Tom Coursework

static void ex1(int [] tab)

{

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

{

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

}

Console.WriteLine();

Random shuffle = new Random();

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

{

int nextindex = shuffle.Next(tab.Length);

int tmp = tab[i];

tab[i] = tab[nextindex];

tab[nextindex] = tmp;

}

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

{

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

}

}

The first table which is dispayed is the original one. The second has the same numbers but in different order.



static int ex2(int number)

{

Console.WriteLine(Fact(number));

int count = 0;

int a = Fact(number);

while (a % 5 == 0)

{

a = a / 5;

count++;

}

return count;

}



Pseudo-code :

Funct(n)

square 🡸 1

For (i 🡸 0 to n)

if(√i % 1 = 0)

square 🡸 1

return square

Implementation in C# :

static int PerfectSquare(int n)

{

int square = 1;

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

{

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

{

square = i;

}

}

return square;

}



static void ex1(int [] tab)

{

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

{

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

}

Console.WriteLine(); 1

Random shuffle = new Random(); 1

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

{

int nextindex = shuffle.Next(tab.Length); n

int tmp = tab[i]; n

tab[i] = tab[nextindex]; n

tab[nextindex] = tmp; n

}

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

{

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

}

}

static int ex2(int number)

{

Console.WriteLine(Fact(number)); 1

int count = 0; 1

int a = Fact(number); 1

while (a % 5 == 0) n

{

a = a / 5; n

count++; n

}

return count; 1

}



Addition :

Function Add(matrixA, matrixB)

dimension 🡸 length[matrixA] 1

matrixC 🡸 new matrix[dimension, dimension] 1

for (i in dimension) n

for(j in dimension) n\*n

matrixC[i,j] 🡸 matrixA[i,j] + matrixB[i,j] n\*n

return matrixC 1

Substraction :

Function Sub(matrixA, matrixB)

dimension 🡸 length[matrixA] 1

matrixC 🡸 new matrix[dimension,dimension] 1

for (i 🡸 0 to dimension) n

for(j 🡸 0 to dimension) n\*n

matrixC[i,j] 🡸 matrixA[i,j] - matrixB[i,j] n\*n

return matrixC 1

Multiplication :

Function Multiply(matrixA, matrixB)

dimension 🡸 length[matrixA] 1

matrixC 🡸 new matrix[dimension,dimension] 1

for (i in dimension) n

for(j in dimension) n\*n

k🡸0 n\*n

while(k<dimension) n^3

c[i,j] 🡸 c[i,j] + (a[i,k] \* b[k,j]) n^3

k++ n^3

return matrixC 1

Operation :

Function Operation(matrixA, matrixB)

dimension 🡸 length[matrixA] 1

matrixD 🡸 new matrix 1

matrixC 🡸 new matrix[dimension,dimension] 1

for (i in dimension) n

for(j in dimension) n\*n

matrixD[i,j] 🡸 2 \* matrixD[i,j] n\*n

matrixC 🡸 Sub(Multiply(matrixA, matrixB),matrixD) 1

return matrixC 1

Run Time : O(n^2) ;



Code :

static string[] Reverse(string sentence)

{

string newword = "";

int k = 0;

string[] words = new string[sentence.Length];

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

{

if (sentence[i] != ' ')

{

newword = newword + sentence[i];

}

else

{

words[k] = newword + " ";

newword = "";

k++;

}

if (i == sentence.Length - 1)

{

words[k] = newword + " ";

}

}

string[] finalsentence = new string[words.Length + 1];

for (int i = 1; i < words.Length; i++)

{

finalsentence[i] = words[words.Length - i];

}

finalsentence[finalsentence.Length - 1] = words[0];

return finalsentence;

}

Pseudo-code :

Function(sentence)

Neword 🡸 « »

k 🡸 0

words 🡸 string[length[sentence]]

for (i 🡸 0 in legth[sentence])

if senntence[i] = « »

newword 🡸 newword + sentence[i]

else

word[k] 🡸 newword + « »

newword 🡸 « »

k++

if i = length[sentence] -1

word[k] 🡸newword + « »

finalesentence 🡸 string[length[word]+1]

for (i 🡸 1 in words)

finqlesentence[i] 🡸 words[length[words]-i]

finalesentence[length[finalsentence] - 1]🡸 words[0] ;

return finalsentence



static bool PrimaryRecusrsive(int n, int i)

{

if (i == n)

{

return true;

}

else if(n % i == 0)

{

return false;

}

return PrimaryRecusrsive(n, i + 1);

}

Pseudo code :

Function (n,i)

{

If (i = n)

{

return true ;

}

Else

return Function(n, i+1) ;

}

}



static string RemoveVowel(string word, int i)

{

if (i < word.Length)

{

if (word[i] == 'a' || word[i] == 'e' || word[i] == 'i' || word[i] == 'o'

|| word[i] == 'u' || word[i] == 'y')

{

return RemoveVowel(word.Remove(i,1), i);

}

else

{

return RemoveVowel(word, i + 1);

}

}

else

{

return word;

}

}

Pseudo code :

Function(word, i)

{

If(i<Length[word])

{

If(word[i] = ‘a’ || word[i] = ‘e’ || word[i] = ‘i’ || word[i] = ‘o’ || word[i] = ‘u’ || word[i] = ‘y’)

{

Return function(word - word[i], i) ;

}

Else

{

Return function(word, i+1) ;

}

}

Else

{

Return word ;

}

}



Code :

static bool BinarySearch(int a, int b)

{

int[] tab = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int first = 0;

int last = tab.Length - 1;

if(a>tab[tab.Length -1] && b > tab[tab.Length - 1])

{

return false;

}

while (last > first)

{

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

if (tab[mid] == a || tab[mid] == b || tab[mid] > a || tab[mid] < b)

{

return true;

}

else if (tab[mid] < a)

{

first = mid + 1;

}

else if (tab[mid] > b)

{

last = mid - 1;

}

}

return false;

}

Pseudo-Code :

Function(a,b)

Tab 🡸 { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } 1

First 🡸 0 1

Last 🡸 length[tab] -1 1

if a > length[tab] -1 && b > length[tab] -1 1

return false 1

while last<first n

mid 🡸 first + (last – first)/2 n

if tab[mid] = a || tab[mid] = b || tab[mid]>a || tab[mid] < bn

return true n

else if tab[mid] < a n

first = mid +1 n

else if tab [mid] > b n

last = mid – 1 n

reutrn false 1

Run Time : O(n) ;



Code :

static int [] AscendingNumbers(int [] tab)

{

int longestlength = 0;

int start = 0;

int startlongest = 0;

int length = 0;

for (int i = 1; i<tab.Length; i++)

{

length = i - start;

if (tab[i] < tab[i - 1])

{

if (longestlength < length)

{

longestlength = length;

startlongest = start;

}

start = i;

}

if (longestlength < length)

{

longestlength = length;

startlongest = start;

}

}

int[] result = new int[longestlength];

for (int i = startlongest; i<(startlongest+longestlength); i++)

{

result[i-startlongest] = tab[i];

}

Console.WriteLine("longestlength : " + longestlength + "\nstartlongest : " + startlongest);

return result;

}