 1653852 veiksmu skaicius = 3 ms
6250 iteraciju skaicius, 41356106 veiksmu skaicius = 75 ms
31250 iteraciju skaicius, 1033998340 veiksmu skaicius = 1944 ms
Antra
20000 iteraciju skaicius, 277934 veiksmu skaicius = 0 ms
520000 iteraciju skaicius, 33232916 veiksmu skaicius = 60 ms
1020000 iteraciju skaicius, 106480987 veiksmu skaicius = 195 ms
1520000 iteraciju skaicius, 158677546 veiksmu skaicius = 297 ms
2020000 iteraciju skaicius, 234462396 veiksmu skaicius = 432 ms
Trecia
70 iteraciju skaicius, 211367 veiksmu skaicius = 1 ms
80 iteraciju skaicius, 1459214 veiksmu skaicius = 10 ms
90 iteraciju skaicius, 10075041 veiksmu skaicius = 71 ms
100 iteraciju skaicius, 69564611 veiksmu skaicius = 487 ms
110 iteraciju skaicius, 480322071 veiksmu skaicius = 3416 ms
```

. *1 pav lygčių rezultatai konsolėje*



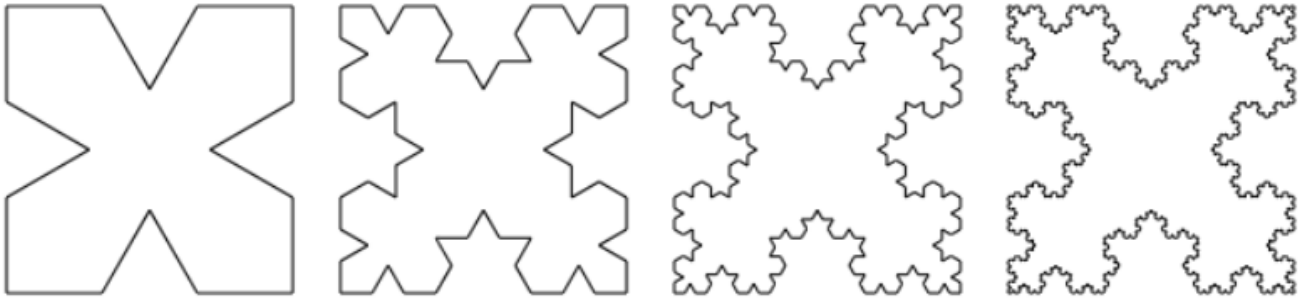

## *1.2. Antra darbo užduotis*

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)



### *1.2.1 Programos kodas*

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Threading;


namespace LD1
{
   class Program
   {
      public class Point
      {
         public double x;
         public double y;

         public Point(double x, double y)
         {
            this.x = x;
            this.y = y;
         }
      }
      public class Line
      {
         public double xStart;
         public double xEnd;
         public double yStart;
         public double yEnd;
         public string orientation;

         public Line(double xStart, double yStart, double xEnd, double yEnd, string orientation)
         {
            this.xStart = xStart;
            this.yStart = yStart;
            this.xEnd = xEnd;
            this.yEnd = yEnd;
            this.orientation = orientation;
         }
         public Point PointA()
```

```
{
    return new Point(this.xStart, this.yStart);
}
public Point PointB()
{
    return new Point(this.xStart + (this.xEnd - this.xStart) / 3,
        this.yStart + (this.yEnd - this.yStart) / 3);
}
public Point PointC()
{
    Point tempA = PointB();
    Point tempB = PointD();
    double tempX = tempB.x - tempA.x;
    double tempY = tempB.y - tempA.y;
    Point temp = new Point(tempX, tempY);
    double angleInRadians;
    if (orientation == "top") angleInRadians = -60 * (Math.PI / 180);
    else angleInRadians = 60 * (Math.PI / 180);
    double x = temp.x * Math.Cos(angleInRadians) + temp.y * Math.Sin(angleInRadians);
    double y = -1 * temp.x * Math.Sin(angleInRadians) + temp.y
        * Math.Cos(angleInRadians);
    x += tempA.x;
    y += tempA.y;
    Point c = new Point(x, y);
    return c;
}

public Point PointD()
{
    return new Point(this.xStart + ((this.xEnd - this.xStart)) / 3 * 2,
        this.yStart + ((this.yEnd - this.yStart) / 3) * 2);
}

public Point PointE()
{
    return new Point(this.xEnd, this.yEnd);
}
internal class Renderer
{
    // Color format is ARGB (to define recomended hex: 0xAARRGGBB),
    // coordinates start from bottom left corner, 1 unit is 1 pixel
    public Renderer(string outputName, ushort Width, ushort Height, uint FillingColor)

    {
        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 Draw(double X0, double Y0, double X1, double Y1, int[] array, double Precision = 1, uint Color = 0)
    {
        double Length = Math.Sqrt(Math.Pow(X0 - X1, 2) + Math.Pow(Y0 - Y1, 2));
        double XStep = (X1 - X0) / (Length / Precision);
```

```
    double YStep = (Y1 - Y0) / (Length / Precision);
    double XRun = X0;
    double YRun = Y0;
    for (double i = 0; i < Length; i += Precision)
    {
        array[0]++;
        XRun += XStep;
        YRun += YStep;
        SetPixel(XRun, YRun, Color);
    }
}

public void Erase(double X0, double Y0, double X1, double Y1, int[] array, double Precision = 0.5,
    uint Color = 0xffffff)
{
    double Length = Math.Sqrt(Math.Pow(X0 - X1, 2) + Math.Pow(Y0 - Y1, 2));

    double XStep = (X1 - X0) / (Length / Precision);
    double YStep = (Y1 - Y0) / (Length / Precision);

    double XRun = X0;
    double YRun = Y0;
    for (double i = 0; i < Length; i += Precision)
    {
        array[0]++;
        XRun += XStep;
        YRun += YStep;

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

    Buffer[Pixel] = Color;
}

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 DrawFirstShape(Renderer render, double X, double Y, double Size, ArrayList list, int[] array)
{
    double line = Size / 3;
    //Virsus
    render.Draw(X + line * 1.5, Y + line * 1.5, X + line * 0.5, Y + line * 1.5, array);
    render.Draw(X + line * 0.5, Y + line * 1.5, X + line * 0, Y + line * 0.5, array);
```

