Basic Programming Tasks

You are free to use any programming language.

Programming Task 1

How would you write a function to reverse a string? And can you do that without a temporary string?

Example solution below.

String str = "The quick brown fox jumps over the lazy dog";

```
   int len = str.length();
   for (int i = (len-1); i >= 0; --i)
      str += str.charAt(i);
   str = str.substring(len);
   System.out.println(str);
```

Your task is to implement a more efficient solution (or at least a different solution).

```
public static void main(String[] args) {

   String test = "Hello World";
   String rev = "";
   Pattern p = Pattern.compile("[\\w|\\W]");
   Matcher m = p.matcher(test);
   while (m.find()) {
      rev = m.group()+rev;
   }
   System.out.println("Reverse==" + rev);
}
```

Programming Task 2

In an array with integers between 1 and 1,000,000 one value is in the array twice. How do you determine which one?

Examples below:

```
public static int FindDuplicateInArrary(int[] arr)

{

    ulong n = arr.Length, dsn, dsn2;

    dsn = n*(n + 1)/2;

    dsn2 = dsn*(2*n + 1)/3;

    for(ulong i = 0; i < n; ++i){

        dsn -= arr [i];

        dsn2 -= arr [i]* arr [i];

    }

    return (dsn2 - dsn*dsn)/(2*dsn);

}
```

```
#include <stdio.h>

int main(int argc, char **argv) {

    int array[] = { 10, 2, 3, 5, 5, 7, 8, 8, 9, 1 };
    int n = 10;
    int i, j, k;

    for (i = 1; i <= n; ) {

        j = array[i - 1];

        if (j != 0) {

            k = array[j - 1];

            if (k != 0) {

                array[j - 1] = 0;
            }
            else printf("%d\n", j);
            if (j != i) {

                array[i - 1] = k;
            }
            else i++;
        }
        else i++;
    }
}
```

Create an array with at least 1 x 10^{10} integers. Try to implement your own solution to solve the problem.

Programming Task 3

Write a program to read words from the attached file. Count the repeated or duplicated words. Sort it by maximum repeated or duplicated word count.

Programming Task 4

Extend the first task with the following feature:

Write a program to find all distinct words from the attached file. Remove special chars like ".,;:" etc. Ignore case sensitivity.

Programming Task 5

Extend task 2 with a menu where the user can choose between running task 1 and task 2. Consider wrap the two different tasks in two different classes.

Programming Task 6

Given a string with a length greater than 5 characters, find the longest substrings without repeating characters. Iterate through the given string, find thelongest maximum substrings. A substring must at least be three characters.

Programming Task 7

Create a Class called ListUtil which contain a public method called MergeSortedLists() which merge two sorted lists (You will have to create two different internal sub methods to solve problem a) and b).

Create a separate project called TestMergeOfSortedList. When the two different lists are merged the final list must be correctly sorted.

a) Two lists with int numbers

List1 {12, 14, 45, 67, 98, 123, 134, 345, 456, 569, 748, 1123, 5757 , 11111, 3636363}

List2 {9, 15, 17, 34, 44, 49, 97, 100, 105, 122, 133, 423, 465, 2929, 48484, 3637632}

b) Two lists with strings (names)

List1 [Adam, Anders, Bertil, Boris, Carl, David, Dennis, Peter, Rasbert, Ronny, Sonny, Sören}

List2 {Anja, Anna, Berit, Bodil, Carina, Cecilia, Doris, Penny, Rosalyn, Signe, Svetlana, Tuva, Sanna}

c) Think of how your code would save the problems as in a) and b) if the lists contained more than 1 000 000 000 elements

Programming Task 8

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlocks can occur in C# when the lock statement causes the executing thread to block while waiting to get the lock, associated with the specified object. Since the thread might already hold locks associated with other objects, two threads could each be waiting for the other to release a lock. In such case, they will end up waiting forever. Write a simple program to demonstrate deadlock between two threads in C# or Java.

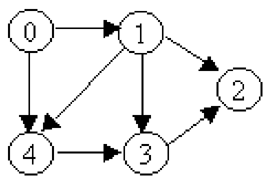
Programming Task 9

Adjacency Matrix

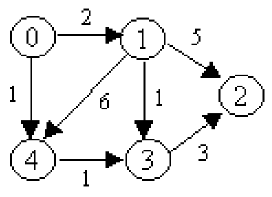
Adjacency matrix is a V x V matrix in which entry A[i][j] = 1 if there exists a path from vertex i to vertex j—else it is 0. We can observe easily that if graph is Undirected then A[i][j] = A[j][i], while the same is not necessary in the case of a directed graph.

