**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Modulio P170B400 „Algoritmų sudarymas ir analizė“**

Laboratorinio darbo aprašas (ataskaita)

**Pirmas laboratorinis darbas**

**Dėstytojas**

lekt. MAKACKAS Dalius

**Studentas**

Matas Palujanskas IFF-1/8

**KAUNAS, 2023**

**TURINYS**

[1. UŽDUOTIS 3](#_Toc127992145)

[2. PIRMA DALIS. REKURENTINĖS LYGTYS 4](#_Toc127992146)

[2.1. Pirma lygtis 4](#_Toc127992147)

[2.1.1. Programinio kodo analizė 4](#_Toc127992148)

[2.1.2. Rekurentinės lygties sprendimas 5](#_Toc127992149)

[2.1.3. Eksperimentinis tyrimas 6](#_Toc127992150)

[2.2. Antra lygtis 7](#_Toc127992151)

[2.2.1. Programinio kodo analizė 7](#_Toc127992152)

[2.2.2. Rekurentinės lygties sprendimas 8](#_Toc127992153)

[2.2.3. Eksperimentinis tyrimas 10](#_Toc127992154)

[2.3. Trečia lygtis 11](#_Toc127992155)

[2.3.1. Programinio kodo analizė 11](#_Toc127992156)

[2.3.2. Rekurentinės lygties sprendimas 12](#_Toc127992157)

[2.3.3. Eksperimentinis tyrimas 13](#_Toc127992158)

[3. ANTRA DALIS. BMP FORMATO BYLOS 14](#_Toc127992159)

[3.1. Programinis kodas 14](#_Toc127992160)

[3.2. Programos formuojami rezultatai 18](#_Toc127992161)

# UŽDUOTIS

**1 užduoties dalis:** 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).

Individualaus užduoties varianto lygtys:

1. 𝑇(𝑛) =
2. 𝑇(𝑛) =
3. 𝑇(𝑛) =

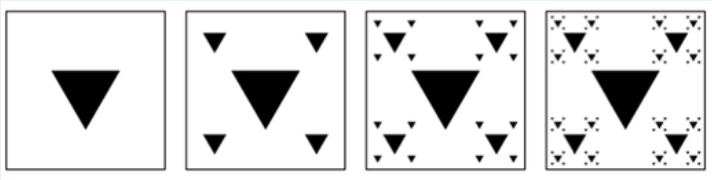
**2 užduoties dalis:** Naudojant rekursiją ir nenaudojant grafinių bibliotekų sudaryti nurodytos struktūros BMP formato ( gautą atlikus užduoties pasirinkimo testą):

* 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 (našumo testus: vykdymo laiką ir veiksmų skaičių). (1 balas)

Individuali užduotis:



# PIRMA DALIS. REKURENTINĖS LYGTYS

## Pirma lygtis

𝑇(𝑛) =

### Programinio kodo analizė

T(k), kur k = n + 1

|  |  |  |
| --- | --- | --- |
| public static void Rec1(int[] A, int start, int end) | Laikas | Kiekis |
| { |  |  |
| if (end <= 0) | c1 | 1 |
| { |  |  |
| return; | c2 | 1 |
| } |  |  |
|  |  |  |
| Rec1( A, start, end / 6); |  | 1 |
| Rec1( A, start, end / 6); |  | 1 |
|  |  |  |
| for (int i = start; i < end; i++) | c3 | end – start + 1 = k |
| { |  |  |
| for (int j = start; j < end; j++) | c3 | k(k – 1)= |
| { |  |  |
| for (int y = start; y < end; y++) | c3 | ()( |
| { |  |  |
| for (int k = start; k < end; k++) | c3 | ()( |
| { |  |  |
| counter++; | c4 | ()( |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

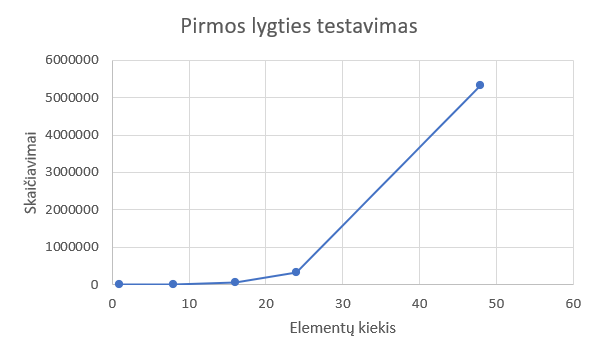
### Rekurentinės lygties sprendimas

Paveikslėlis, kuriame yra žinutė, rašymo lenta

Automatiškai sugeneruotas aprašymas

Panaudotas teoremos 3 atvejis.

