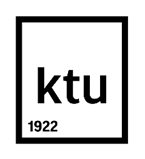
****

**Kauno technologijos universitetas**

Informatikos fakultetas, programų inžinerijos katedra

**P170B400 Algoritmų sudarymas ir analizė**

Antras laboratorinis darbas

|  |
| --- |
|  |
| **Arnas Švenčionis**  Projekto autorius  **IFF-8/11** |
| Akademinė grupė |

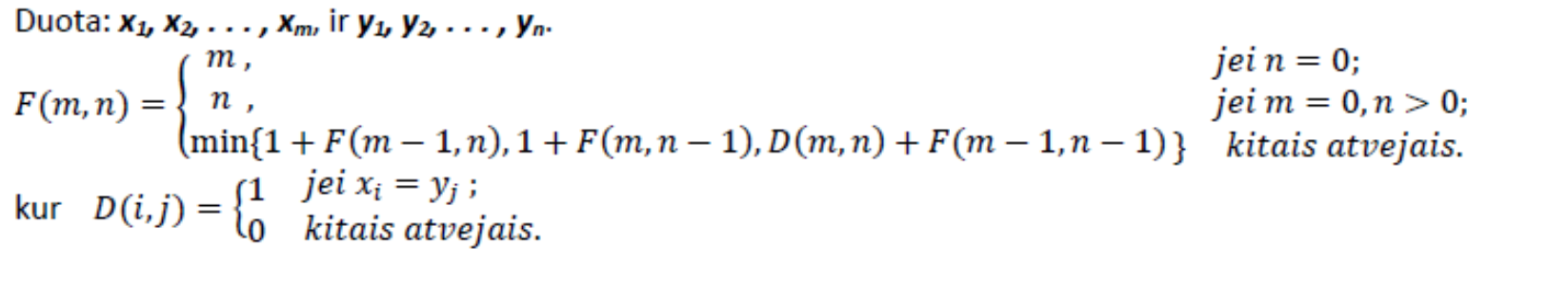
Contents

[1. Uždavinys 2](#_Toc41481658)

[2. Uždavinys 5](#_Toc41481659)

[3. Uždavinys 10](#_Toc41481660)

# Uždavinys



Duotai rekurentinei formulei sudarykite du algoritmus:

• Tiesiogiai panaudojant rekursiją;

• Panaudojant dinaminio programavimo metodo savybę, kad galime įsiminti dalinius sprendinius;

Programiškai realizuokite ir eksperimentiškai įvertinkite ir palyginkite abiejų algoritmų sudėtingumą.

**1.1 Programos pseudo kodo sudarymas**

a) Sudarome algoritmą, naudojant rekursiją:

private static int FD(int m, int n) kiekis kaina

{

if (n == 0) return m; c1 1

if (m == 0 && n > 0) return n; c1 1

int temp; c1 1

int min = 1 + FD(m - 1, n); F(m-1, n) 1

if ((temp = 1 + FD(m, n - 1)) < min) F(m, n-1) 1

min = temp; c1 1

if ((temp = D(m, n) + FD(m - 1, n - 1)) < min) 3c1+F(m-1,n-1) 1

min = temp; c1 1

return min; c1 1

}

private static int D(int i, int j) kiekis kaina

{

if (i >= x.Length || j >= y.Length) return 0; c1 1

if (x[i] == y[j]) return 1; c1 1

return 0; c1 1

}

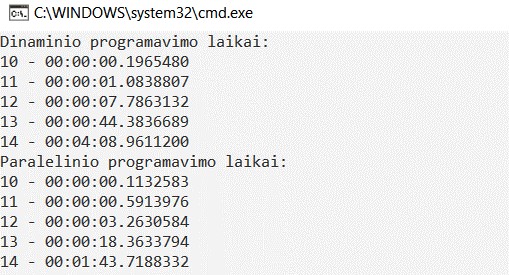
D(I,j) sudėtingumas – **3c1**

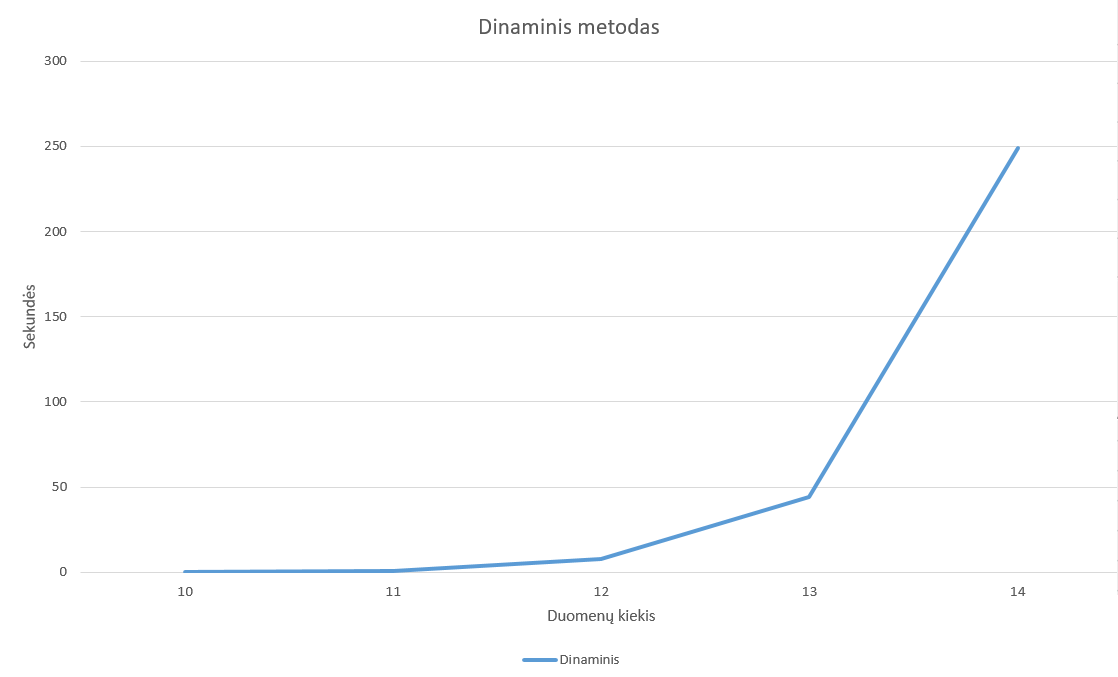
**Algoritmo sudėtingumas:**

**T(n) sudėtingumas bus didžiausias kai m = n = a. Taigi T(a) = F(a-1,a) + F(a,a-1) + F(a-1,a-1) + 9c = Gylis nuo pirmo rekursijos iškvietimo taško yra a, taiga = a((a-1)c + ac + ac) + a(ac + (a-1)c + ac) + a(ac+ac+(a-1)c)+c =**

**= a2c – ac + a2c + a2c + a2c + a2c –ac + a2c + a2c + a2c + a2c – ac + c = 9a2c – 3ac +c = O(a2)**

**1.2 Eksperimentinis algoritmų sudėtingumo įvertinimas**





**1.3 Dinaminio algoritmo sudarymas**

private static int FD2(int m, int n) kiekis kaina

{

int tm = m; c1 1

int tn = n; c1 1

int cm = 0; c1 1

int rez1 = 1 \* m + n; c1 1

int rez2 = 1 \* n + m; c1 1

int rez3 = 0; c1 1

int min = rez1; c1 1

while (n > 0 && m > 0) c2 n+1

{

tn--; c3 n

tm--; c3 n

if (tn == 0) c3 n

{

rez3 = m + cm; c3 n

break; c3 n

}

else if (tm == 0)

{

rez3 = n + cm;

break;

