KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

P170B400 Algoritmų sudarymas ir analizė (P170B400)

Laboratorinių darbų ataskaita

Atliko:
IFF-1/9 gr. studentas
Nedas Liaudanskis
2023 m. vasario 22 d.

Priėmė:
Doc. Pilkauskas vytautas
Lekt. Kraujalis tadas
Doc. Čalnerytė dalia
Lekt. Makackas dalius

KAUNAS 2023

1. UŽDUOTIS

1 Dalis:

- 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:

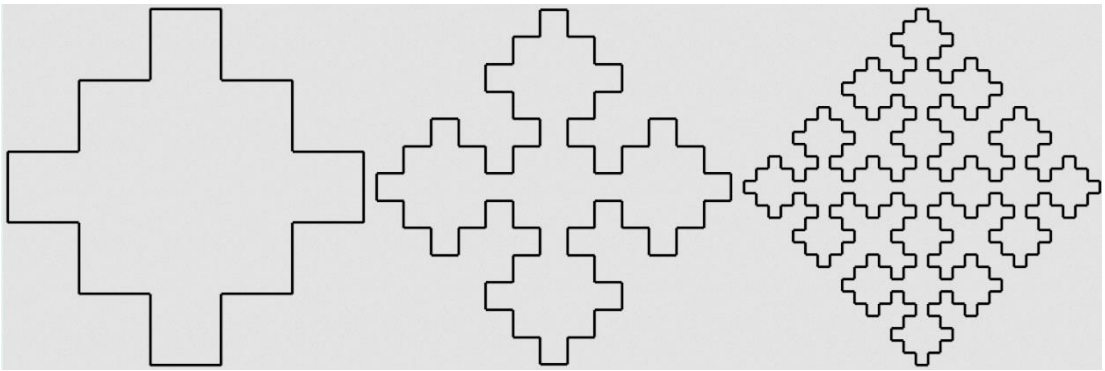
1. $T(n) = 2 * T\left(\frac{n}{10}\right) + n^2$
2. $T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{n}{8}\right) + n^2$
3. $T(n) = T(n-7) + T(n-6) + n$

2 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. (1 balas)

Individualaus užduoties varianto struktūra:



