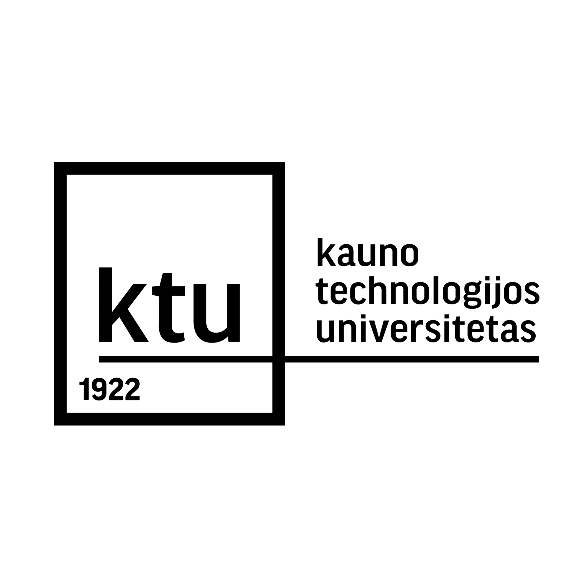
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

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

Laboratorinio darbo aprašas (ataskaita)

Antras laboratorinis darbas

**Dėstytojas**

lekt. MAKACKAS Dalius

**Studentas**

Matas Palujanskas IFF-1/8

KAUNAS 2023

TURINYS

[1. Užduotis Nr. 1 3](#_Toc130418005)

[2. Algoritmai ir analizė 5](#_Toc130418006)

[2.1. Kodas Nr. 1 5](#_Toc130418007)

[2.2. Kodas Nr. 2 8](#_Toc130418008)

[3. Kodas 10](#_Toc130418009)

[4. Užduotis Nr. 2 12](#_Toc130418010)

[5. Algoritmai 13](#_Toc130418011)

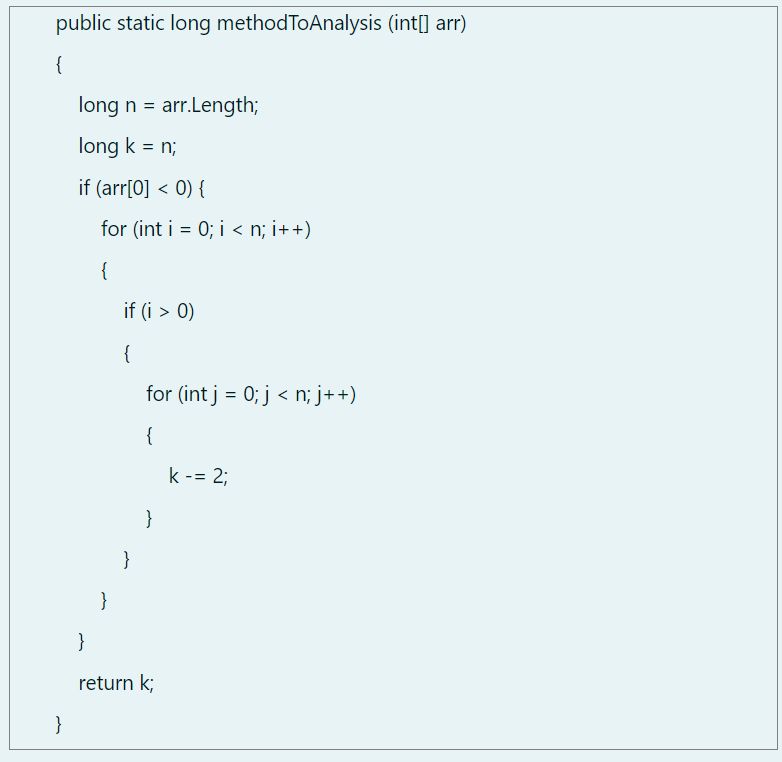
[6. Rezultatai 18](#_Toc130418012)

# Užduotis Nr. 1

**1 užduoties dalis:**Pateiktiems programinio kodo metodams „*methodToAnalysis(...)*“ (gautiems atlikus užduoties pasirinkimo testą):