Adjacency matrix can be easily modified to store cost of path A[i][j] = C(i,j) in case of weighted graph.

Time complexity of accessing information from Adjacency Matrix is O(1) while Space complexity is O(N^2)
Example of above can be seen in the image below



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$



$$A = \begin{pmatrix} \infty & 2 & \infty & \infty & 1 \\ \infty & \infty & 5 & 1 & 6 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 1 & \infty \end{pmatrix}$$

Write a simple program to demonstrate the use of Adjacency matrix. I have written an example below to demonstrate on what level we expect a solution.

```
int main()
{
  int nodes, edges;
  cin >> nodes;      // input number of nodes in graph
  cin >> edges;      // input number of edges in graph

  int graph[nodes+1][nodes+1];
  for( int i =0 ; i <= nodes ; i++)
    for( int j = 0 ; j <= nodes ; j++)
      graph[i][j] = 0; // initilise all nodes as disconnected from all nodes.
  for( int i = 0 ; i < edges ; i++)
  {
    int u , v ;
    cin >> u >> v;
    graph[u][v] = 1;  // graph[u][v] = c if c is weight of edge
    graph[v][u] = 1;  // if graph is Directed this line will be omitted.
  }
  for( int i = 1 ; i <= nodes ; i++ )
  {
    cout << " Node " << i << " is connected to : " ;
    for( int j = 1 ; j <= nodes ; j++)
    {
      if( graph[i][j] )
        cout << j << " ";
    }
    cout << endl;
  }
  return 0;

}
```

Programming Task 10

Insertion sort

Example

Array	Shifts
[4,3,2,1]	
[3,4,2,1]	1
[2,3,4,1]	2
[1,2,3,4]	3

Total shifts = 1 + 2 + 3 = 6

Insertion Sort is a simple sorting technique. Sometimes arrays may be too large for you to wait around for insertion sort to finish, so many suggest that you can calculate the number of times Insertion Sort shifts each element when sorting an array.

If ki is the number of elements over which the ith element of the array has to shift, then the total number of shifts will be k1 +k2 +…+kN.

Time complexity is O(N*log(N)) and space complexity is O(1).

There are more solutions with nlogn time for this challenge.

Build a short solution that demonstrate Insertion sort. Below example, demonstrate on what level I expect a solution.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CommonInsertion_Sort
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] numbers = new int[10] {2, 5, -4, 11, 0, 18, 22, 67, 51, 6};
            Console.WriteLine("\nOriginal Array Elements :");
                    PrintIntegerArray(numbers);
            Console.WriteLine("\nSorted Array Elements :");
            PrintIntegerArray(InsertionSort(numbers));
```

```
                    Console.WriteLine("\n");
            }

        static int[] InsertionSort(int[] inputArray)
        {
            for (int i = 0; i < inputArray.Length - 1; i++)
            {
                for (int j = i + 1; j > 0; j--)
                {
                    if (inputArray[j - 1] > inputArray[j])
                    {
                        int temp = inputArray[j - 1];
                        inputArray[j - 1] = inputArray[j];
                        inputArray[j] = temp;
                    }
                  }
            }
            return inputArray;
        }
        public static void PrintIntegerArray(int[] array)
        {
            foreach (int i in array)
            {
                Console.Write(i.ToString() + "  ");
            }
         }

        public static int[] InsertionSortByShift(int[] inputArray)
        {
            for (int i = 0; i < inputArray.Length - 1; i++)
            {
                int j;
                var insertionValue = inputArray[i];
                for (j = i; j > 0; j--)
                {
                    if (inputArray[j - 1] > insertionValue)
                    {
                        inputArray[j] = inputArray[j - 1];
                    }
                }
                inputArray[j] = insertionValue;
            }
            return inputArray;
        }

     }
}
```

```
Sample Output:
Original Array Elements :
2  5  -4  11  0  18  22  67  51  6
Sorted Array Elements :
-4  0  2  5  6  11  18  22  51  67
```