}

if (x[tm] == y[tn]) c3 n

cm++; c3 n

}

if (rez2 < rez1) c1 1

min = rez2; c1 1

if (rez3 < min) c1 1

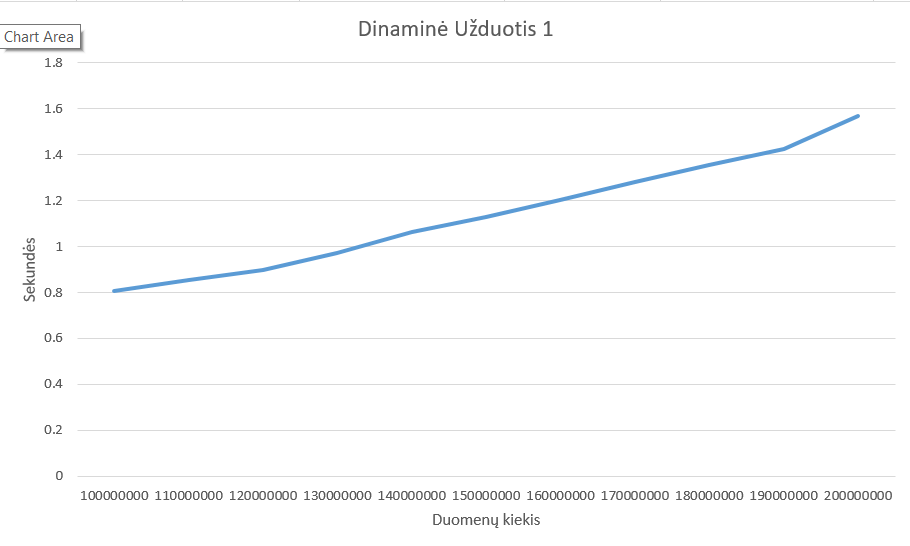
min = rez3; c1 1

return min; c1 1

}

**Sudėtingumo įvertinimas**

T(n) = 12c1+ c2(n+1) + 5c3n = **O(n);**

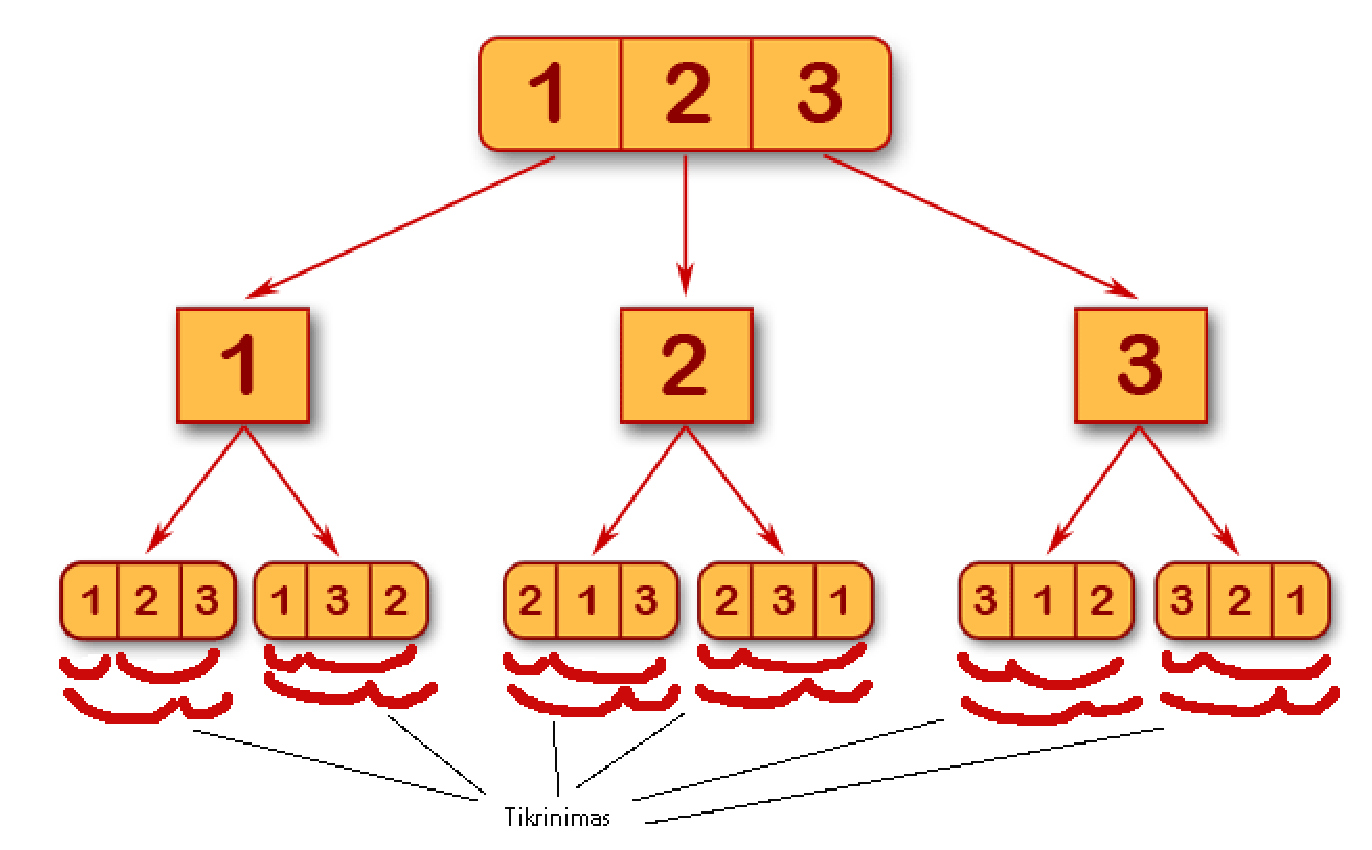


# Uždavinys

Pagal pateiktą žodinį uždavinio aprašymą, naudojantis dinaminio programavimo metodu, sudaryti uždavinio sprendimo algoritmą ir įvertinti jo sudėtingumą.

Dvi draugės – A ir B – nori pasidalyti ***n***dovanų rinkinį. Kiekviena dovana turi būti atiduota arba A, arba B, ir nė viena dovana negali būti padalyta į dvi dalis. Kiekviena dovana turi vertę, išreikštą sveikuoju skaičiumi nuo ***0*** iki ***m***. Pažymėkime ***Sa*** ir ***Sb*** dovanų, kurias atitinkamai gaus A ir B, verčių sumas. Reikia rasti, kaip padalyti dovanas A ir B, kad **|*Sa – Sb*|** būtų minimalus.