### Eksperimentinis tyrimas



Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

## Antra lygtis

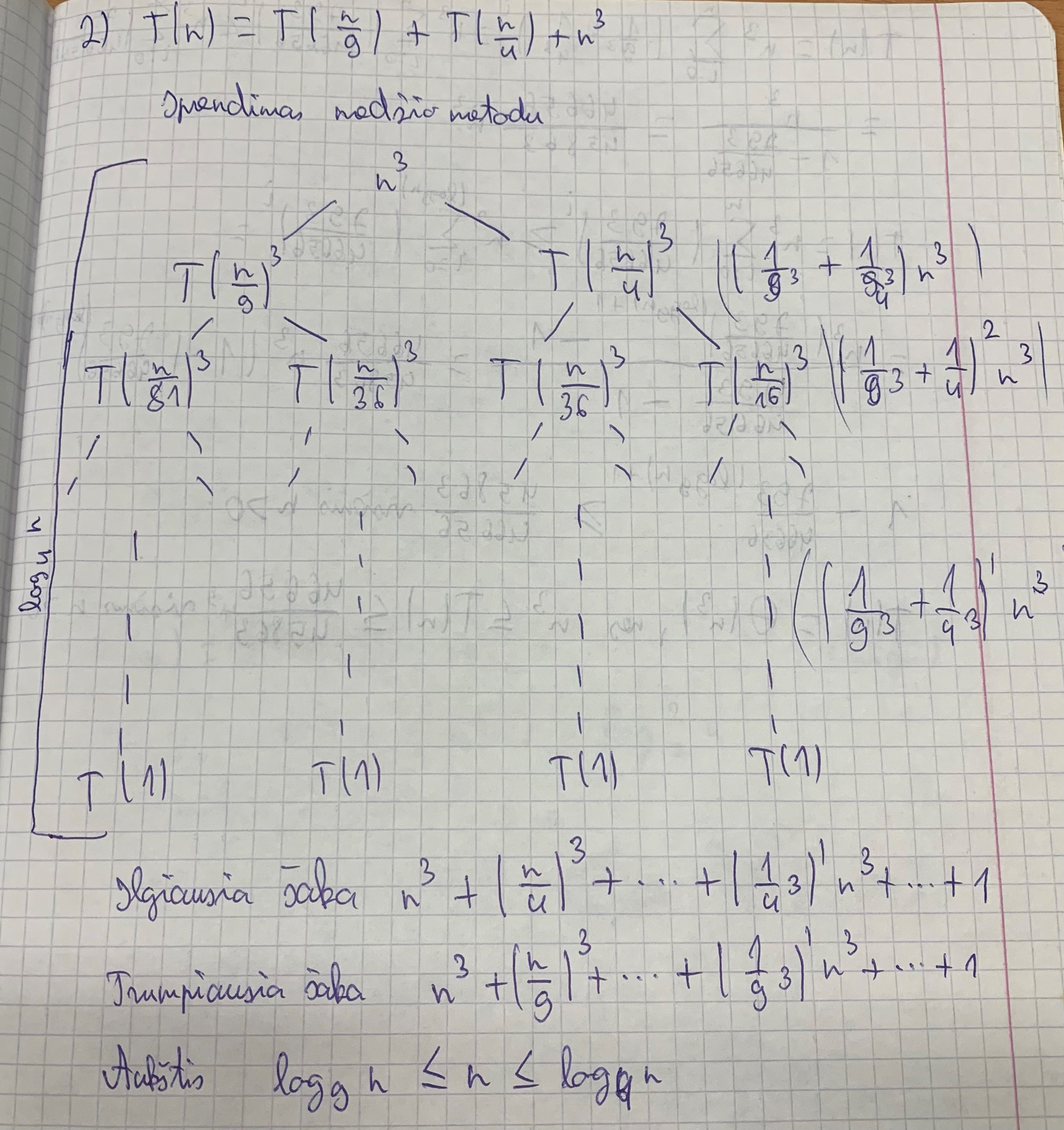
𝑇(𝑛) =

### Programinio kodo analizė

T(k), kur k = n + 1

|  |  |  |
| --- | --- | --- |
| public static void Rec2(int[] A, int start, int end) | Laikas | Kiekis |
| { |  |  |
| if (end <= 0) | c1 | 1 |
| return; | c2 | 1 |
|  |  |  |
| Rec2(A, start, end / 4); |  | 1 |
| Rec2(A, start, end / 9); |  | 1 |
|  |  |  |
| for(int i = 0; i < end; i++) | c3 |  |
| { |  |  |
| for(int j = 0; j < end; j++) | c3 |  |
| { |  |  |
| for(int k = 0; k < end; k++) | c3 |  |
| { |  |  |
| counter++; | c4 |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