1. Pirma Dalis. Rekurentinės lygtys

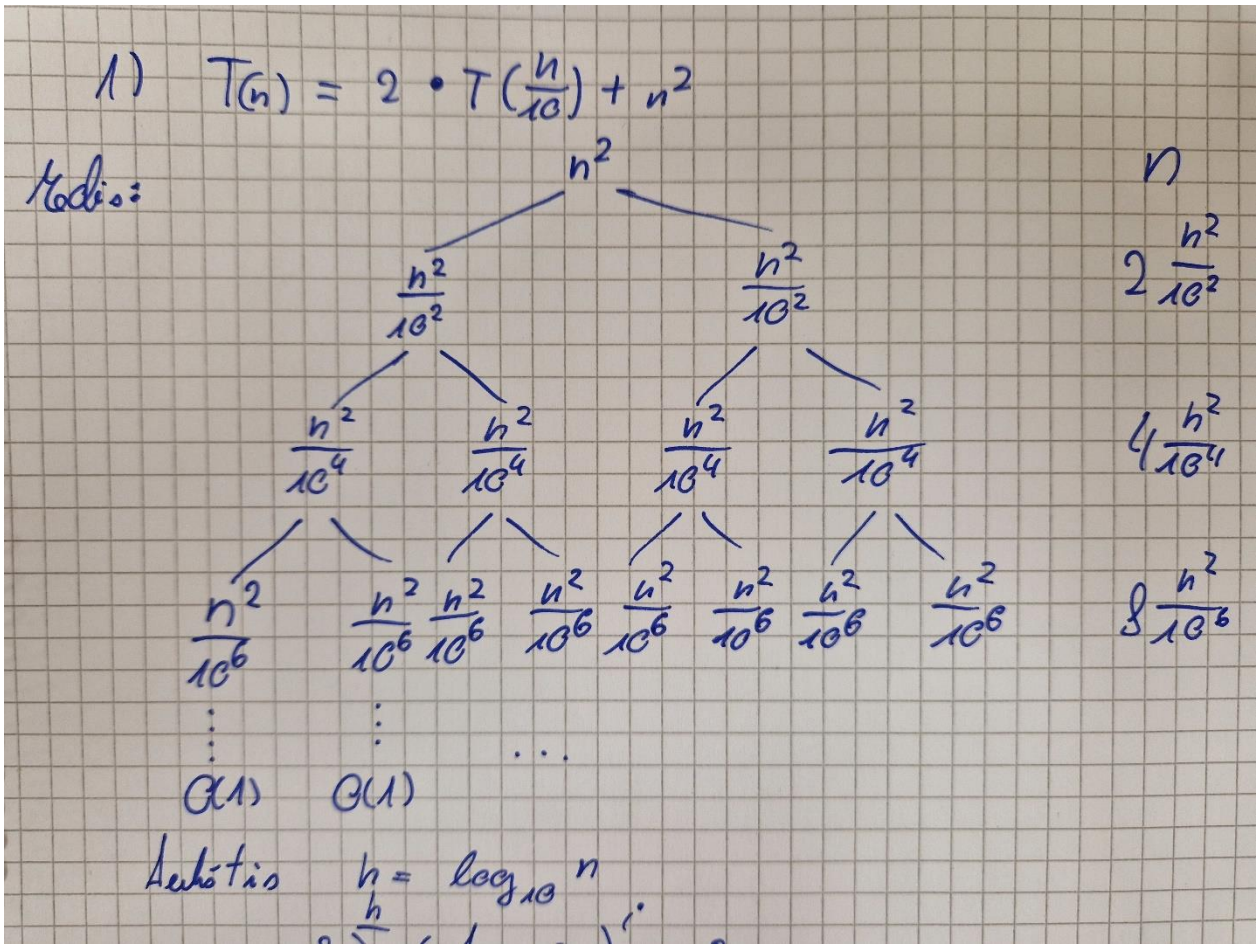
Lygtis T1

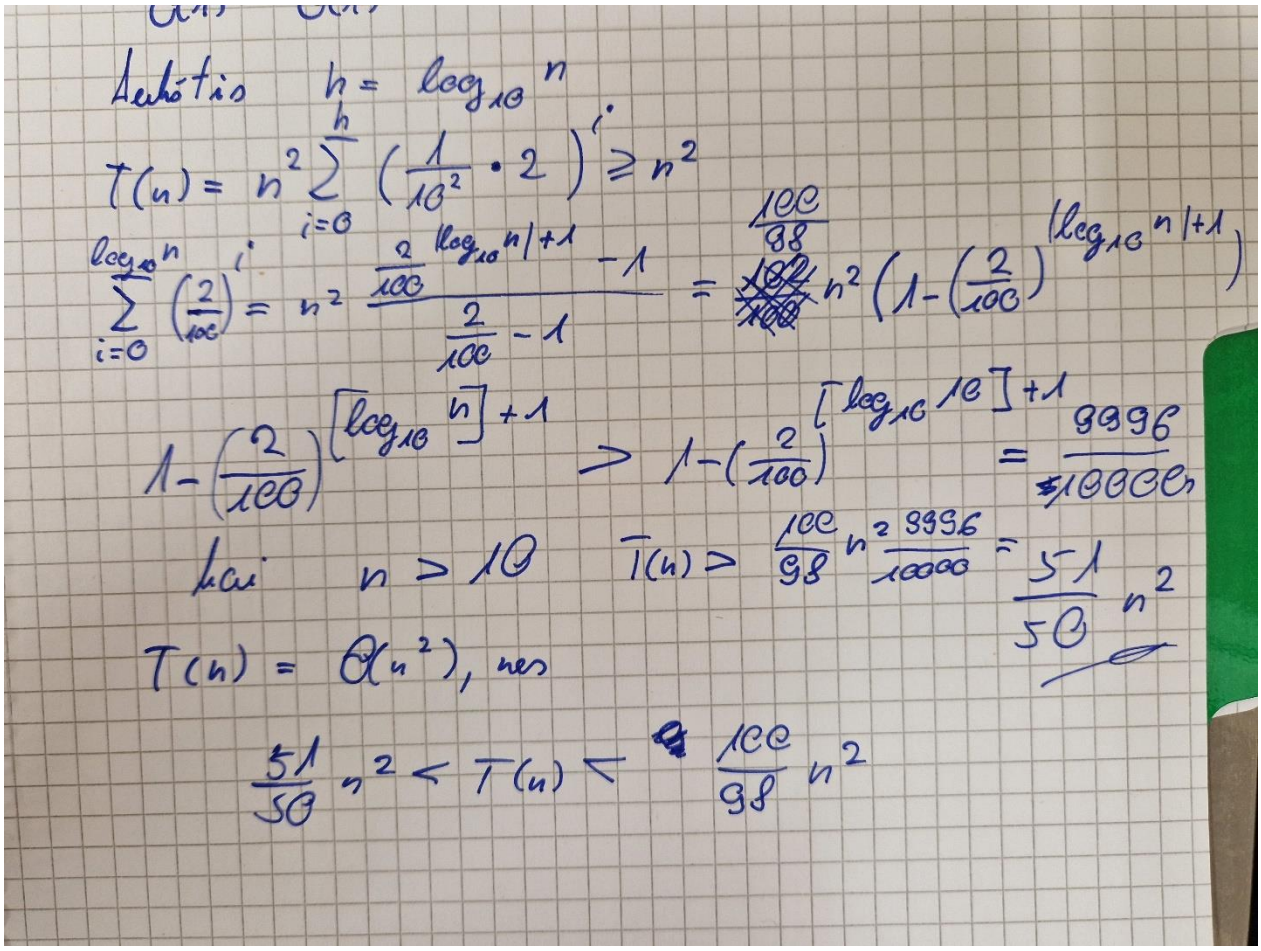
1. $T(n) = 2 * T\left(\frac{n}{10}\right) + n^2$

Programinio kodo analizė

```
//T(n) = 2 * T(n/10) + n^2
        public static void T1(int[] A, int n)
        {


            if (n < 11)                             // c1 | 1
                return;                             // c2 | 1
            T1(A, n / 10);                          // T(n/10) | 1
            T1(A, n / 10);                          // T(n/10) | 1

            for (int i = 0; i < n*n; i++)           // c3 + c4 | 1
            {
                A[i] = 0;                           // c5 + c4 | n^2
            }

        }
        //        { c1 + c2 , n < 10
        // T(k) = {
        //        { T(n/10) + T(n/10) + c1 + c3 + c4 + (c5 + c4) * n^2, n >= 10
```