Kad patikrinti ar įmanoma išdalinti dovanas lygiai, programa turės perrykiuoti dovanų masyvą visais įmanomais variantais. Tai išskaldysime į atskirus variantus kai kiekvienas masyvo elementas yra pirmas:



Suradus kiekvieną dovanų masyvo išdėstymą, perduosime masyvams A ir B po 1..n elementų. Kiekvieną kartą tikrinsime elementų sumą ir ar dovanos yra padalintos geriau nei praeitą kartą. Taigi gausime:

Reorder(k,m) = 2m\*Swap(k, i) + m\*Reorder(k+1, m) + m!\*Split(1, m)

Kodo įgyvendijimas:

public static void reorder(int[] list, int k, int m) kiekis kaina

{

if (list.Length == 0) c1 1

return; c1 1

if (k == m) c1 1

split(list, 1); T(S) 1

else

{

for (int i = k; i <= m; i++) c2 n

{

swap(ref list[k], ref list[i]); 3c1 n-1

reorder(list, k + 1, m); reorder(n-1) n-1

swap(ref list[k], ref list[i]); 3c1 n-1

}

}

}

public static void swap(ref int a, ref int b) kiekis kaina

{

int temp = a; c1 1

a = b; c1 1

b = temp; c1 1

}

Metodo swap sudėtingumas = 3c1

public static void split(int[] list, int s)

