

2016/2017

**Programming, Algorithms and Data Structures (210CT)**

Coursework

**MODULE LEADER:** Dr. Diana Hintea

**Student Name:**

**SID:**

**I can confirm that all work submitted is my own: Yes**

This coursework is testing your ability to design algorithms, write pseudocode, describe their efficiency, implement various data structures and use the programming language of your choice for the implementation.

Basic:

**1.**

The algorithm below randomly shuffles an array of integers by permetutation of each elements with a random element of the array.

static int [] shuffle(int[] tab)

{

Random rand = new Random(); (1 times)

for (int i = 0; i < tab.Length; i++) (n times)

{

int r = rand.Next(0, tab.Length); (n times)

int tmp = tab[i]; (n times)

tab[i] = tab[r]; (n times)

tab[r] = tmp; (n times)

}

return tab; (1 times)

}

**Display:**

static void Main(string[] args)

{

int[] tab = {1,2,3,4,5,6,7,8,9};

shuffle(tab);

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

{

Console.Write(tab[i] + "|");

}

Console.ReadLine();

}

**2.**

static int Factorielle(int n) (n times)

{

if (n == 0)

return 1;

else

return n \* Factorielle(n - 1);

}

static int zeros(int n)

{

int primerFactorOfFive = 0; (1 time)

while (n % 5 == 0) (n times)

{

n = n / 5; (n times)

primerFactorOfFive++; (n times)

}

return primerFactorOfFive; (1 time)

}

static void Main(string[] args)

{

Console.WriteLine(zeros(Factorielle(10)));

Console.ReadLine();

}

**3.**

**Pseudocode**:

PERFECT\_SQUARE(n)

for i <- n Downto 0

if(sqrt(i) % 1 = 0)

return i

return -1

**Implementation**:

static int perfectSquare(int n)

{

for (int i = n; i > 0; i--)

{

if (Math.Sqrt(i) % 1 == 0)

return i;

}

return -1;

}

static void Main(string[] args)

{

Console.WriteLine(perfectSquare(30));

Console.ReadLine();

}

**4.**

**Week1 :**

Basic Task 1: 5n +2 removing constans and multipliers gives O(n).

Basic Task2: 4n +2 removing constans and multipliers gives O(n).

**5. Pseudocode** :

matAddition(mat1,mat2)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

mat3[i,j] <- mat1[i,j] + mat2[i,j]

return mat3

matSubtraction(mat1,mat2)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

mat3[i,j] <- mat1[i,j] - mat2[i,j]

return mat3

matMultiplication(mat1,mat2)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

for k <- 0 To length

mat3[i,j] <- mat3[i,j] + (mat1[i,k] \* mat2[k,j])

return mat3

matTimesInt(a,mat1)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

mat3[i,j] <- a \* mat1[i,j]

return mat3

**Implementation** :

static int [,] matAddition(int[,] mat1, int[,] mat2) (2n² + n + 3 times)

{

int length = mat1.GetLength(0); (1 time)

//Because mat1 and mat2 are two quadratic matrices of the same order //so(mat1.GetLength(0)=mat1.GetLength(1)=mat2.GetLength(0)=mat2.GetLength(1))

int[,] mat3 = new int[length, length]; (1 time)

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

{

for (int j = 0; j < length; j++) (n² times)

{

mat3[i, j] = mat1[i, j] + mat2[i, j]; (n² times)

}

}

return mat3; (1 time)

}

static int [,] matSubtraction(int[,] mat1, int[,] mat2){ (2n² + n + 3 times)

int length = mat1.GetLength(0); (1 time)

int[,] mat3 = new int[length, length]; (1 time)

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

{

for (int j = 0; j < length; j++) (n² times)

{

mat3[i, j] = mat1[i, j] - mat2[i, j]; (n² times)

}

}

return mat3; (1 time)

}

static int [,] matMultiplication(int[,] mat1, int[,] mat2){(2n^3+n²+n+3 times)

int length = mat1.GetLength(0); (1 time)

int[,] mat3 = new int[length, length]; (1 time)

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

{

for (int j = 0; j < length; j++) (n² times)

{

for (int k = 0; k < length; k++) (n^3 times)

{

mat3[i, j] = mat3[i, j] + (mat1[i, k] \* mat2[k, j]);(n^3 times)

}

}

}

return mat3; (1 time)

}

static int[,] matTimesInt(int a, int[,] mat1) (2n² + n + 3 times)

{

int length = mat1.GetLength(0); (1 time)

int[,] mat3 = new int[length, length]; (1 time)

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

{

for (int j = 0; j < length; j++) (n² times)

{

mat3[i, j] = a \* mat1[i, j]; (n² times)

}

}

return mat3; (1 time)

}

static void Main(string[] args)

{

int[,] mat1 = { { 1, 2 }, { 3, 4 } }; //B (1 time)

int[,] mat2 = { { 4, 5 }, { 6, 7 } }; //C (1 time)

int[,] matAdd = matAddition(mat1, mat2); //(B+C) (1 + 2n² + n + 3 times)

int[,] matMul = matMultiplication(mat1, mat2);//(B\*C)(1+2n^3+n²+n+3 times)

matAdd = matTimesInt(2, matAdd); //2\*(B+C) (1 + 2n² + n + 3 times)

int[,] finalMat = matSubtraction(matMul, matAdd);//(B\*C)-2\*(B+C)(1 + 2n²+n+3 times)

Console.ReadLine();

}

**So**:

(1) + (1) + (1 +2n²+n+3) +( 1 +2n^3 + n²+n+3) +(1+2n²+n+3) + (1+2n²+n+3) =

(2n^3+7n²+4n+14) removing constans and multipliers gives **O(n^3)**.

**6.**

**Pseudocode:**

reverseString(sentence)

tabReversed <- [sentence.Length]

reversed <- ""

j <- 0

for(i <-0 To sentence.Length)

if(sentence[i]!=' ')

tabReversed[j] <- tabReversed[j] + sentence[i]

else j <- j + 1

for(i <- j Downto 0)

reversed <- reversed + tabReversed[i] + " "

return reversed

**Code:**

static string reverseString(string sentence)

{

string[] tabReversed = new string[sentence.Length]; (1 time)

string reversed = ""; (1 time)

int j = 0; (1 time)

for (int i = 0; i < sentence.Length; i++) (n times)

{

if (sentence[i] != ' ') (n times)

tabReversed[j] += sentence[i]; (n times)

else j++; (n times)

}

for (int i = j; i >=0; i--) (n times)

{

reversed+=tabReversed[i] + " "; (n times)

}

return reversed; (1 time)

}

**Display (in the main)** : Console.WriteLine(reverseString("This is awesome"));

**Run time:** 4 + 6n removing constans and multipliers gives O(n).

**7.**

static bool prime(int n, int i)

{

if (n == i) //Base case (n times)

return true; (1 time)

else if (n % i == 0) (n times)

return false; (1 time)

return prime(n, i+1); (n times)

}

**Display (in the main) :**

Console.WriteLine(prime(11,2));

**Run time:** 2 + 3n removing constans and multipliers gives O(n).

**8.**

static string removeVocals(string s,int i)

{

if (i >= s.Length) //Base case

return s;

if (s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u' || s[i] == 'y')

{

s = s.Remove(i, 1); //Remove a letter (index, length)

return removeVocals(s,i);

}

else

{

i++;

return removeVocals(s,i);

}

}

**Display (in the main) :**

Console.WriteLine(removeVocals(“beautiful”,0);