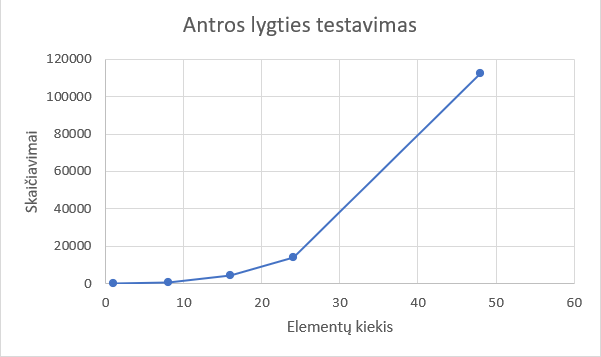
### Rekurentinės lygties sprendimas



Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

### Eksperimentinis tyrimas



Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

## Trečia lygtis

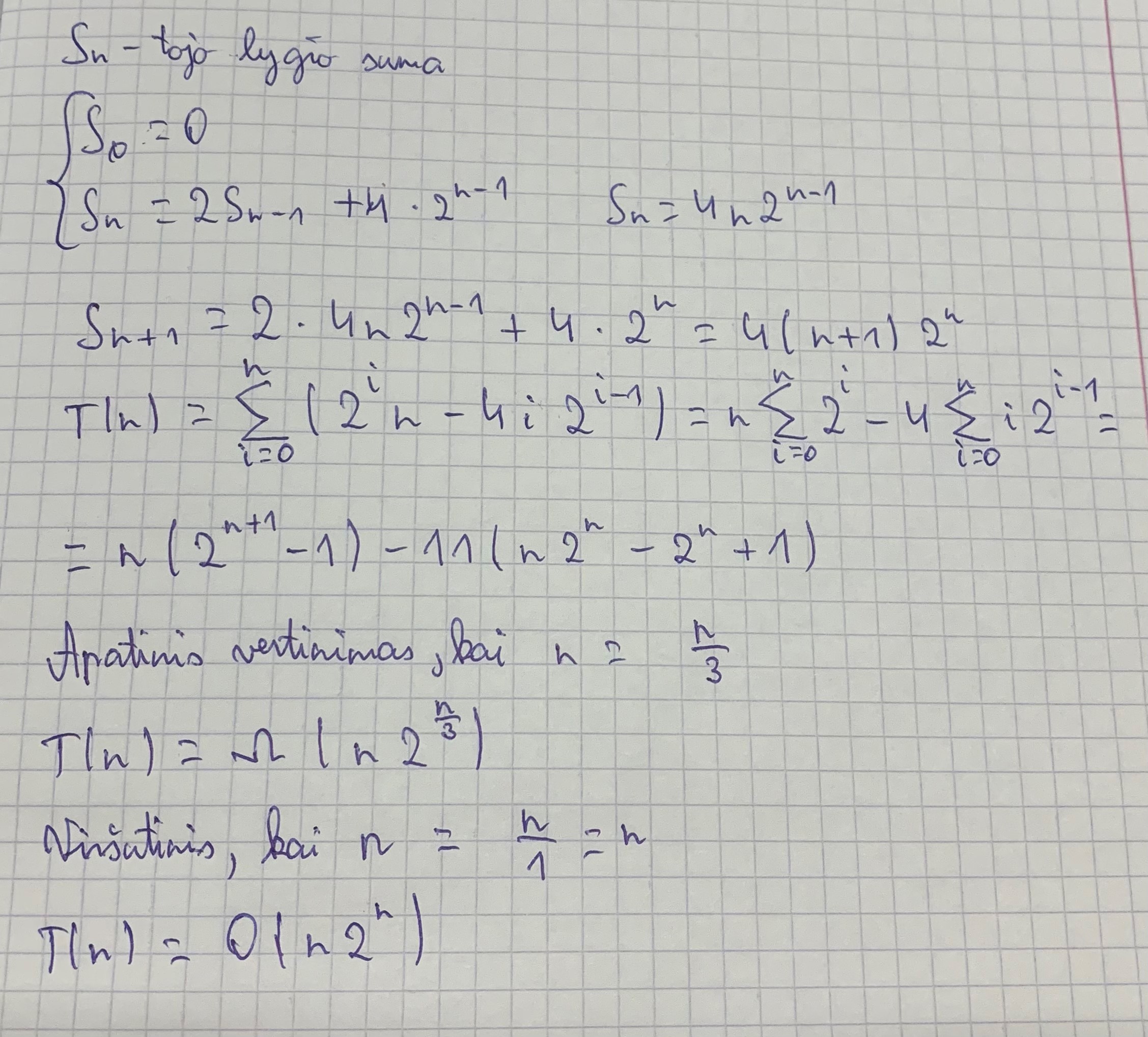
𝑇(𝑛) =

T(k), kur k = n + 1

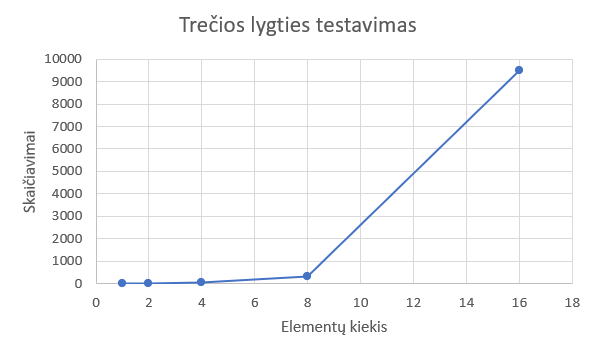
### Programinio kodo analizė

|  |  |  |
| --- | --- | --- |
| public static void Rec3(int[] A, int start, int end) | Laikas | Kiekis |
| { |  |  |
| if (end <= 0) | c1 | 1 |
| return; | c2 | 1 |
|  |  |  |
| Rec3(A, start, end - 3); | T(n - 3) | 1 |
| Rec3(A, start, end - 1); | T(n – 1) | 1 |
|  |  |  |
| for (int i = 0; i < end; i++) | c3 | n – 1 + 1 + 1 = n + 1 |
| { |  |  |
| counter++; | c4 | n |
| } |  |  |
| } |  |  |

### Rekurentinės lygties sprendimas



### Eksperimentinis tyrimas



Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

# ANTRA DALIS. BMP FORMATO BYLOS

## Programinis kodas

using System;

using System.IO;

namespace Laboras

{

internal class Renderer

{

private const int pictureSize = 3000;

static void Main(string[] args)

{

Renderer renderer = new Renderer("Triangle", pictureSize, pictureSize, 0xFFFFFF);

// X Y coords, Width, Height, Color

renderer.InitializeFractal();

renderer.Print();

}

// 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";

}

private void InitializeFractal()

{

double xW = 33 \* (pictureSize / 113);

double yW = 71 \* (pictureSize / 113);

double xE = 82 \* (pictureSize / 113);

double yE = 71 \* (pictureSize / 113);

double xS = 57 \* (pictureSize / 113);

double yS = 30 \* (pictureSize / 113);

DrawFullTriangle(xW, yW, xE, yE, xS, yS);

int recursionLength = -1;

double xLength = xE - xW;

while (xLength > 3)

{

xLength = xLength \* ((double)15 / (double)49);

recursionLength++;

}

MoveInDifferentDirections(recursionLength, xW, yW, xE, yE, xS, yS);