{

if (s == list.Length) c1 1

return; c1 1

int Sa = 0; c1 1

int Sb = 0; c1 1

int[] Aa = new int[s]; c1 1

for (int i = 0; i < s; i++) c2 n/2

{

Sa += list[i]; c3 n/2-1

Aa[i] = list[i]; c3 n/2-1

}

int[] Ab = new int[list.Length - s]; c1 1

int c = 0; c1 1

for (int i = s; i < list.Length; i++) c2 n/2

{

Sb += list[i]; c3 n/2-1

Ab[c++] = list[i]; c3 n/2-1

}

if(Math.Abs(Sa - Sb) < min) c1 1

{

min = Math.Abs(Sa - Sb); c1 1

A = new int[Aa.Length]; c1 1

for (int i = 0; i < Aa.Length; i++) c2 n/2

A[i] = Aa[i]; c3 n/2-1

B = new int[Ab.Length]; c1 1

for (int i = 0; i < Ab.Length; i++) c2 n/2

B[i] = Ab[i]; c3 n/2-1

}

if (min == 0) c1 1

return; c1 1

split(list, ++s); split(n) 1

}

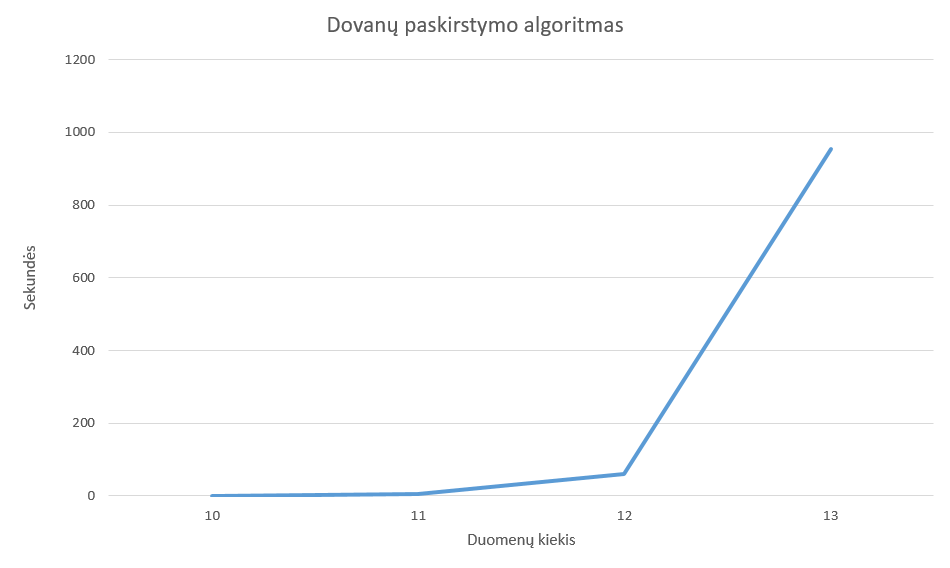
Metodo split sudėtingumas T(s)= 13c1 + 4c2(n/2) + 6c3(n/2-1) + split(n++) = 13c1+2c2n +3c3n+6c3+split(n++) =

= kadangi gylis yra n-1, tai = (n-1)(13c1+n(2c2+3c3))+6c3 = 13c1n+n2(2c2+3c3)-13c1-n(2c2+3c3)+6c3 = O(n2)

**Algoritmo sudėtingumas:**

Reorder(n) = 3c1 + c2n + 3c1(n-1) + (n-1)Reorder(n-1) = kadangi gylis yra n(n-1) =

= (n2-n)(3c1+c2n+3c1(n-1))=(n2-n)(n(c2+3c1))=n3(c2+3c1)-n2(c2+3c1) = **O(n3)**



**Dinaminis algoritmo įgyvendijimas**

private static int getMax(int[] list) kiekis kaina

{

int max = 0; c1 1

foreach (var item in list) c2 n

if (item > max && item > 0) c3 n-1

max = item; c3 n-1

return max; c1 1

}

Metodo getMax sudėtingumas T(m2)= 2c1 + 2c3(n-1) + c2n = 2c1 + 2c3n – 2c3 + c2n =

=n(2c3+c2) + 2c1 – 2c3 = O(n)

private static int getMin(int[] list) kiekis kaina

{

int min = getMax(list); T(m2) 1

int ind = 0; c1 1

for (int i = 0; i < list.Length; i++) c2 n

if (list[i] < min && list[i] > 0) c3 n-1

{

min = list[i]; c3 n-1

ind = i; c3 n-1

}

list[ind] \*= -1; c1 1

return min; c1 1

}

Metodo getMin sudėtingumas T(m)= 3c1 + c2n + 2c3(n-1) + T(m2) = n(c2 + 2c3) + 3c1 – 2c3 + n