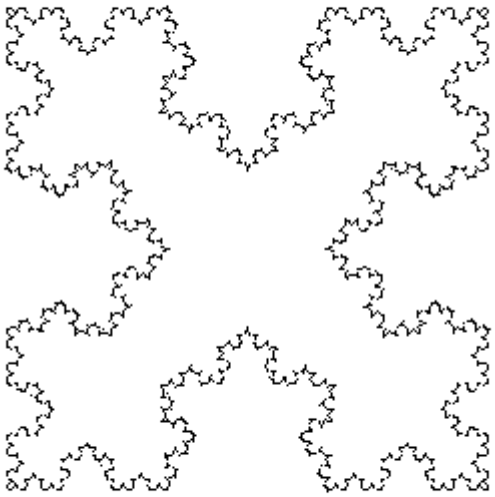
```
    render.Draw(X + line * 0, Y + line * 0.5, X + line * -0.5, Y + line * 1.5, array);
    render.Draw(X + line * -0.5, Y + line * 1.5, X + line * -1.5, Y + line * 1.5, array);
    list.Add(new Line(X + line * 1.5, Y + line * 1.5, X + line * 0.5, Y + line * 1.5, "top"));
    list.Add(new Line(X + line * 0.5, Y + line * 1.5, X + line * 0, Y + line * 0.5, "top"));
    list.Add(new Line(X + line * 0, Y + line * 0.5, X + line * -0.5, Y + line * 1.5, "top"));
    list.Add(new Line(X + line * -0.5, Y + line * 1.5, X + line * -1.5, Y + line * 1.5, "top"));

    //Desine
    render.Draw(X + line * 1.5, Y + line * 1.5, X + line * 1.5, Y + line * 0.5, array);
    render.Draw(X + line * 1.5, Y + line * 0.5, X + line * 0.5, Y + line * 0, array);
    render.Draw(X + line * 0.5, Y + line * 0, X + line * 1.5, Y + line * -0.5, array);
    render.Draw(X + line * 1.5, Y + line * -0.5, X + line * 1.5, Y + line * -1.5, array);
    list.Add(new Line(X + line * 1.5, Y + line * 1.5, X + line * 1.5, Y + line * 0.5, "left"));
    list.Add(new Line(X + line * 1.5, Y + line * 0.5, X + line * 0.5, Y + line * 0, "left"));
    list.Add(new Line(X + line * 0.5, Y + line * 0, X + line * 1.5, Y + line * -0.5, "left"));
    list.Add(new Line(X + line * 1.5, Y + line * -0.5, X + line * 1.5, Y + line * -1.5, "left"));
    //Kaire
    render.Draw(X + line * -1.5, Y + line * -1.5, X + line * -1.5, Y + line * -0.5, array);
    render.Draw(X + line * -1.5, Y + line * -0.5, X + line * -0.5, Y + line * 0, array);
    render.Draw(X + line * -0.5, Y + line * 0, X + line * -1.5, Y + line * 0.5, array);
    render.Draw(X + line * -1.5, Y + line * 0.5, X + line * -1.5, Y + line * 1.5, array);
    list.Add(new Line(X + line * -1.5, Y + line * -1.5, X + line * -1.5, Y + line * -0.5, "left"));
    list.Add(new Line(X + line * -1.5, Y + line * -0.5, X + line * -0.5, Y + line * 0, "left"));
    list.Add(new Line(X + line * -0.5, Y + line * 0, X + line * -1.5, Y + line * 0.5, "left"));
    list.Add(new Line(X + line * -1.5, Y + line * 0.5, X + line * -1.5, Y + line * 1.5, "left"));
    //Apacia
    render.Draw(X + line * -1.5, Y + line * -1.5, X + line * -0.5, Y + line * -1.5, array);
    render.Draw(X + line * -0.5, Y + line * -1.5, X + line * 0, Y + line * -0.5, array);
    render.Draw(X + line * 0, Y + line * -0.5, X + line * 0.5, Y + line * -1.5, array);
    render.Draw(X + line * 0.5, Y + line * -1.5, X + line * 1.5, Y + line * -1.5, array);
    list.Add(new Line(X + line * -1.5, Y + line * -1.5, X + line * -0.5, Y + line * -1.5, "top"));
    list.Add(new Line(X + line * -0.5, Y + line * -1.5, X + line * 0, Y + line * -0.5, "top"));
    list.Add(new Line(X + line * 0, Y + line * -0.5, X + line * 0.5, Y + line * -1.5, "top"));
    list.Add(new Line(X + line * 0.5, Y + line * -1.5, X + line * 1.5, Y + line * -1.5, "top"));
}

public void DrawRecursiveShape(Renderer Render, ArrayList lines, int[] array, int i)
{
    if (i == 0) return;
    ArrayList newList = new ArrayList();
    foreach (Line line in lines)
    {
        Point a = line.PointA();
        Point b = line.PointB();
        Point c = line.PointC();
        Point d = line.PointD();
        Point e = line.PointE();

        Render.Draw(a.x, a.y, b.x, b.y, array);
        Render.Draw(b.x, b.y, c.x, c.y, array);
        Render.Draw(c.x, c.y, d.x, d.y, array);
        Render.Draw(d.x, d.y, e.x, e.y, array);
        Render.Erase(b.x, b.y, d.x, d.y, array);
        newList.Add(new Line(a.x, a.y, b.x, b.y, line.orientation));
        newList.Add(new Line(b.x, b.y, c.x, c.y, line.orientation));
        newList.Add(new Line(c.x, c.y, d.x, d.y, line.orientation));
        newList.Add(new Line(d.x, d.y, e.x, e.y, line.orientation));
    }
    i--;
```

```
            DrawRecursiveShape(Render, newList, array, i);
        }

        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;

    }

    static void Main(string[] args)
    {
        ushort pixels = 200;
        int iterations = 3;
        for (int i = 0; i < 5; i++)
        {
            ArrayList newList = new ArrayList();
            pixels *= 2;
            int[] array = new int[10];
            var render = new Renderer("Result" + i, pixels, pixels, 0xffffff);

            var watch = System.Diagnostics.Stopwatch.StartNew();
            render.DrawFirstShape(render, pixels / 2.0, pixels / 2.0, pixels / 5.0 * 3.0, newList, array);
            render.DrawRecursiveShape(render, newList, array, iterations);
            render.Write();
            watch.Stop();

            var elapsedMs = watch.ElapsedMilliseconds;
            Console.WriteLine(pixels + "X" + pixels + " dydis, " + array[0] + " veiksmų skaičius = " + elapsedMs + " ms");
        }
    }
  }
}
```

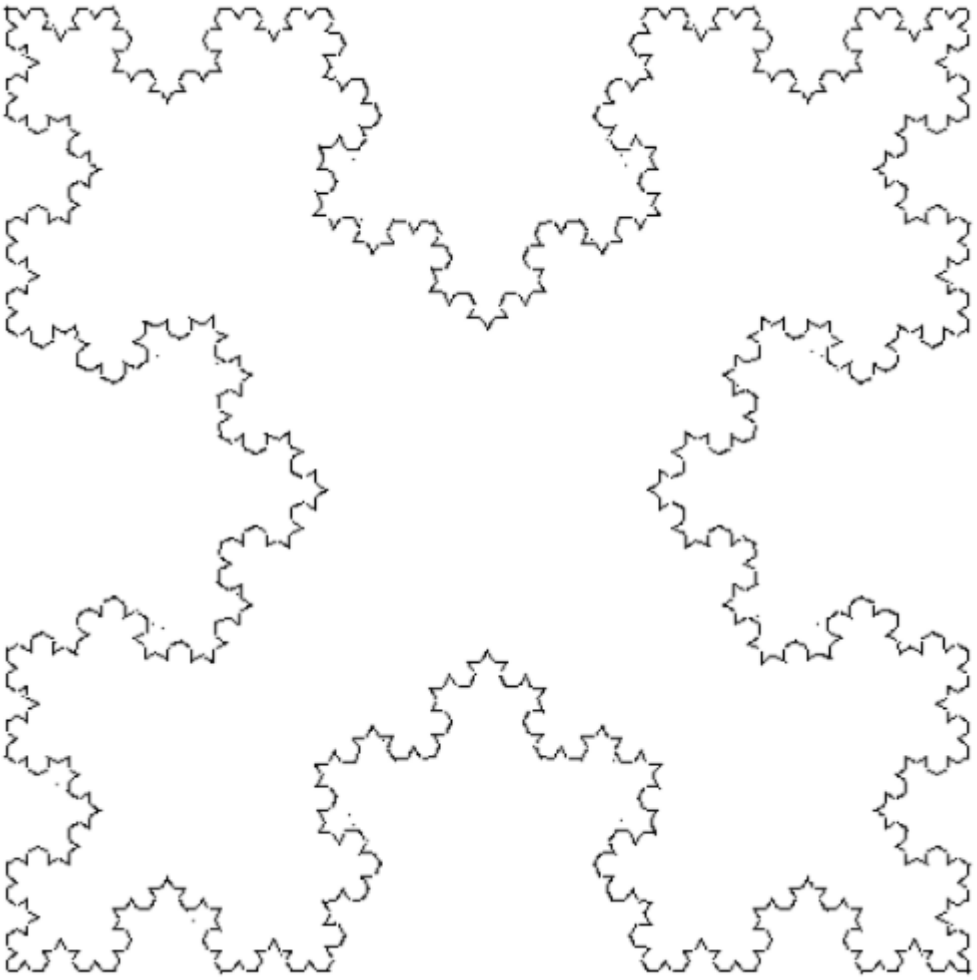
### 1.2.2 Rezultatai

```
400X400 dydis, 12992 veiksmu skaicius = 9 ms
800X800 dydis, 25456 veiksmu skaicius = 5 ms
1600X1600 dydis, 50744 veiksmu skaicius = 12 ms
3200X3200 dydis, 100904 veiksmu skaicius = 46 ms
6400X6400 dydis, 201320 veiksmu skaicius = 225 ms
```