}

//Moving triangle in different directions

private void MoveInDifferentDirections(int recursionLength, double \_xW, double \_yW, double \_xE, double \_yE, double \_xS, double \_yS)

{

if (recursionLength <= 0) return;

double xW\_SW = \_xW - (\_xE - \_xW) \* ((double)21 / (double)49);

double yW\_SW = \_yW - (\_yW - \_yS) \* ((double)48 / (double)42);

double xE\_SW = xW\_SW + (\_xE - \_xW) \* ((double)15 / (double)49);

double xS\_SW = (xE\_SW + xW\_SW) / (double)2;

double yS\_SW = \_yW - (\_yW - \_yS) \* ((double)61 / (double)42);

double xW\_NE = \_xW + (\_xE - \_xW) \* ((double)57 / (double)49);

double yW\_NE = \_yW + (\_yW - \_yS) \* ((double)27 / (double)42);

double xE\_NE = xW\_NE + (\_xE - \_xW) \* ((double)15 / (double)49);

double xS\_NE = (xE\_NE + xW\_NE) / (double)2;

double yS\_NE = \_yW + (\_yW - \_yS) \* ((double)14 / (double)42);

//Second picture

DrawFullTriangle(xW\_SW, yW\_SW, xE\_SW, yW\_SW, xS\_SW, yS\_SW); // South West Triangle

DrawFullTriangle(xW\_NE, yW\_NE, xE\_NE, yW\_NE, xS\_NE, yS\_NE); // North East Triangle

DrawFullTriangle(xW\_NE, yW\_SW, xE\_NE, yW\_SW, xS\_NE, yS\_SW); // South East Triangle

DrawFullTriangle(xW\_SW, yW\_NE, xE\_SW, yW\_NE, xS\_SW, yS\_NE); // North West Triangle

//Third picture

//MoveInAllDirections(recursionLength - 2, xW\_SW, yW\_SW, xE\_SW, yW\_SW, xS\_SW, yS\_SW); // South West Triangle

//MoveInAllDirections(recursionLength - 2, xW\_NE, yW\_NE, xE\_NE, yW\_NE, xS\_NE, yS\_NE); // North East Triangle

//MoveInAllDirections(recursionLength - 2, xW\_NE, yW\_SW, xE\_NE, yW\_SW, xS\_NE, yS\_SW); // South East Triangle

//MoveInAllDirections(recursionLength - 2, xW\_SW, yW\_NE, xE\_SW, yW\_NE, xS\_SW, yS\_NE); // North West Triangle

//Fourth picture

MoveInDifferentDirections(recursionLength - 1, xW\_SW, yW\_SW, xE\_SW, yW\_SW, xS\_SW, yS\_SW); // South West Triangle

MoveInDifferentDirections(recursionLength - 1, xW\_NE, yW\_NE, xE\_NE, yW\_NE, xS\_NE, yS\_NE); // North East Triangle

MoveInDifferentDirections(recursionLength - 1, xW\_NE, yW\_SW, xE\_NE, yW\_SW, xS\_NE, yS\_SW); // South East Triangle

MoveInDifferentDirections(recursionLength - 1, xW\_SW, yW\_NE, xE\_SW, yW\_NE, xS\_SW, yS\_NE); // North West Triangle

}

//Drawing triangle

public void DrawFullTriangle(double X0, double Y0, double X1, double Y1, double X2, double Y2,

double Precision = 0.5, uint Color = 0)

{

double Length = Math.Sqrt(Math.Pow(X1 - X0, 2) + Math.Pow(Y1 - Y0, 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)

{

XRun += XStep;

YRun += YStep;

DrawLine(XRun, YRun, X2, Y2, Precision, Color);

}

}

//Drawing lines

public void DrawLine(double X0, double Y0, double X1, double Y1, double Precision = 0.5, 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)

{

XRun += XStep;

YRun += YStep;

SetPixel(XRun, YRun, Color);

}

}

//Setting pixels

private void SetPixel(double X, double Y, uint Color)

{

int Pixel = GetPixel(X, Y);

if (Pixel < 0)

return;

Buffer[Pixel] = Color;

}

//Getting pixels

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;

}

//Printing picture to file

public void Print()

{

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;