Rekurentinės lygties sprendimas





Eksperimentinis tyrimas

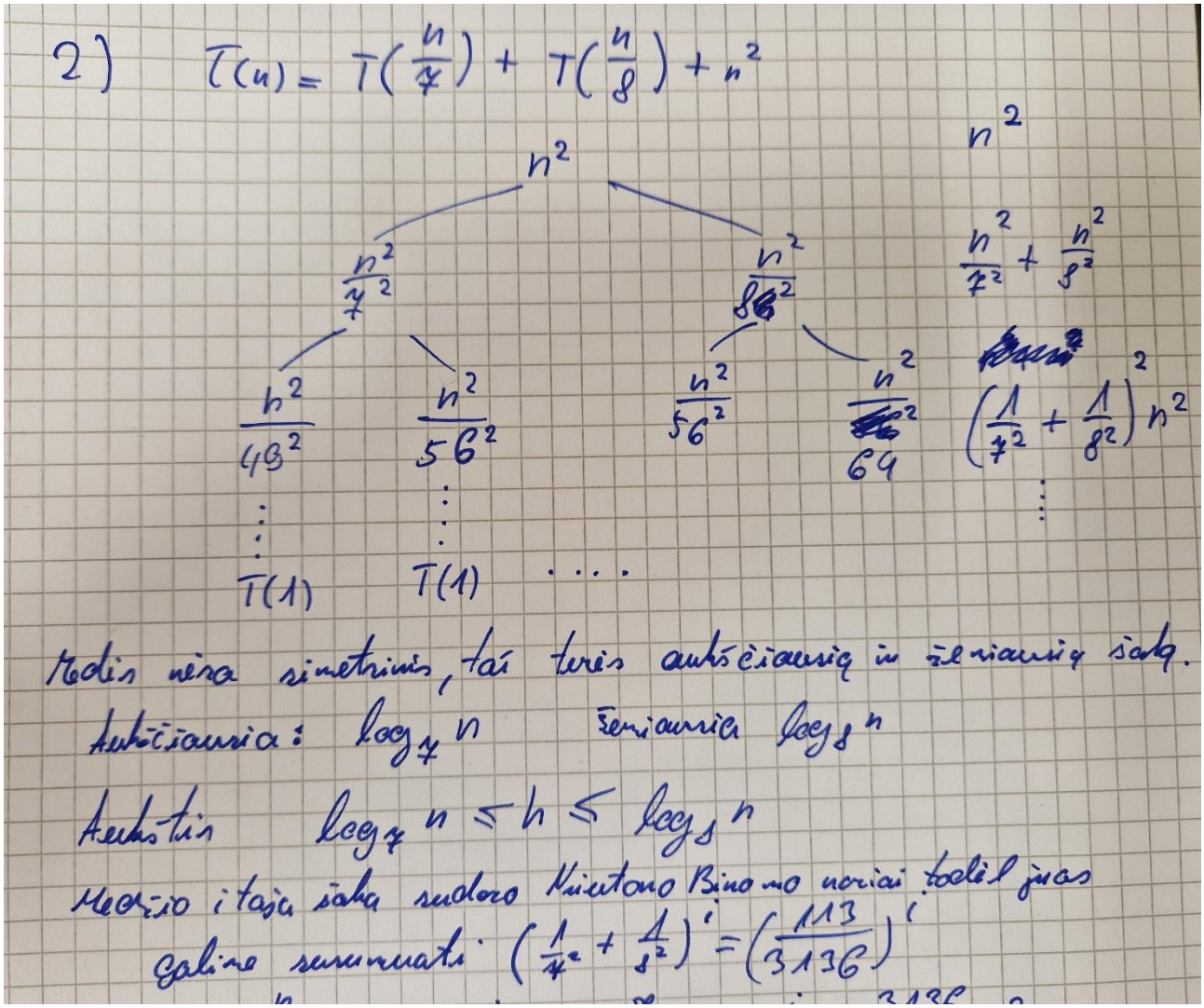
Lygtis T2

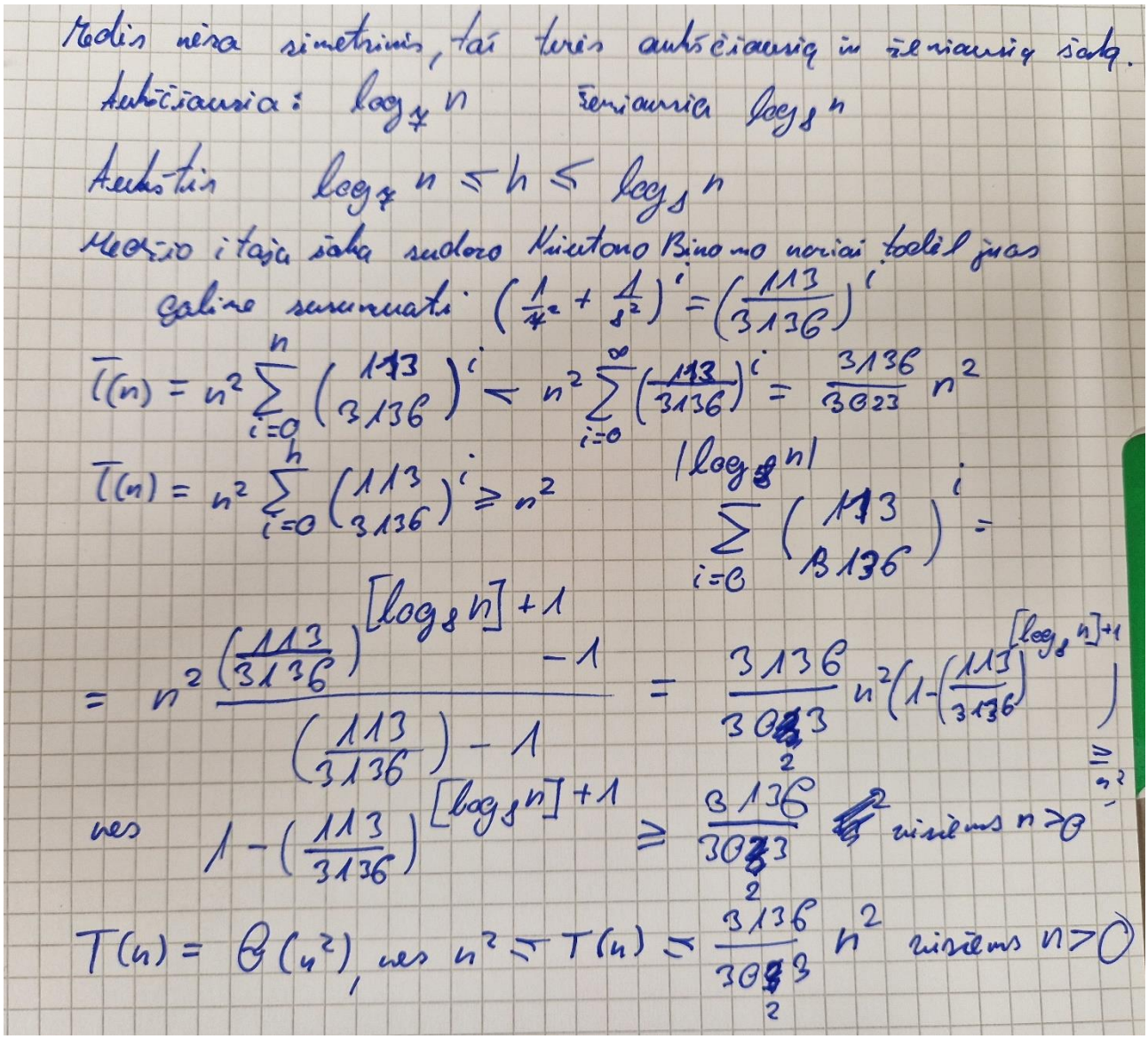
1. $T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{n}{8}\right) + n^2$