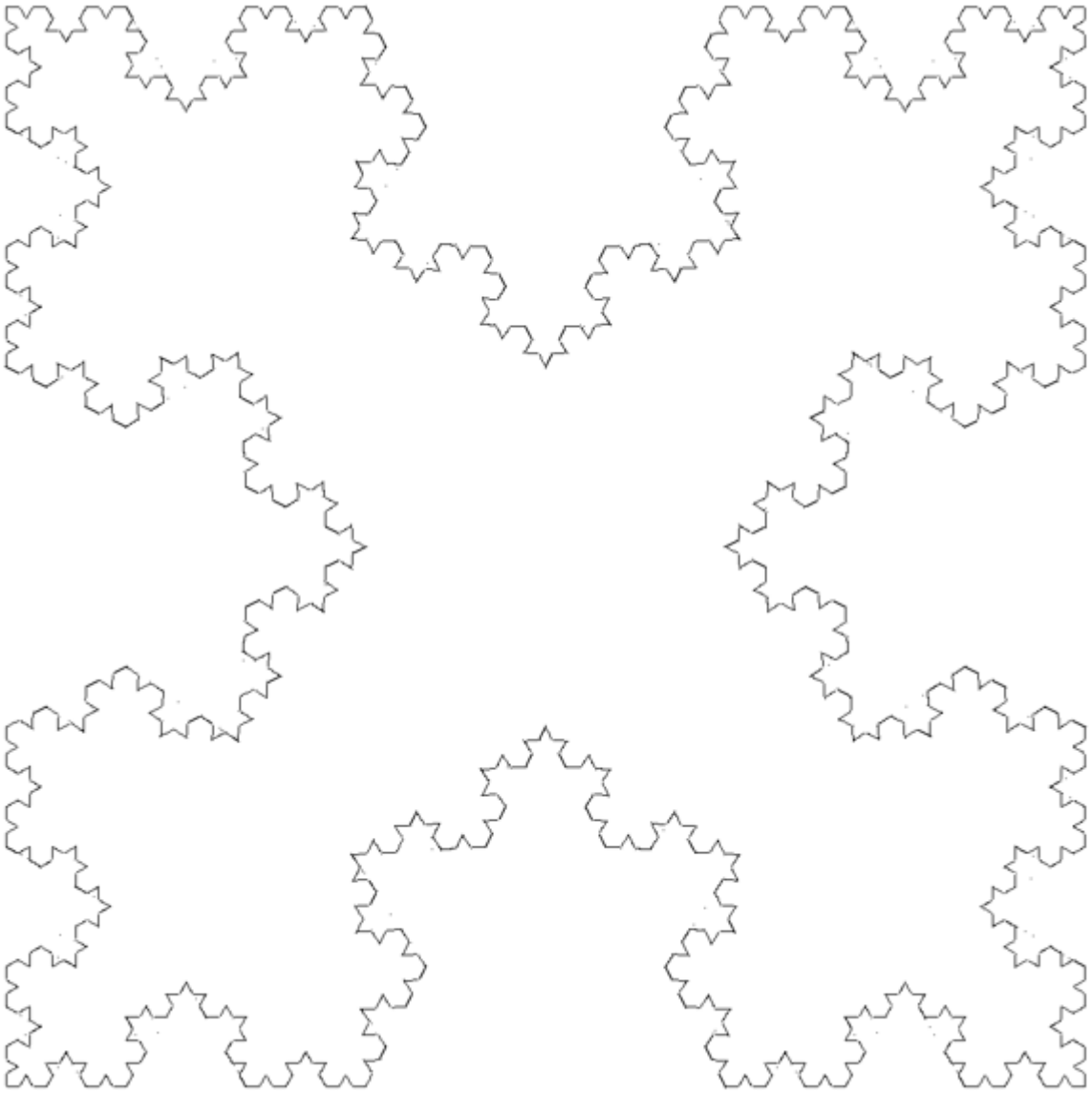
*2 pav. rezultatai consolėje*



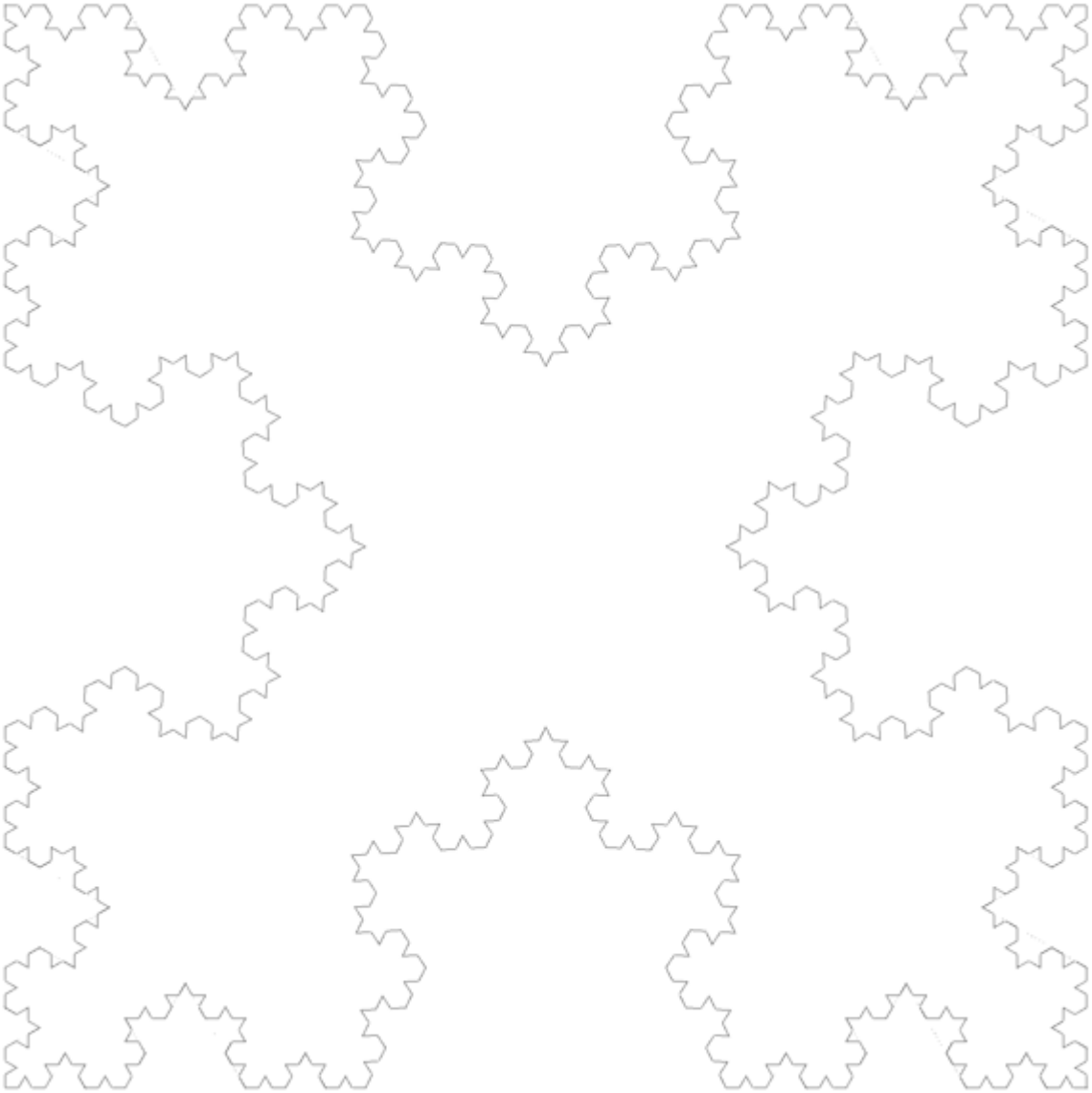
*3 pav. 400x400 raiškos paveikslėlis, užimama vieta 626 KB*



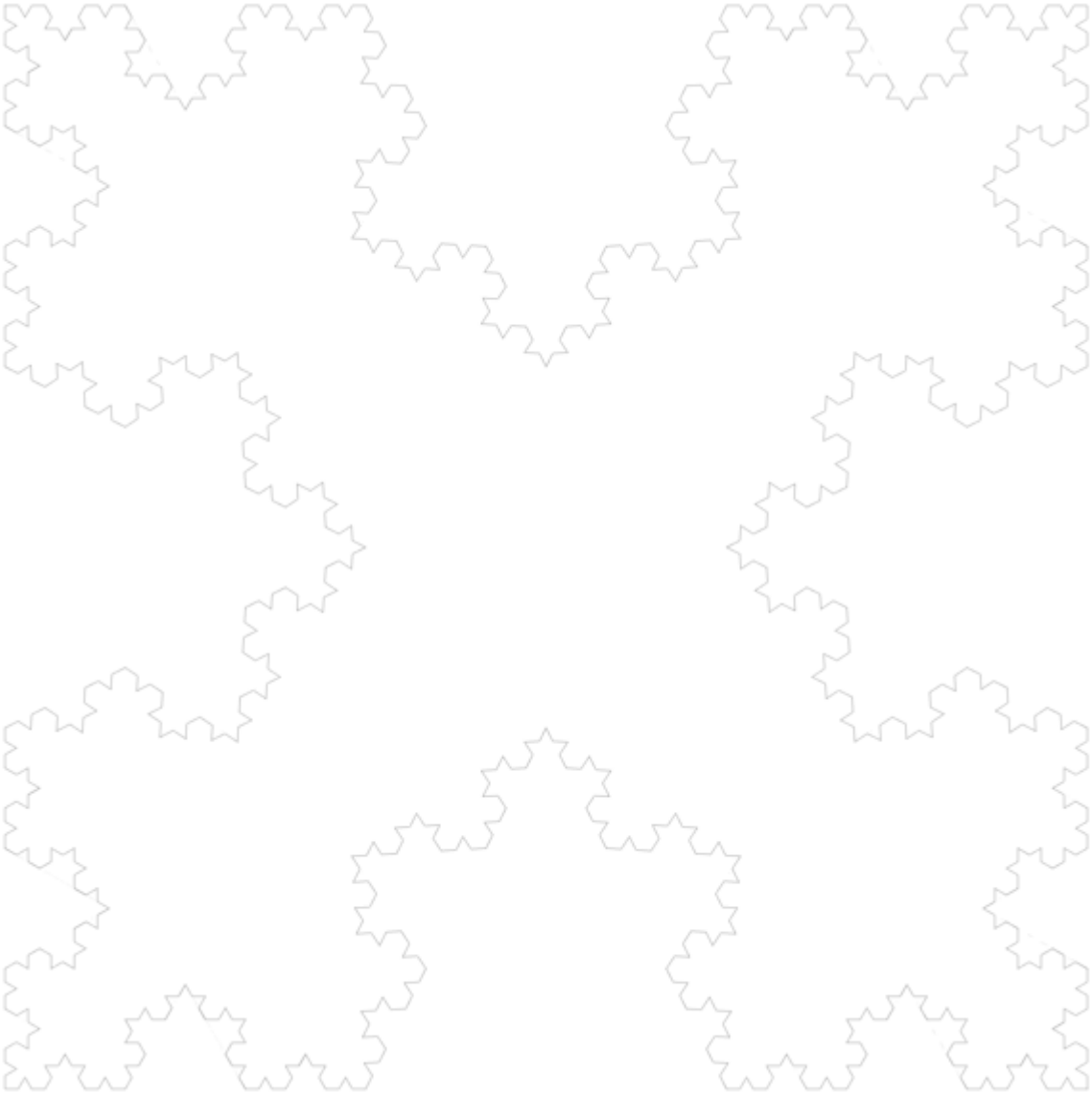
*4 pav. 800x800 raiškos paveikslėlis, užimama vieta 2501KB*



*5 pav. 1600x1600 raiškos paveikslėlis, užimama vieta 10001 KB*



*6 pav. 3200x3200 raiškos paveikslėlis, užimama vieta 40001 KB*



*7 pav. 6400x6400 raiškos paveikslėlis, užimam vieta 160001 KB*

# 2. LD 2

## 2.1. Pirma darbo užduotis

- 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)*.