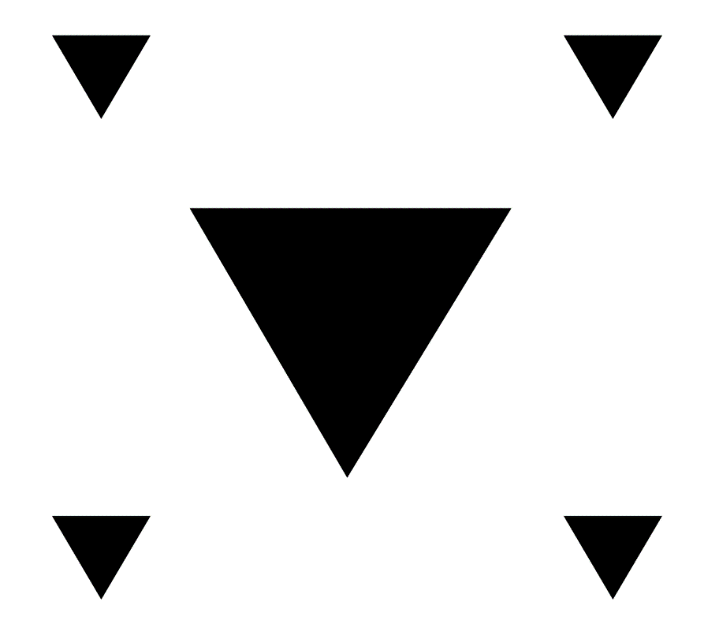
}

}

## Programos formuojami rezultatai

Paveikslėlis, kuriame yra strėlė

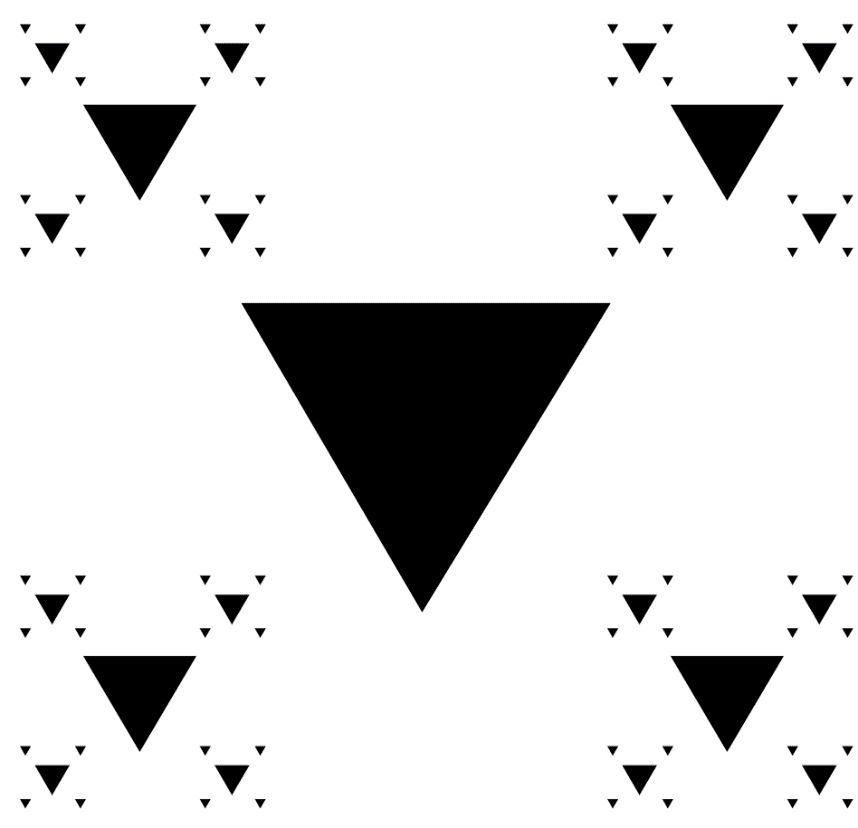
Automatiškai sugeneruotas aprašymas



pav.

|  |  |
| --- | --- |
| pav. |  |

pav.



|  |  |
| --- | --- |
|  |  |

Paveikslėlis, kuriame yra strėlė

Automatiškai sugeneruotas aprašymas

pav.