Programinio kodo analizė

```
//T(n)=T(n/7)+ T(n/8)+ n^2
        public static void T2(int[] A, int n)
        {
            if (n == 0) return;                 // c1 + c2 | 1
            T2(A, n / 7);                       // T(n/7) | 1
            T2(A, n / 8);                       // T(n/8) | 1

            for (int i = 0; i < n*n; i++)       // c3 + c4 | 1
            {
                A[i] = 0;                       // c4 + c5 | n^2
            }
        }
        //        { c1 + c2 , n < 1
        // T(k) = {
        //        { T(n/7) + T(n/8) + c1 + c3 + c4 + (c5 + c4) * n^2, n >= 1
```

Rekurentinės lygties sprendimas





Eksperimentinis tyrimas

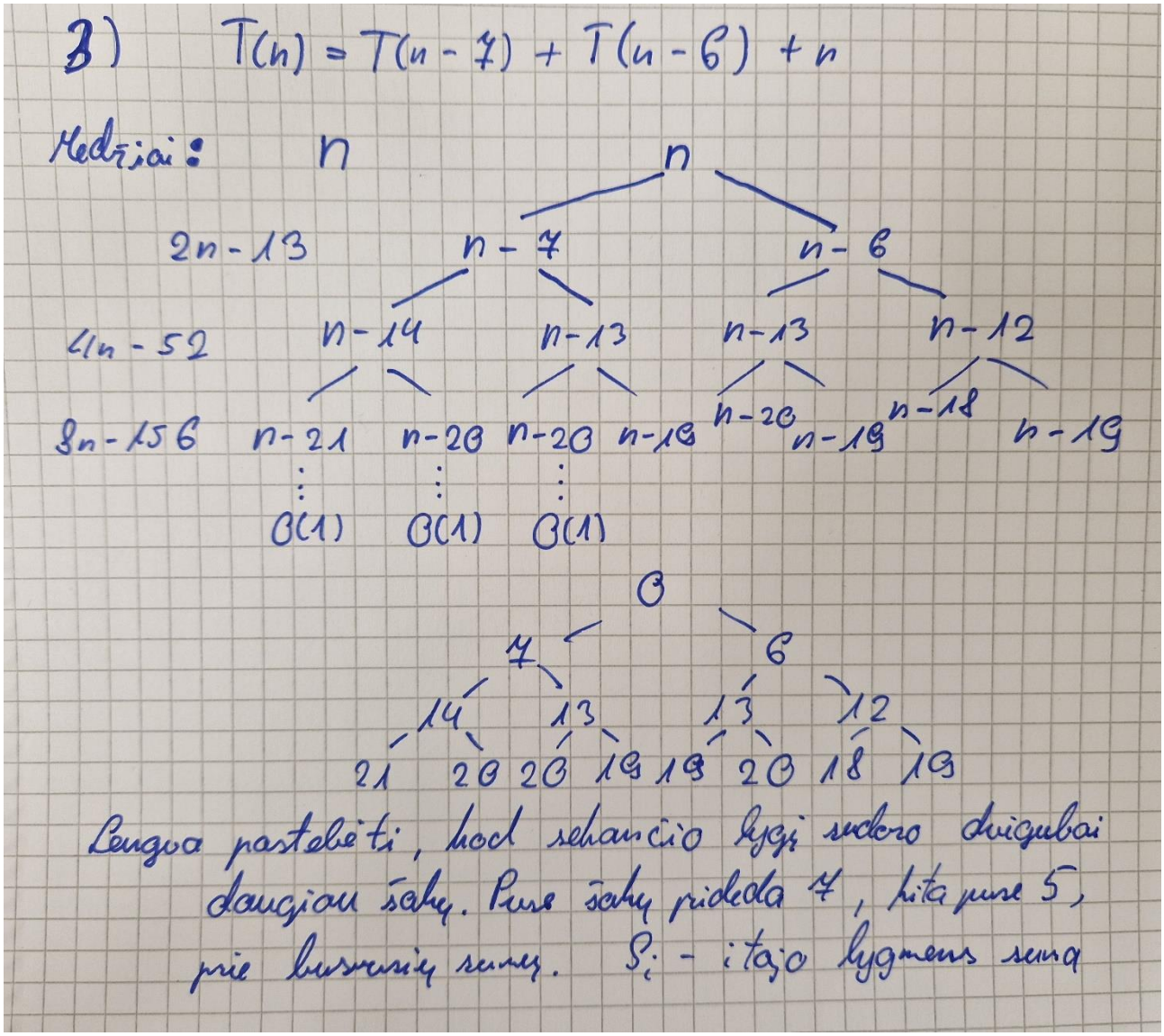
Lygtis T3

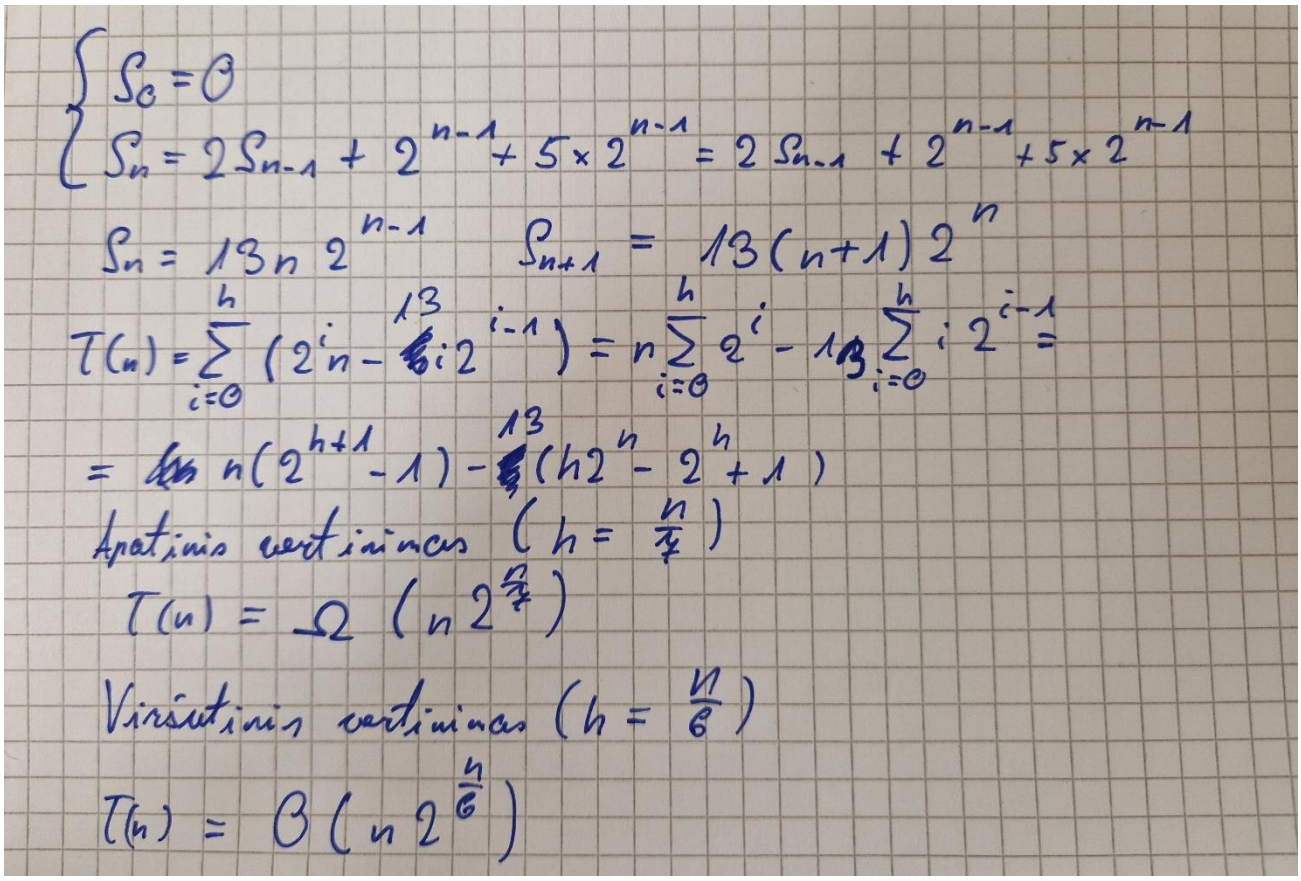
1. $T(n) = T(n-7) + T(n-6) + n$

Programinio kodo analizė

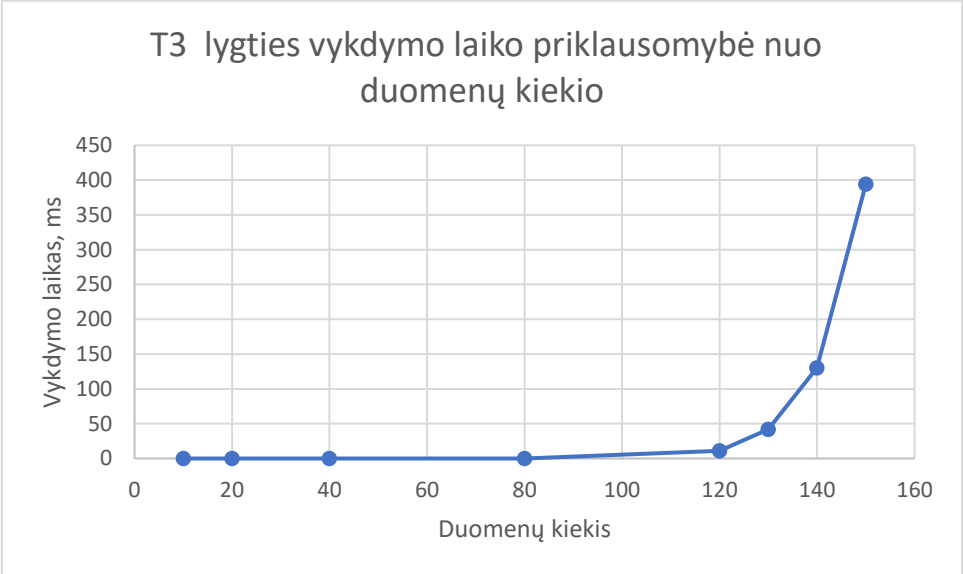
```
//T(n)=T(n-7)+ T(n-6) + n
        public static void T3(int[] A, int n)
        {
            if (n < 7) return;                      // c1 + c2 | 1
            T3(A, n - 7);                           // T(n - 7) | 1