* atlikite programinio kodo analizę, bei sudarykite rekurentinę lygį. Jei metodas neturi vidinių rekursinių kreipinių, apskaičiuokite pateikto metodo asimptotinį sudėtingumą. Jei metodo sudėtingumas priklauso nuo duomenų pateikiamų per parametrus – apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“ *(2 balai).*
* Metodams, kurie turi rekurentinių kreipinių išspręskite rekurentinę lygtį apskaičiuodami jos asimptotinį sudėtingumą*(1 balas).*
* Atlikti eksperimentinį tyrimą (našumo testus: vykdymo laiką ir veiksmų skaičių) ir patikrinkite ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus. Jei pateikto metodo asimptotinis sudėtingumas priklauso nuo duomenų, atitinkamai atliekant analizę reikia parinkti tokias testavimo duomenų imtis, kad rezultatai atspindėtų įvertinimus iš viršaus ir iš apačios*(1 balas).*

**Duoti kodai:**

****

**Paveikslėlis, kuriame yra tekstas

Automatiškai sugeneruotas aprašymas**

# Algoritmai ir analizė

## Kodas Nr. 1

|  |  |  |
| --- | --- | --- |
| **Kodas** | **Kaina** | **Kiekis** |
| public static long methodToAnalysis (int [] arr) |  |  |
| { |  |  |
| long n = arr.Length; | c1 | 1 |
| long k = n; | c2 | 1 |
| if (arr[0] < 0) { | c3 | 1 |
| for (int i = 0; i < n; i++) | c4 | n |
| { |  |  |
| if (i > 0) | c5 | n |
| { |  |  |
| for (int j = 0; j < n; j++) | c6 | n \* (n \* γ); |
| { |  |  |
| k -= 2; | c7 | n2 \* γ; |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| return k; | c8 | 1 |
| } |  |  |

jei arr[0] < 0, tai γ=-1, kitu atveju γ=0.

* Rezultatų atvaizdavimas grafikuose:

Kai γ=0:

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas

* Kai γ= -1:

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas

## Kodas Nr. 2

|  |  |  |
| --- | --- | --- |
| **Kodas** | **Kaina** | **Kiekis** |
| public static long methodToAnalysis2(int n) |  |  |
|  |  |  |
| { |  |  |
|  |  |  |
| long k = 0; | c1 | 1 |
| int[] arr = new int[n]; | c2 | 1 |
| Random randNum = new Random(); | c3 | 1 |
| for (int i = 0; i < n; i++) | c4 | n |
| { |  |  |
| arr[i] = randNum.Next(0, n); | c5 | n |
| k += arr[i] + FF1(i); | F(n) | n |
| } |  |  |
| return k; | c6 | 1 |
|  |  |  |
| } |  |  |
|  |  |  |
| public static long FF1(int n) |  |  |
| { |  |  |
| if (n > 0) |  | 1 |
| { |  |  |
| return FF1(n - 1); | F(n-1) | 1 |
| } |  |  |
| return n; |  | 1 |
| } |  |  |

Rekursinis algoritmas vykdys mažinimus vienetu tol kol pasieks 0, todėl sudėtingumas yra O(n).

# Kodas

using System.Diagnostics;

namespace Pirma

{

class Program

{

static void Main(string[] args)

{

int size = 2000;

int n = 10;

long k;

Console.WriteLine("Pirma rekurentine: ");

Console.WriteLine("1 - arr[0] = 0");

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

{

Stopwatch stopwatch = new Stopwatch();

int[] A = new int[size];

A[0] = 0;

stopwatch.Start();

k = methodToAnalysis1(A);

stopwatch.Stop();

Console.WriteLine($"Function working time: {stopwatch.Elapsed}");

Console.WriteLine($"Elements amount: {size}");

Console.WriteLine($"Counter: {k}");

Console.WriteLine();

GC.Collect();

size += 2000;

}

size = 20;

Console.WriteLine("Pirma rekurentine: ");

Console.WriteLine("1 - arr[0] = 1");

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

{

Stopwatch stopwatch = new Stopwatch();

int[] A = new int[size];

A[0] = -1;

stopwatch.Start();

k = methodToAnalysis1(A);

stopwatch.Stop();

Console.WriteLine($"Function working time: {stopwatch.Elapsed}");

Console.WriteLine($"Elements amount: {size}");

Console.WriteLine($"Counter: {k}");

Console.WriteLine();

GC.Collect();

size += 20;

}

n = 8;

size = 2000;

Console.WriteLine("Antra rekurentine:");

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

{

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

k = methodToAnalysis2(size);

stopwatch.Stop();

Console.WriteLine($"Function working time: {stopwatch.Elapsed}");

Console.WriteLine($"Elements amount: {size}");

Console.WriteLine($"Counter: {k}");