## 2.2. Pateiktas programinis kodas

```
0 references
public static long methodToAnalysis(int[] arr)
{
    long n = arr.Length;
    long k = n;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] / 7 == 0)
        {
            k -= 2;
        }
        else
        {
            k += 3;
        }
    }
    if (arr[0] > 0)
    {
        for (int i = 0; i < n * n; i++)
        {
            if (arr[0] > 0)
            {
                k += 3;
            }
        }
    }
    return k;
}
```

### 2.2.1 Programinio kodo analizė

### 2.2.2 Lygties sprendimas

| public static long methodToAnalysis(int[] arr) | Laikas | Kiekis |
|---|---|---|
| { | T(n) | |
| long n = arr.Length; | c1 | 1 |
| long k = n; | c2 | 1 |
| for (int i = 0; i < n; i++) | c3 | n+1 |
| { | | |
| if (arr[i] / 7 == 0) | c4 | n |
| { | | |
| k -= 2; | c5 | n |
| } | | |
| else | c6 | n |
| { | | |
| k += 3; | c7 | n |
| } | | |
| } | | |
| if (arr[0] > 0) | c8 | 1 |
| { | | |
| for (int i = 0; i < n * n; i++) | c9 | (n^2) + 1 |
| { | | |
| if (arr[0] > 0) | c10 | n^2 |
| { | | |
| k += 3; | c11 | n^2 |
| } | | |
| } | | |
| } | | |
| return k; | c12 | 1 |
| } | | |

| | |
|---|---|
| Iš viršaus | T(n) = c1 + c2 + c3(n+1) + c4 + c8 = O(n) |
| Iš apačios | T(n) = c1 + c2 + c3(n+1) + c4(n) + c5(n) + c6(n) + c7(n) + c8 + c9((n^2) + 1) + c10(n^2) + c11(n^2) + c12 = O(n^2) |

### 2.3. Pateiktas programinis kodas

```
0 references
public static long methodToAnalysis(int n, int[] arr)
{
    long k = 0;
    Random randNum = new Random();
    for (int i = 0; i < n; i++)
    {
        k += arr[i] + FF3(i, arr);
    }
    return k;
}

3 references
public static long FF3(int n, int[] arr)
{
    if (n > 0 && arr.Length > n && arr[n] > 0)
    {
        return FF3(n - 1, arr) + FF3(n - 3, arr);
    }
    return n;
}
```

### 2.3.1 Kodo analizė

| | | |
|---|---|---|
| public static long methodToAnalysis(int n, int[] arr) | T(n) | |
| { | | |
| long k = 0; | c1 | 1 |
| Random randNum = new Random(); | c2 | 1 |
| for (int i = 0; i < n; i++) | c3 | n+1 |
| { | | |
| k += arr[i] + FF3(i, arr); | FF8(i) | n |
| } | | |
| return k; | c4 | 1 |
| } | | |
| | | |
| public static long FF3(int n, int[] arr) | FF8(n) | |
| { | | |
| if (n > 0 && arr.Length > n && arr[n] > 0) | c5 | 1 |
| { | | |
| return FF3(n - 1, arr) + FF3(n - 3, arr); | FF8(n-1) + FF8(n-2) | |
| } | | |
| return n; | c6 | 1 |
| } | | |

### 2.3.2 Lygties sprendimas

| | |
|---|---|
| Iš viršaus | T(n) = c1 + c2 + c3(n+1) + FF8(1)n + FF8(1) + c4 = O(n) |
| Iš apačios | T(n) = c1 + c2 + c3(n+1) + FF8(n)n + FF8(n) + c4 = O(n^2) |

## 2.4. Antra darbo užduotis

- 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.

## 2.5. Gauta užduotis

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?

### 2.6. Uždavinio sprendimas rekursiniu būdu, programos kodas

```
namespace Lab2AA2
{
    class Program
    {
        static void Main(string[] args)
        {
            //int[] board = { 0, 6, 10, 1, 2, 6, 11, 8, 3, 3, 20}; //Ats 46; 0, -, -,
                              (1*2), 2, -, -, (8*2), 3, 3, 20
            //int[] board = { 0, 10, 20, 100, 300, 10, 20, 5, 6 }; //Ats 62; 0, 10, 20, -
                              , -, (10*2), -, -, (6*2)
            int[] board = new int[5000];
            int[] dp = new int[board.Length]; //Laikomas minimalus taškų skaičius
                                               kiekviename ejime
            Random random = new Random(); //Reikalingas atsitiktiniam tašku generavimui
            for (int i = 1; i < board.Length; i++)
            {
                board[i] = random.Next(1, 100);
            }
            Console.WriteLine("");
            Console.WriteLine("Užduoties sprendimas rekursiniu būdu: " +
                              MinScoreRecursive(board, 0, dp, 0));

        }

        public static int MinScoreRecursive(int[] board, int position, int[] dp, int
                                            dpPos)
        {
            int n = board.Length - 1;  //langelių skaičius
            Boolean canJump = true;

            if (position >= n - 2) canJump = false;

            if (canJump == true)
            {
                if (board[position + 1] <= board[position + 3] * 2)
                {
                    int tempPos = board[position + 1] + board[position + 2];
                    if (board[position + 3] * 2 <= tempPos)
                    {
                        dp[dpPos] = board[position + 3] * 2;
                        position += 3;
                        dpPos++;
                        MinScoreRecursive(board, position, dp, dpPos);
                    }
                    else
                    {
                        dp[dpPos] = board[position + 1];
                        position += 1;
                        dpPos++;
                        MinScoreRecursive(board, position, dp, dpPos);
                    }
                }
                else if (board[position + 1] > board[position + 3] * 2)
                {
                    dp[dpPos] = board[position + 3] * 2;
                    position += 3;
                    dpPos++;
                    MinScoreRecursive(board, position, dp, dpPos);
                }
            }
            else if (canJump == false)
            {
                if (position != n && position == n - 2)
                {
                    dp[dpPos] = board[position + 1];
```

```
                    dp[dpPos + 1] = board[position + 2];
                    return dp.Sum();
                }
                else if (position != n && position == n - 1)
                {
                    dp[dpPos] = board[position + 1];
                    return dp.Sum();
                }
                else return dp.Sum();
            }
            return dp.Sum();
        }
    }
}
```

### 2.6.1 Kodo analizė

| | Laikas | Kiekis |
|---|---|---|
| public static int MinScoreRecursive(int[] board, int position, int[] dp, int dpPos) | T(n) | |
| { | | |
| int n = board.Length - 1; //langelių skaičius | c1 | 1 |
| Boolean canJump = true; | c2 | 1 |
| | | |
| if (position >= n - 2) canJump = false; | c3 | 1 |
| | | |
| if (canJump == true) | c4 | 1 |
| { | | |
| if (board[position + 1] <= board[position + 3] * 2) | c5 | 1 |
| { | | |
| int tempPos = board[position + 1] + board[position + 2]; | c6 | 1 |
| if (board[position + 3] * 2 <= tempPos) | c7 | 1 |
| { | | |
| dp[dpPos] = board[position + 3] * 2; | c8 | 1 |
| position += 3; | c9 | 1 |
| dpPos++; | c10 | 1 |
| MinScoreRecursive(board, position, dp, dpPos); | T(n-1) | |
| } | | |
| else | c11 | 1 |
| { | | |
| dp[dpPos] = board[position + 1]; | c12 | 1 |
| position += 1; | c13 | 1 |
| dpPos++; | c14 | 1 |
| MinScoreRecursive(board, position, dp, dpPos); | T(n-2) | |
| } | | |
| } | | |
| else if (board[position + 1] > board[position + 3] * 2) | c15 | 1 |

| | | |
|---|---|---|
| { | | |
| dp[dpPos] = board[position + 3] * 2; | c16 | 1 |
| position += 3; | c17 | 1 |
| dpPos++; | c18 | 1 |
| MinScoreRecursive(board, position, dp, dpPos); | T(n-3) | |
| } | | |
| } | | |
| else if (canJump == false) | c19 | 1 |
| { | | |
| if (position != n && position == n - 2) | c20 | 1 |
| { | | |
| dp[dpPos] = board[position + 1]; | c21 | 1 |
| dp[dpPos + 1] = board[position + 2]; | c22 | 1 |
| return dp.Sum(); | c23 | 1 |
| } | | |
| else if (position != n && position == n - 1) | c24 | 1 |
| { | | |
| dp[dpPos] = board[position + 1]; | c25 | 1 |
| return dp.Sum(); | c26 | 1 |
| } | | |
| else return dp.Sum(); | c27 | 1 |
| } | | |
| return dp.Sum(); | c28 | 1 |
| } | | |

