

2016/2017

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

Coursework

**MODULE LEADER:** Dr. Diana Hintea

**Student Name: Alain Trantik**

**SID: 7455647**

**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. Shuffle**

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

**Code:**

static int [] shuffle(int[] array)

{

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

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

{

int r = rand.Next(0, array.Length); (n times)   
//random number between [0,tab.Length[

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

array[i] = array[r]; (n times) //permutation

array[r] = array; (n times)

}

return array; (1 times)

}

**Display:** static void Main(string[] args)

{

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

shuffle(array);

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

{

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

}

Console.ReadLine();

}

**2. Number of zeros**

**Code:**

static int Factorielle(int n) (n times)

{

if (n == 0)

return 1;

else

return n \* Factorielle(n - 1);

}

If a number is divisible n times by 5, it has n trailing 0s.

static int zeros(int n)

{

int primerFactorOfFive = 0;

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

{

n = n / 5; (n times)

primerFactorOfFive++; (n times)

}

return primerFactorOfFive;

}

**Display**:

static void Main(string[] args)

{

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

**3. Perfect square**

**Pseudocode**:

PERFECT\_SQUARE(n)

for i <- n Downto 0

if(sqrt(i) % 1) = 0   
//if sqrt(i)%1=0 => sqrt(i) is a whole number => i is a perfect square

return i

return -1   
//error case: if there is not perfect square less or equal to n

**Code**:

static int perfectSquare(int n)

{

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

{

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

return i;

}

return -1;

}

**Display**

static void Main(string[] args)

{

Console.WriteLine(perfectSquare(30));

Console.ReadLine();

}

**4: Run time with Big O notation (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: Matrix operations**

**Pseudocode** :

//we add each elements which are in the same position

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

//we substract each elements which are in the same position

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

**DESCRIPTION**

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

//we multiply each element with a (integer)

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

**code** :

static int [,] matAddition(int[,] mat1, int[,] mat2) O(n²)

{

int length = mat1.GetLength(0);

//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];

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;

}

static int [,] matSubtraction(int[,] mat1, int[,] mat2){ O(n²)

int length = mat1.GetLength(0);

int[,] mat3 = new int[length, length];

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;

}

static int [,] matMultiplication(int[,] mat1, int[,] mat2) O(n^3)

int length = mat1.GetLength(0);

int[,] mat3 = new int[length, length];

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;

}

static int[,] matTimesInt(int a, int[,] mat1) O(n²)

{

int length = mat1.GetLength(0);

int[,] mat3 = new int[length, length];

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;

}

**Main:**

static void Main(string[] args)

{

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

int[,] mat2 = { { 4, 5 }, { 6, 7 } }; //C

int[,] matAdd = matAddition(mat1, mat2); //(B+C)

int[,] matMul = matMultiplication(mat1, mat2);//(B\*C)

matAdd = matTimesInt(2, matAdd); //2\*(B+C)

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

Console.ReadLine();

}

Finally we just have to display finalMat.

**So** **the run time of the algorithm above is O(n^3)**.

**6: Reverse a string**

I create an array of strings, then I write each letter of the sentence in the first cell of the array until I found a space. When I found a space, I jump to the next cell and I continue to write into the array until the end of the sentence. Finally each cell of my array is a word of my sentence so I just have to display it upside down (from the end to the beginning).

**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];

string reversed = "";

int j = 0;

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

{

if (sentence[i] != ' ')

tabReversed[j] += sentence[i];

else j++;

}

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

{

reversed+=tabReversed[i] + " ";

}

return reversed;

}

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

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

**7: Prime number ?**

**Pseudocode:**

n is prime if it’s only divisible by 2 and itself

prime(n,i) //i starts at 2

if(n = i) //base case: all numbers between 1 and n don’t divide n

return true

if(n % i = 0) //to be prime n must not be divisible by 2,…,n-1

return false

return prime(n, i+1)

**Code:**

static bool prime(int n, int i) //i starts at 2

{

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

return true;

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

return false;

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

}

**Display (in the main) :**

Console.WriteLine(prime(11,2)); =>true

**Run time:**

2 + 3n removing constans and multipliers gives O(n).

**8: Remove vocals from a sentence/word**

**Pseudocode:**

I check the word letter by letter, if it’s a vocal i remove it, if it’s not I check the next letter

removeVocals(s,i)

if(i>s.Length) //base case (all the letters have been checked)

return s

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

remove letter i from s

return removeVocals(s,i)

else

return removeVocals(s,i+1)

**Code:**

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

{

return removeVocals(s,i+1);

}

}

**Display (in the main) :**

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

**9: Binary search with an interval**

**Pseudocode:**

**Code:**

static bool binarySearch(int[] A, int a ,int b, int first, int last)

{

if (last < first)

return false;

int i = first + (last - first) / 2;

if (A[i] >= a && A[i]<=b)

return true;

if (A[i] < a)

return binarySearch(A, a, b, i + 1, last);

else

return binarySearch(A, a, b, first, i - 1);

}

**Display (in the main) :**

static void Main(string[] args)

{

int[] array = { 1, 2, 3, 8, 10, 15, 21 };

Console.WriteLine(binarySearch(array, 4, 8, 0, array.Length - 1));

Console.ReadLine();

}

**Run time :  
O(log n)**

**9: Longer sub-sequence of ascending numbers**

**Code :**

static int[] max\_sub\_sequence(int[] sequence)

{

int curr\_length = 0; //current length of the sub-sequence

int max\_length = 0; //maximum length of the sub-sequence

int end = 0; //end index of the longer sub-sequence

int curr\_end = 0; //end index of the current sub-sequence

for (int i = 0; i < sequence.Length - 1; i++)

{

if (sequence[i] < sequence[i + 1])   
//Couting the length of the current sub-sequence of ascending numbers and saving the index of the last number

{

temp\_end = i + 1;

length++;

}

if (!(sequence[i] < sequence[i + 1]) || i == sequence.Length - 2)   
//I put (if i=sequence.Length-2) in order to save the variables if the last number of the sub-sequence is also the last number of the sequence.

{

if (length > max\_length)   
//when the sub-sequence is broken (sequence[i]>sequence[i+1]), I save its length and the index of its last number if the length is higher than the maximum length stored.

{

max\_length = length;

end = temp\_end;

}

length = 0;   
//I reinitialize the length at 0 so it cans count the length of the second sub-sequence...

}

}

int[] sub\_sequence = new int[max\_length + 1];   
//I create an array to store the longer sub-sequence found

for (int i = 0; i <sub\_sequence.Length; i++)

{

sub\_sequence[i] = sequence[end - max\_length + i];   
//the first index of the sub-sequence is lastIndex - length.

}

for (int i = 0; i < sub\_sequence.Length;i++ )

{

Console.Write(sub\_sequence[i] + "|");   
//I display the sub-sequence before return it back.

}

return sub\_sequence;

}

**Main :**

static void Main(string[] args)

{

int[] sequence = {1,2,3,1,2,3,4,5,1};

max\_sub\_sequence(sequence);

Console.ReadLine();

}