= O(n)

private static void Dov2() kiekis kaina

{

int sumA = 0; c1 1

int sumB = 0; c1 1

int[] dov2 = dov; c1 1

int dovC = dov2.Length; c1 1

bool broken = false; c1 1

for (int i = 0; i < dov2.Length; i++) c2 n

{

if (sumA == sumB) c3 n-1

{

sumA += getMin(dov2); T(m) n

dovC--; c3 n-1

if (dovC == 0) c3 n-1

{

broken = true; c3 n-1

break; c3 n-1

}

}

if (broken) break; c3 n-1

while (sumA < sumB) c2 n

{

sumA += getMin(dov2); c3 n-1

dovC--; c3 n-1

if (dovC == 0) c3 n-1

{

broken = true; c3 n-1

break; c3 n-1

}

}

if (broken) break; c3 n-1

while (sumB < sumA) c2 n

{

sumB += getMin(dov2); c3 n-1

dovC--; c3 n-1

if (dovC == 0) c3 n-1

{

broken = true; c3 n-1

break; c3 n-1

}

}

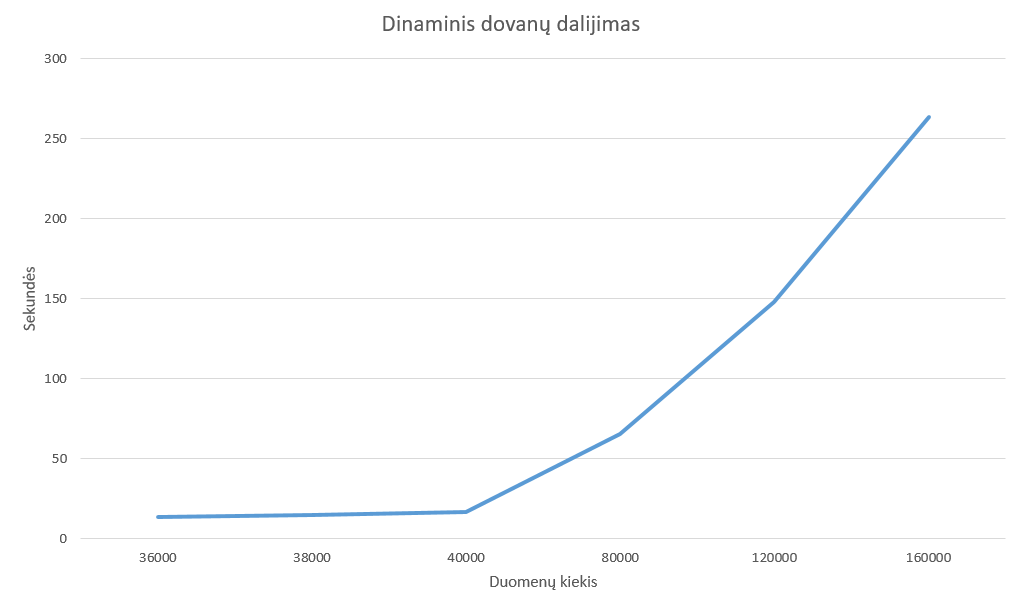
if (broken) break; c3 n-1

}

}

**Sudėtingumo įvertinimas:**

T(N) = 5c1 + c2n + 18c3(n-1) + n\*T(m) = n(18c3 + c2) +5c1 – 18c3 + n2 = **O(n2)**



# Uždavinys

Panaudojus 1 uždavinyje duotą rekurentinę formulę realizuoti jai algoritmą tiesiogiai panaudojant rekursiją bei lygiagretų programavimą.

private static int FP(int m, int n)

{

if (n == 0) return m;

if (m == 0 && n > 0) return n;

int countCPU = 3;

Task[] tasks = new Task[countCPU];

var task1 = Task.Factory.StartNew(() => 1 + FD(m - 1, n));

var task2 = Task.Factory.StartNew(() => 1 + FD(m, n - 1));

var task3 = Task.Factory.StartNew(() => D(m, n) + FD(m - 1, n - 1));

Task.WaitAll(task1, task2, task3);

int min = task1.Result;

if (task2.Result < min)

min = task2.Result;

if (task3.Result < min)

min = task3.Result;

return min;

}

**Eksperimentinis palyginimas:**