*8 pav. Rekursinio algoritmo kodo analizė*

### 2.7. Uždavinio sprendimas dinaminio programavimo būdu, programos kodas

```
namespace Lab2AA2
{
    class Program
    {
        static void Main(string[] args)
        {
            //int[] board = { 0, 6, 10, 1, 2, 6, 11, 8, 3, 3, 20}; //Ats 46; 0, -, -, (1*2), 2, -, -, (8*2), 3, 3, 20
            //int[] board = { 0, 10, 20, 100, 300, 10, 20, 5, 6 }; //Ats 62; 0, 10, 20, -, -, (10*2), -, -, (6*2)
            int[] board = new int[5000];
            int[] dp = new int[board.Length]; //Laikomas minimalus taškų skaičius kiekviename ejime
            Random random = new Random(); //Reikalingas atsitiktiniam tašku generavimui
            for (int i = 1; i < board.Length; i++)
            {
                board[i] = random.Next(1, 100);
            }
            Console.WriteLine("Užduoties sprendimas dinaminio programavimo būdu: " + MinScoreDynamic(board));
        }

        public static int MinScoreDynamic(int[] board)
        {
            int n = board.Length - 1;
            int[] dp = new int[n];
            Boolean canJump = true;
            int dpPos = 0;

            for (int i = 0; i < n; i++)
            {
```

```
                if (i >= n - 2) canJump = false;
                if (canJump == true)
                {
                    if (board[i + 1] <= board[i + 3] * 2)
                    {
                        int tempPos = board[i + 1] + board[i + 2];
                        if (board[i + 3] * 2 <= tempPos)
                        {
                            dp[dpPos] = board[i + 3] * 2;
                            i += 2;
                            dpPos++;
                        }
                        else
                        {
                            dp[dpPos] = board[i + 1];
                            dpPos++;
                        }
                    }
                    else if (board[i + 1] > board[i + 3] * 2)
                    {
                        dp[dpPos] = board[i + 3] * 2;
                        i += 2;
                        dpPos++;
                    }
                }
                else if (canJump == false)
                {
                    if (i != n && i == n - 2)
                    {
                        dp[dpPos] = board[i + 1];
                        dp[dpPos + 1] = board[i + 2];
                        return dp.Sum();
                    }
                    else if (i != n && i == n - 1)
                    {
                        dp[dpPos] = board[i + 1];
                        return dp.Sum();
                    }
                }
            }
            return dp.Sum();
        }
```

### 2.7.1 Kodo analizė

| | Laikas | Kiekis |
|---|---|---|
| public static int MinScoreDynamic(int[] board) | T(n) | |
| { | | |
| int n = board.Length - 1; | c1 | 1 |
| int[] dp = new int[n]; | c2 | 1 |
| Boolean canJump = true; | c3 | 1 |
| int dpPos = 0; | c4 | 1 |
| for (int i = 0; i < n; i++) | c5 | n+1 |
| { | | |
| if (i >= n - 2) canJump = false; | c6 | n |
| if (canJump == true) | c7 | n |
| { | | |
| if (board[i + 1] <= board[i + 3] * 2) | c8 | n |
| { | | |
| int tempPos = board[i + 1] + board[i + 2 | c9 | n |
| if (board[i + 3] * 2 <= tempPos) | c10 | n |
| { | | |
| dp[dpPos] = board[i + 3] * 2; | c11 | n |
| i += 2; | c12 | n |
| dpPos++; | c13 | n |
| } | | |
| else | | |
| { | | |
| dp[dpPos] = board[i + 1]; | c14 | n |
| dpPos++; | c15 | n |
| } | | |
| } | | |
| else if (board[i + 1] > board[i + 3] * 2) | c16 | n |
| { | | |

| | | |
|---|---|---|
| dp[dpPos] = board[i + 3] * 2; | c17 | n |
| i += 2; | c18 | n |
| dpPos++; | c19 | n |
| } | c20 | |
| } | | |
| else if (canJump == false) | c21 | n |
| { | | |
| if (i != n && i == n - 2) | c22 | n |
| { | | |
| dp[dpPos] = board[i + 1]; | c23 | n |
| dp[dpPos + 1] = board[i + 2]; | c24 | n |
| return dp.Sum(); | c25 | n |
| } | | |
| else if (i != n && i == n - 1) | c26 | n |
| { | | |
| dp[dpPos] = board[i + 1]; | c27 | n |
| return dp.Sum(); | c28 | n |
| } | | |
| } | | |
| } | | |
| return dp.Sum(); | c29 | 1 |
| } | | |

*9 pav. dinaminio algoritmo kodo analizė*

# 3. LD3

## 3.1. Pirma užduotis

**1 užduoties daliss (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.

### 3.1.1 Gauta užduotis

Duotas reiškinys, kurį sudaro sveiki skaičiai ir du operatoriai '+' ir '*'. Reikia rasti mažiausią reikšmę, gaunamą skirtingai suskliaudus skaičius. Pvz.: reiškinys = "1+2*3+4*5". Mažiausia reikšmė = 1 + (2*3) + (4*5) = 27

### 3.1.2 Programos kodas rekursiniu būdu

```
static List<int> EvalRecursive(string reiskinys)
    {
        List<int> reiksmes = new List<int>();

        int depth = 0;
        for (int i = 0; i < reiskinys.Length; i++)
        {
            if (reiskinys[i] == '(')
                depth++;
            else if (reiskinys[i] == ')')
                depth--;
            else if (depth == 0 && (reiskinys[i] == '+' || reiskinys[i] == '*'))
            {
                List<int> kairiojiReiksmes = EvalRecursive(reiskinys.Substring(0, i));
                List<int> desiniojiReiksmes = EvalRecursive(reiskinys.Substring(i + 1));

                foreach (int kairioji in kairiojiReiksmes)
                {
                    foreach (int desinioji in desiniojiReiksmes)
                    {
                        int reiksme = 0;
                        if (reiskinys[i] == '+')
                            reiksme = kairioji + desinioji;
                        else if (reiskinys[i] == '*')
                            reiksme = kairioji * desinioji;

                        reiksmes.Add(reiksme);
                    }
                }
            }
        }

        if (reiksmes.Count == 0)
            reiksmes.Add(int.Parse(reiskinys));

        return reiksmes;
    }
```