            T3(A, n - 6);                           // T(n - 6) | 1

            for (int i = 0; i < n; i++)             // c3 + c4 | 1
            {
                A[i] = 0;                           // c4 + c5 | n
            }
        }

        //        { c1 + c2 , n <= 7
        // T(k) = {
        //        { T(n - 7) + T(n - 6) + c1 + c3 + c4 + (c5 + c4) * n, n > 7
```

Rekurentinės lygties sprendimas





Eksperimentinis tyrimas



2. Antra Dalis fraktalo formavimas su BMP

Programinis kodas

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


namespace BMP
{
   class Program
   {
   enum FaceMask
   {
   None = 0,
   Top = 1 << 0,
   Left = 1 << 1,
   Bottom = 1 << 2,
   Right = 1 << 3,
   Mask_All = Top | Left | Right | Bottom
   }

   static void Main(string[] args)
   {
   ushort Size = 1000;
   Renderer Render = new Renderer("Result", Size, Size, 0xffffffff);



   void DrawFigure(Renderer render , double x0, double y0, double x1, double y1, double lenght,
FaceMask Mask)
   {
   lenght = lenght / 5;

   if((Mask & FaceMask.Top) != 0)
                {
   render.DrawLine(x0 - lenght * 0.5, y0 + lenght * 2.5, x0 + lenght * 0.5, y0 + lenght * 2.5);
   }


   if ((Mask & FaceMask.Right) != 0)
   {
   render.DrawLine(x0 + lenght * 2.5, y0 + lenght * 0.5, x0 + lenght * 2.5, y0 - lenght * 0.5);
   }



   if ((Mask & FaceMask.Left) != 0)
   {
   render.DrawLine(x0 - lenght * 2.5, y0 - lenght * 0.5, x0 - lenght * 2.5, y0 + lenght * 0.5);
   }


   if ((Mask & FaceMask.Bottom) != 0)
   {
   render.DrawLine(x0 - lenght * 0.5, y0 - lenght * 2.5, x0 + lenght * 0.5, y0 - lenght * 2.5);
   }


   render.DrawLine(x0 - lenght * 0.5, y0 + lenght * 1.5, x0 - lenght * 0.5, y0 + lenght * 2.5);
```

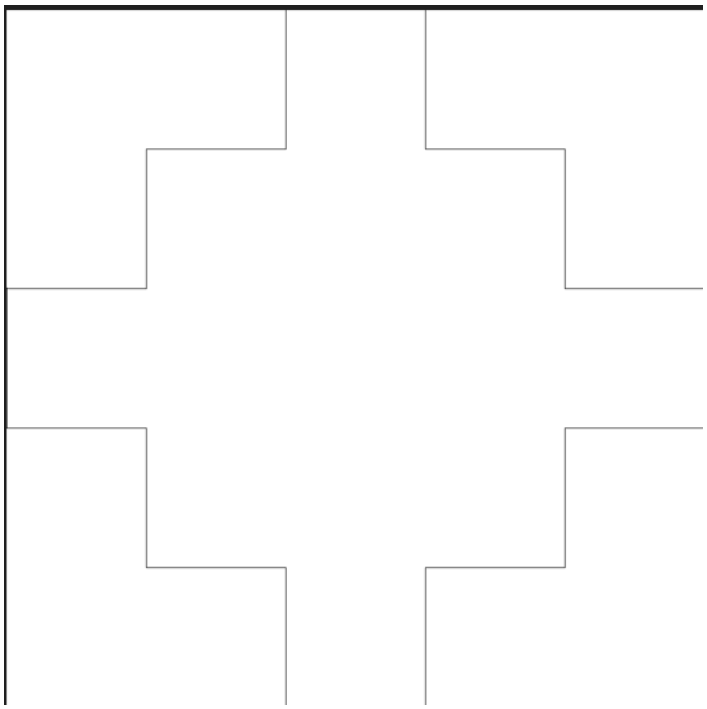
```
render.DrawLine(x0 + lenght * 0.5, y0 + lenght * 1.5, x0 + lenght * 0.5, y0 + lenght * 2.5);
render.DrawLine(x0 - lenght * 1.5, y0 + lenght * 1.5, x0 - lenght * 1.5, y0 + lenght * 0.5);
render.DrawLine(x0 - lenght * 1.5, y0 + lenght * 1.5, x0 - lenght * 0.5, y0 + lenght * 1.5);
render.DrawLine(x0 - lenght * 1.5, y0 - lenght * 1.5, x0 - lenght * 0.5, y0 - lenght * 1.5);