Console.WriteLine();

GC.Collect();

size += 2000;

} for (int i = 0; i < n; i++)

}

}

public static long methodToAnalysis1(int[] arr)

{

long n = arr.Length;

long k = n \* n;

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

{

if (arr[0] > 0)

{

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

{

k -= 2;

}

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

{

k += 3;

}

}

}

return k;

}

public static long methodToAnalysis2(int n)

{

long k = 0;

int[] arr = new int[n];

Random randNum = new Random();

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

{

arr[i] = randNum.Next(0, n);

k += arr[i] + FF1(i);

}

return k;

}

public static long FF1(int n)

{

if (n > 0)

{

return FF1(n - 1);

}

return n;

}

}

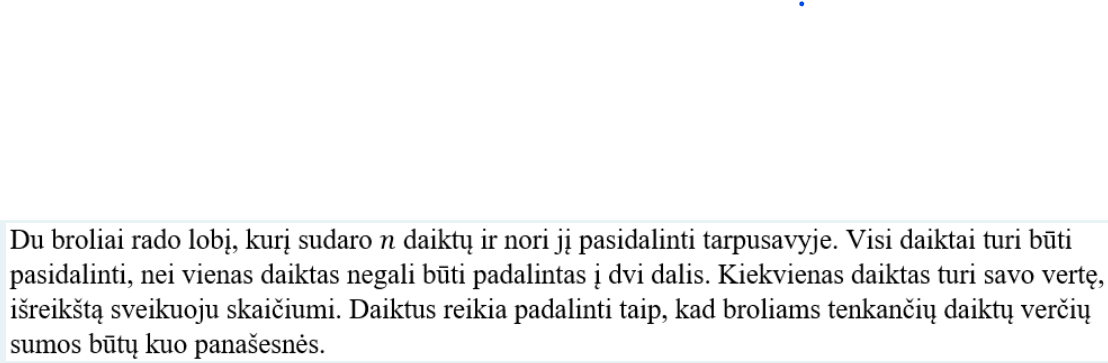
}

# Užduotis Nr. 2

**2 užduoties dalis (6 balai):**

* Pateikite rekursinį uždavinio sprendimo algoritmą (rekursinis sąryšis su paaiškinimais), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (rekursinis sprendimas netaikant dinaminio programavimo).
* Pritaikykite dinaminio programavimo metodologiją pateiktam uždaviniui (pateikti paaiškinimą), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (taikant dinaminį programavimą).
* Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.

**Užduotis:**

****

# Algoritmai

**Sprendimas naudojant rekursiją:**

|  |  |  |
| --- | --- | --- |
| **Kodas** | **Kaina** | **Kiekis** |
| static void CanDivide(int index, int sum1, int sum2) |  |  |
| { |  |  |
| if (index == values.Length) |  | 1 |
| { |  |  |
| int difference = Math.Abs(sum1 - sum2); |  | 1 |
| if (difference < minDifference) |  | 1 |
| { |  |  |
| minDifference = difference; |  | 1 |
| firstBrother.Clear(); |  | 1 |
| secondBrother.Clear(); |  | 1 |
| for (int i = 0; i < selected.Length; i++) |  | n |
| { |  |  |
| if (selected[i]) |  | n |
| { |  |  |
| firstBrother.Add(values[i]); |  | n |
| } |  |  |
| else |  |  |
| { |  |  |
| secondBrother.Add(values[i]); |  | n |
| } |  |  |
| } |  |  |
| } |  |  |
| return; |  | 1 |
| } |  |  |
| selected[index] = true; |  | 1 |
| CanDivide(index + 1, sum1 + values[index], sum2); | T(n) | 1 |
| selected[index] = false; |  | 1 |
| CanDivide(index + 1, sum1, sum2 + values[index]); | T(n) | 1 |
| } |  |  |

**T(n) = (c1+c2+c3+c4+c5+c6+(c7+c8+c9+c10)\*n+c11+c12+T(n)+c13+T(n)**

**Gaunamas sudėtingumas yra O(n).**

**Kodas naudojant rekursiją:**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

class TreasureDivisionRecursion

{

static int[] values;

static bool[] selected;

static List<int> firstBrother, secondBrother;

static int minDifference = int.MaxValue;

/// <summary>

/// Recursive solution

/// </summary>

/// <param name="index"></param>

/// <param name="sum1"></param>

/// <param name="sum2"></param>

static void CanDivide(int index, int sum1, int sum2)