### 3.1.3 Kodo analizė

| static List<int> EvalRecursive(string reiskinys) | Laikas | Kiekis |
|---|---|---|
| { | | |
| List<int> reiksmes = new List<int>(); | c1 | 1 |
| | | |
| int depth = 0; | c2 | 1 |
| for (int i = 0; i < reiskinys.Length; i++) | c3 | n+1 |
| { | | |
| if (reiskinys[i] == '(') | c4 | n |
| depth++; | c5 | n |
| else if (reiskinys[i] == ')') | c6 | n |
| depth--; | c7 | n |
| else if (depth == 0 && (reiskinys[i] == '+' \|\| reiskinys[i] == '*')) | c8 | n |
| { | | |
| List<int> kairiojiReiksmes = EvalRecursive(reiskinys.Substring(0, i)); | T(n) | |
| List<int> desiniojiReiksmes = EvalRecursive(reiskinys.Substring(i + 1)); | T(n+1) | |
| | | |
| foreach (int kairioji in kairiojiReiksmes) | c9 | n(n+1) |
| { | | |
| foreach (int desinioji in desiniojiReiksmes) | c10 | n^2(n+1) |
| { | | |
| int reiksme = 0; | c11 | n^3 |
| if (reiskinys[i] == '+') | c12 | n^3 |
| reiksme = kairioji + desinioji; | c13 | n^3 |
| else if (reiskinys[i] == '*') | c14 | n^3 |
| reiksme = kairioji * desinioji; | c15 | n^3 |
| | | |
| reiksmes.Add(reiksme); | c16 | n^3 |
| } | | |
| } | | |
| } | | |
| } | | |
| | | |
| if (reiksmes.Count == 0) | c17 | 1 |
| reiksmes.Add(int.Parse(reiskinys)); | c18 | 1 |
| | | |
| return reiksmes; | c19 | 1 |
| } | | |

### 3.1.4 Įverčiai

Įverčiai iš viršaus ir apačios yra vienodi algoritme:
T(n) = O(n^3)

### 3.1.5 Programos kodas dinaminio programamvimo būdu

```
static int Eval(string reiskinys)
    {
        Stack<int> operandai = new Stack<int>();
        Stack<char> operatoriai = new Stack<char>();

        for (int i = 0; i < reiskinys.Length; i++)
        {
            if (reiskinys[i] == ' ')
                continue;

            if (char.IsDigit(reiskinys[i]))
            {
                int skaičius = 0;
                while (i < reiskinys.Length && char.IsDigit(reiskinys[i]))
                {
                    skaičius = skaičius * 10 + (reiskinys[i] - '0');
```

```
                i++;
            }
            operandai.Push(skaičius);
            i--;
        }
        else if (reiskinys[i] == '+')
        {
            while (operatoriai.Count > 0 && operatoriai.Peek() != '(')
            {
                int desinysis = operandai.Pop();
                int kairysis = operandai.Pop();
                char operatorius = operatoriai.Pop();
                operandai.Push(ApvalintiRezultatą(kairysis, operatorius, desinysis));
            }
            operatoriai.Push(reiskinys[i]);
        }
        else if (reiskinys[i] == '*')
        {
            operatoriai.Push(reiskinys[i]);
        }
        else if (reiskinys[i] == '(')
        {
            operatoriai.Push(reiskinys[i]); //O(1)
        }
        else if (reiskinys[i] == ')')
        {
            while (operatoriai.Count > 0 && operatoriai.Peek() != '(')
            {
                int desinysis = operandai.Pop();
                int kairysis = operandai.Pop();
                char operatorius = operatoriai.Pop();
                operandai.Push(ApvalintiRezultatą(kairysis, operatorius, desinysis));
            }
            operatoriai.Pop(); // Pašaliname '(' iš operatorių steko O(1)
        }
    }

    while (operatoriai.Count > 0)
    {
        int desinysis = operandai.Pop();
        int kairysis = operandai.Pop();
        char operatorius = operatoriai.Pop();
        operandai.Push(ApvalintiRezultatą(kairysis, operatorius, desinysis));
    }

    return operandai.Pop();
}

static int ApvalintiRezultatą(int kairysis, char operatorius, int desinysis)
{
    if (operatorius == '+')
    {
        return kairysis + desinysis;
    }
    else if (operatorius == '*')
    {
        return kairysis * desinysis;
    }
    return 0;
}
```

### 3.1.6 Kodo analize

| static int Eval(string reiskinys) | Laikas | Kiekis |
|---|---|---|
| { | | |
| Stack<int> operandai = new Stack<int>(); | c1 | 1 |
| Stack<char> operatoriai = new Stack<char>(); | c2 | 1 |
| | | |
| for (int i = 0; i < reiskinys.Length; i++) | c3 | n+1 |
| { | | |
| if (reiskinys[i] == ' ') | c4 | n+1 |
| continue; | c5 | n+1 |
| | | |
| if (char.IsDigit(reiskinys[i])) | c6 | n |
| { | | |
| int skaičius = 0; | c7 | n |
| while (i < reiskinys.Length && char.IsDigit(reiskinys[i])) | c8 | n^2 |
| { | | |
| skaičius = skaičius * 10 + (reiskinys[i] - '0'); | c9 | n^2 |
| i++; | c10 | n^2 |
| } | | |
| operandai.Push(skaičius); | c11 | n |
| i--; | c12 | n |
| } | | |
| else if (reiskinys[i] == '+') | c13 | n |
| { | | |
| while (operatoriai.Count > 0 && operatoriai.Peek() != '(') | c14 | n^2 |
| { | | |
| int desinysis = operandai.Pop(); | c15 | n^2 |
| int kairysis = operandai.Pop(); | c16 | n^2 |
| char operatorius = operatoriai.Pop(); | c17 | n^2 |
| operandai.Push(ApvalintiRezultatą(kairysis, operatorius, desinysis)); | c18 | n^2 |
| } | | |
| operatoriai.Push(reiskinys[i]); | c19 | n |
| } | | |
| else if (reiskinys[i] == '*') | c20 | n |
| { | | |
| operatoriai.Push(reiskinys[i]); | c21 | n |
| } | | |
| else if (reiskinys[i] == '(') | c22 | n |
| { | | |
| operatoriai.Push(reiskinys[i]); //O(1) | c23 | n |
| } | | |
| else if (reiskinys[i] == ')') | c24 | n |
| { | | |
| while (operatoriai.Count > 0 && operatoriai.Peek() != '(') | c25 | n^2 |
| { | | |
| int desinysis = operandai.Pop(); | c26 | n^2 |
| int kairysis = operandai.Pop(); | c27 | n^2 |
| char operatorius = operatoriai.Pop(); | c28 | n^2 |
| operandai.Push(ApvalintiRezultatą(kairysis, operatorius, desinysis)); | c29 | n^2 |
| } | | |
| operatoriai.Pop(); // Pašaliname '(' iš operatorių steko O(1) | c30 | n^2 |
| } | | |
| } | | |

| | | |
|---|---|---|
| while (operatoriai.Count > 0) | c31 | n+1 |
| { | | |
| int desinysis = operandai.Pop(); | c32 | n |
| int kairysis = operandai.Pop(); | c33 | n |
| char operatorius = operatoriai.Pop(); | c34 | n |
| operandai.Push(ApvalintiRezultatą(kairysis, operatorius, desinysis)); | c35 | n |
| } | | |
| | | |
| return operandai.Pop(); | c36 | 1 |
| } | | |
| | | |
| static int ApvalintiRezultatą(int kairysis, char operatorius, int desinysis) | | |
| { | | |
| if (operatorius == '+') | c37 | 1 |
| { | | |
| return kairysis + desinysis; | c38 | 1 |
| } | | |
| else if (operatorius == '*') | c39 | 1 |
| { | | |
| return kairysis * desinysis; | c40 | 1 |
| } | | |
| return 0; | c41 | 1 |
| } | | |