render.DrawLine(x0 + lenght * 1.5, y0 - lenght * 1.5, x0 + lenght * 0.5, y0 - lenght * 1.5);
render.DrawLine(x0 + lenght * 1.5, y0 + lenght * 1.5, x0 + lenght * 0.5, y0 + lenght * 1.5);
render.DrawLine(x0 - lenght * 1.5, y0 - lenght * 1.5, x0 - lenght * 1.5, y0 - lenght * 0.5);

render.DrawLine(x0 + lenght * 1.5, y0 + lenght * 1.5, x0 + lenght * 1.5, y0 + lenght * 0.5);
render.DrawLine(x0 + lenght * 1.5, y0 - lenght * 1.5, x0 + lenght * 1.5, y0 - lenght * 0.5);
render.DrawLine(x0 - lenght * 1.5, y0 - lenght * 0.5, x0 - lenght * 2.5, y0 - lenght * 0.5);
render.DrawLine(x0 - lenght * 1.5, y0 + lenght * 0.5, x0 - lenght * 2.5, y0 + lenght * 0.5);
render.DrawLine(x0 + lenght * 1.5, y0 - lenght * 0.5, x0 + lenght * 2.5, y0 - lenght * 0.5);
render.DrawLine(x0 + lenght * 1.5, y0 + lenght * 0.5, x0 + lenght * 2.5, y0 + lenght * 0.5);
render.DrawLine(x0 - lenght * 0.5, y0 - lenght * 1.5, x0 - lenght * 0.5, y0 - lenght * 2.5);

render.DrawLine(x0 + lenght * 0.5, y0 - lenght * 1.5, x0 + lenght * 0.5, y0 - lenght * 2.5);

}

void Funkcija(Renderer Render ,double x, double y, int depth, double scale, FaceMask l)
{
if (depth == 0)
{
DrawFigure(Render, x, y, x, y, scale, l);
return;

}


Funkcija(Render, x, y, depth - 1, scale / 3, FaceMask.None);

if ((l & FaceMask.Right) == 0)
                {
Funkcija(Render, x + scale / 3, y, depth - 1, scale / 3, FaceMask.Top | FaceMask.Bottom);
}
else
                {
Funkcija(Render, x + scale / 3, y, depth - 1, scale / 3, FaceMask.Mask_All & ~FaceMask.Left);
}

if ((l & FaceMask.Left) == 0)
{
Funkcija(Render, x - scale / 3, y, depth - 1, scale / 3, FaceMask.Top | FaceMask.Bottom);
}
else
{
Funkcija(Render, x - scale / 3, y, depth - 1, scale / 3, FaceMask.Mask_All & ~FaceMask.Right);
}

if ((l & FaceMask.Top) == 0)
{
Funkcija(Render, x, y + scale / 3, depth - 1, scale / 3, FaceMask.Right | FaceMask.Left);
}
else
{
Funkcija(Render, x, y + scale / 3, depth - 1, scale / 3, FaceMask.Mask_All & ~FaceMask.Bottom);
}

if ((l & FaceMask.Bottom) == 0)
{
Funkcija(Render, x, y - scale / 3, depth - 1, scale / 3, FaceMask.Right | FaceMask.Left);
}
```

```
  else
  {
  Funkcija(Render, x, y - scale / 3, depth - 1, scale / 3, FaceMask.Mask_All & ~FaceMask.Top);
  }









  }

  Funkcija(Render, Convert.ToDouble(Size) / 2, Convert.ToDouble(Size) / 2, 2, Convert.ToDouble(Size),
FaceMask.Mask_All);
  Render.Write();


  }


  }


  internal class Renderer
  {
  private readonly uint[] Buffer;
  private readonly ushort Width;
  private readonly ushort Height;
  private readonly string OutputName;


  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";
  }

  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)
```

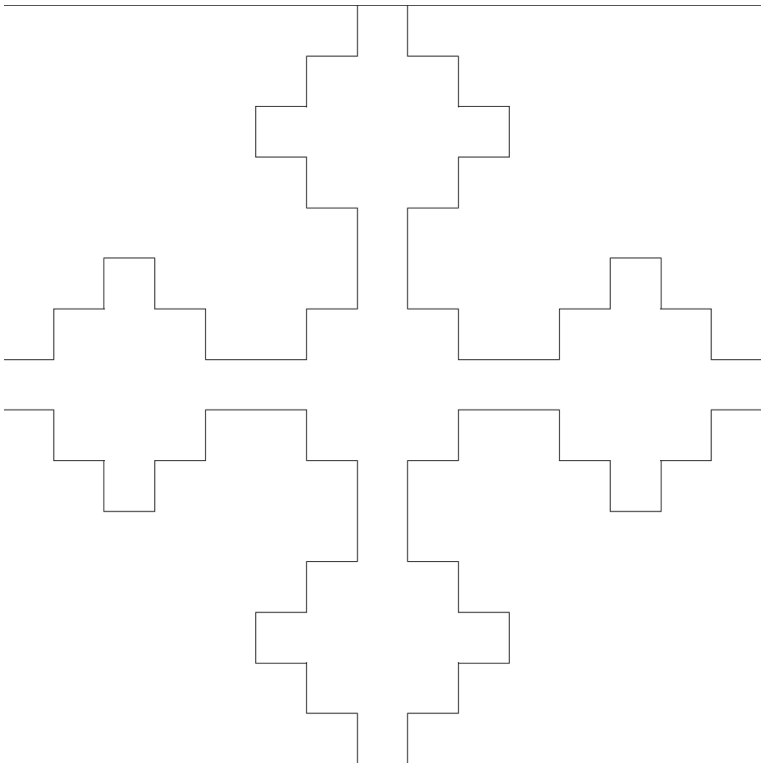
```
   return -1;
   else if (X > Width)
   return -1;

