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) ;



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 ;

}

}



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) ;

}

}