### 3.1.7 Įverčiai

| | | |
|---|---|---|
| Iš viršaus | Kai visi operatoriai ‚*‘ | T(n) = c1+...+c12+c20+c21+c31+...+c41= O(n) |
| Iš apačios | Kai visi operatoriai ‚+‘ | T(n) = c1+...+c19+c31+...+c41= O(n^2) |

## 3.2. Antra darbo dalis

**2-3 užduotys (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*

### 3.2.1 2 užduotis

### 3.2.2 Algoritmo Kodas

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

```
            {
                for (int j = 0; j < n * n / 2; j++)
                {
                    k -= 2;
                }
                for (int j = 0; j < n * n / 2; j++)
                {
                    k += 3;
                }
            }
        }
        return k;
    }
```

### 3.2.3 Nelygiagretinta

#### Kodo analize

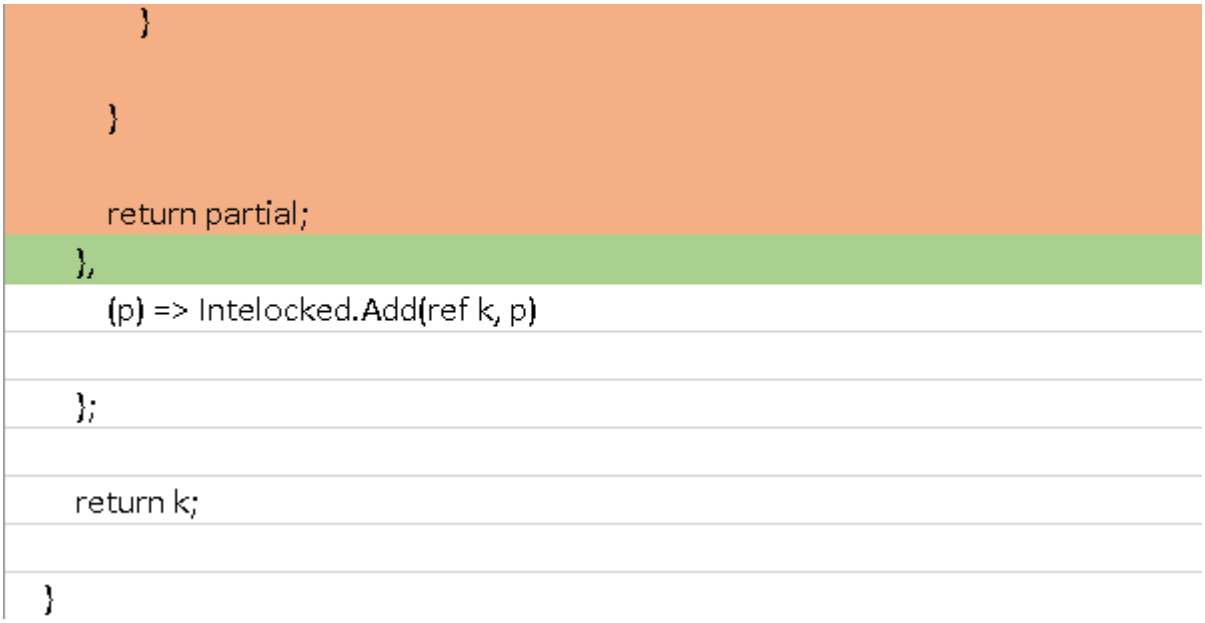
| public static long methodToAnalysis (int[] arr) | Laikas | Kiekis |
|---|---|---|
| { | | |
| long n = arr.Length; | c1 | 1 |
| long k = n; | c2 | 1 |
| for (int i = 0; i < n; i++) | c3 | n+1 |
| { | | |
| if (arr[i] > 0) | c4 | n |
| { | | |
| for (int j = 0; j < n * n / 2; j++) | c5 | ((n^3)/2) + 1 |
| { | | |
| k -= 2; | c6 | (n^3)/2 |
| } | | |
| for (int j = 0; j < n * n / 2; j++) | c7 | ((n^3)/2) + 1 |
| { | | |
| k += 3; | c8 | (n^3)/2 |
| } | | |
| } | | |
| } | | |
| return k; | c9 | 1 |
| } | | |

| | | |
|---|---|---|
| Iš viršaus | Neteisinga Arr[i] > 0 | T(n) = c1+c2+c3+c9 = O(n) |
| Iš apačios | Teisinga Arr[i] > 0 | T(n) = c1+...+c9 = O(n^3) |

### 3.2.4 Lygiagretinta

***Kodo analize***

```
public static long methodToAnalysis(int[] arr)

  {

     long n = arr.Length;

     long k = n;

     Parallel.For<long>(0, n, () => 0, (1, loop, partial)) =>
     {

       if (arr[i] > 0)

       {

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

          {

             k -= 2;

          }

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

          {

             k += 3;
```




### 3.2.5 3 užduotis

```
public static long methodToAnalysis (int n, int[] arr)
    {
       long k = 0;
       for (int i = 0; i < n; i++)
       {
          k += k;
          k += FF8(i, arr);
       }
       k += FF8(n, arr);
       return k;
    }

    public static long FF8(int n, int[] arr)
    {
       if (n > 1&& arr.Length > n && arr[0] < 0)
       {
          return FF8(n - 2, arr) + FF8(n / n, arr);
       }
       return n;
```

```
    }
```

### 3.2.6 Nelygiagretinta

#### Kodo analize

| public static long methodToAnalysis (int n, int[] arr) | Laikas | Kiekis |
|---|---|---|
| { | | |
| long k = 0; | c1 | 1 |
| for (int i = 0; i < n; i++) | c2 | n+1 |
| { | | |
| k += k; | c3 | n |
| k += FF8(i, arr); | FF8(i) | n |
| } | | |
| k += FF8(n, arr); | FF8(n) | 1 |
| return k; | c4 | 1 |
| } | | |
| | | |
| public static long FF8(int n, int[] arr) | FF8(n) | |
| { | | |
| if (n > 1&& arr.Length > n && arr[0] < 0) | c5 | 1 |
| { | | |
| return FF8(n - 2, arr) + FF8(n / n, arr); | FF8(n-2) | |
| } | | |
| return n; | c6 | 1 |
| } | | |

| | | |
|---|---|---|
| Iš viršaus | Teisinga if (n > 1&& arr.Length > n && arr[0] < 0) | T(n)=c1+…+c4+nFF8(1)+FF8(1)=Omega(n) |
| Iš apačios | Neteisinga if (n > 1&& arr.Length > n && arr[0] < 0) | T(n)=c1+…+c4+nFF8(n)+FF8(n)=O(n^2) |

### 3.2.7 Lygiagretinta

#### Kodo analize

```
public static long ParallelMethodToAnalysis(int n, int[] arr)
{
   Long k = 0;
   Parallel.For<long>(0, n, () => 0, (1, loop, partial) =>
   {
      partial += k;
      partial += FF8(l, arr);
      return partial;
   },
   (p) => Interlocked.Add(ref k, p)
   };
   k += FF8(n, arr);
   return k;
);

public static long FF8(int n, int[] arr)
{
   if (n > 1 && arr.Length > n && arr[0] < 0)
```

```
    {
        return FF8(n - 2, arr) + FF8(n / n, arr);
    }
    return n;
}
```

Įverčiai vienodi kaip ir nelygiagretintoje.