   return Pixel;
   }

   public void 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);
   }
   }


   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();
   }
   }

   }
}
```

Programos rezultatai

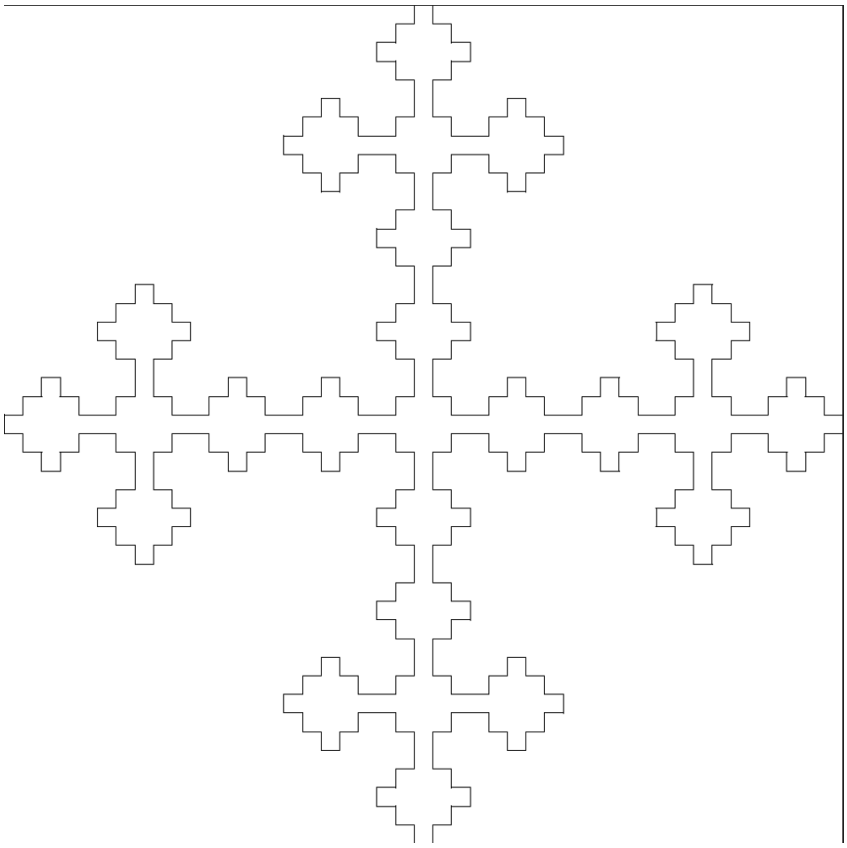
Iteracija Nr. 1:



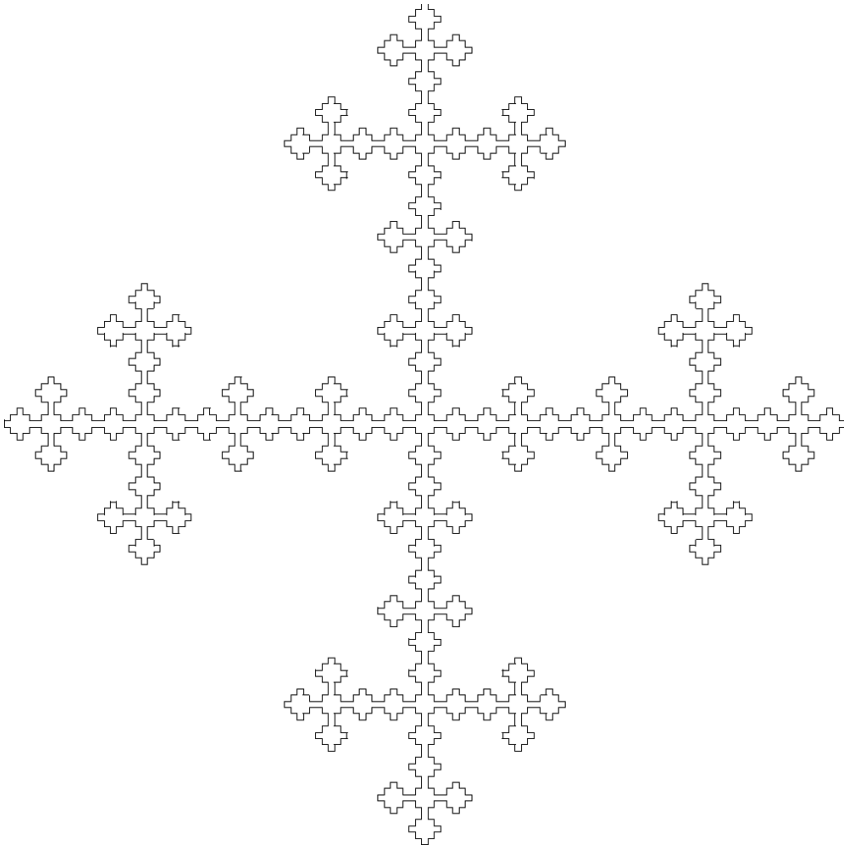
Iteracija Nr. 2:



Iteracija Nr. 3:



Iteracija Nr. 4:



Programos rezultatų greitaveika