{

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

if (index == values.Length)

{

int difference = Math.Abs(sum1 - sum2);

if (difference < minDifference)

{

minDifference = difference;

firstBrother.Clear();

secondBrother.Clear();

for (int i = 0; i < selected.Length; i++)

{

if (selected[i])

{

firstBrother.Add(values[i]);

}

else

{

secondBrother.Add(values[i]);

}

}

}

return;

}

// Try adding the item to the first brother's share

selected[index] = true;

CanDivide(index + 1, sum1 + values[index], sum2);

// Try adding the item to the second brother's share

selected[index] = false;

CanDivide(index + 1, sum1, sum2 + values[index]);

stopwatch.Stop();

Console.WriteLine($"Function working time: {stopwatch.Elapsed}");

Console.WriteLine();

}

static void Main()

{

values = new int[] { 2, 4, 5, 3 };

selected = new bool[values.Length];

firstBrother = new List<int>();

secondBrother = new List<int>();

CanDivide(0, 0, 0);

if (minDifference == 0)

{

Console.WriteLine("The treasure can be divided equally:");

}

else

{

Console.WriteLine("The treasure cannot be divided equally, but can be divided similarly:");

}

Console.WriteLine("First brother's share: " + string.Join(", ", firstBrother));

Console.WriteLine("Second brother's share: " + string.Join(", ", secondBrother));

}

}

**Sprendimas naudojant dinaminio programavimo metodologiją:**

|  |  |  |
| --- | --- | --- |
| **Kodas** | **Kaina** | **Kiekis** |
| public static bool CanDivide(int[] items) |  |  |
| { |  |  |
| int n = items.Length; | c1 | 1 |
| int S = 0; | c2 | 1 |
| for (int i = 0; i < n; i++) | c3 | n |
| { |  |  |
| S += items[i]; | c4 | n |
| } |  |  |
| if (S % 2 != 0) | c5 | 1 |
| { |  |  |
| return false; | c6 | 1 |
| } |  |  |
| bool[,] DP = new bool[n + 1, S / 2 + 1]; | c7 | 1 |
| for (int i = 0; i <= n; i++) | c8 | n |
| { |  |  |
| DP[i, 0] = true; |  | n |
| } |  |  |
| for (int i = 1; i <= n; i++) |  | n |
| { |  |  |
| for (int j = 1; j <= S / 2; j++) |  |  |
| { |  |  |
| if (items[i - 1] > j) |  |  |
| { |  |  |
| DP[i, j] = DP[i - 1, j]; |  |  |
| } |  |  |
| else |  |  |
| { |  |  |
| DP[i, j] = DP[i - 1, j] || DP[i - 1, j - items[i - 1]]; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| return DP[n, S / 2]; |  | 1 |
| } |  |  |

**T(n) = (c1+c2+(c3+c4)\*n+c5+c6+c7+(c8+c9+c10)\*n+(c11+c12+c13+c14)\*+c15**

**Gaunamas sudėtingumas yra O().**

**Kodas naudojant dinaminio programavimo metodologiją:**

using System;

public class TreasureDivision

{

public static bool CanDivide(int[] items)

{

int n = items.Length;

int S = 0;

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

{

S += items[i];

}

if (S % 2 != 0)

{

// The total sum is odd, so the items cannot be divided into two groups with equal sums.

return false;

}

bool[,] DP = new bool[n + 1, S / 2 + 1];

// Initialize the base case where the sum is zero.

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

{

DP[i, 0] = true;

}

// Compute the remaining cases.

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= S / 2; j++)

{

if (items[i - 1] > j)

{

DP[i, j] = DP[i - 1, j];

}

else

{

DP[i, j] = DP[i - 1, j] || DP[i - 1, j - items[i - 1]];

}

}

}

return DP[n, S / 2];

}

public static void Main(string[] args)

{

int[] items = new int[] { 2, 4, 5, 6, 7 };

if (CanDivide(items))

{

Console.WriteLine("The items can be divided into two groups with equal sums.");

}

else

{

Console.WriteLine("The items cannot be divided into two groups with equal sums.");

}

}

}

# Rezultatai

**Apskaičiuotas algoritmo darbo laikas, su skirtingu kiekiu duomenų ir gauti tokie rezultatai:**

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas

**Prilyginus grafą O(), matoma, jog sutampa kilimo tendencija, todėl galima spręsti, jog toks yra ir šio algoritmo sudėtingumas.**