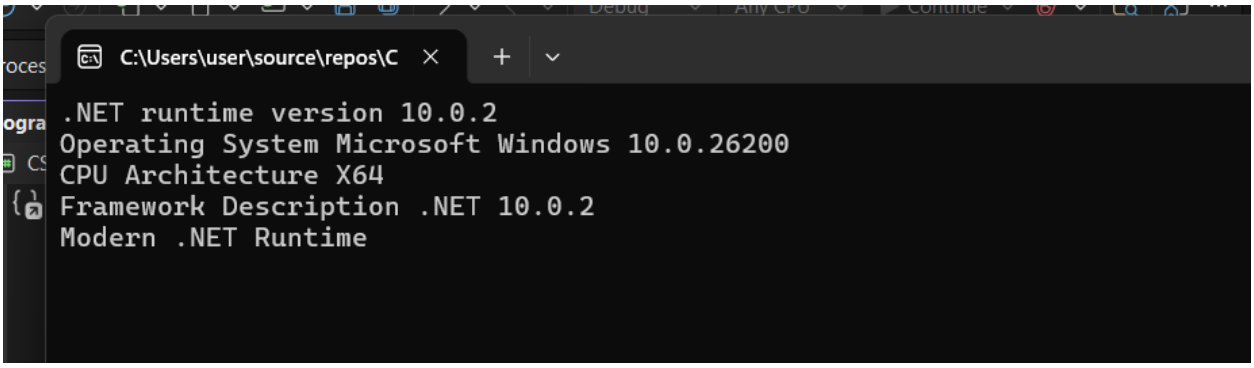
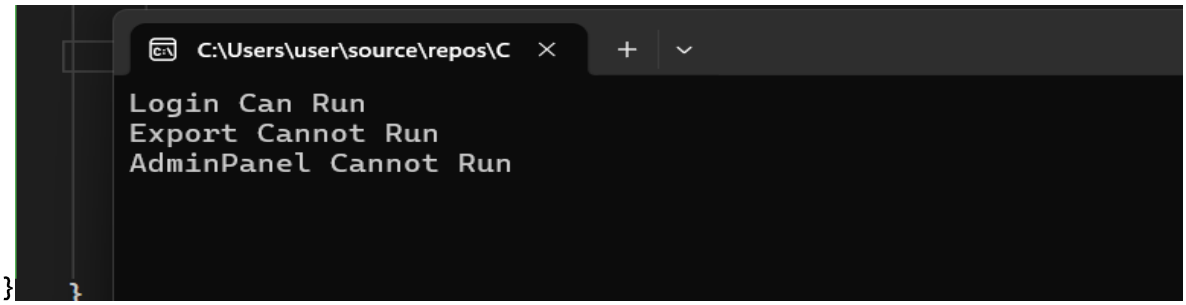
Question 1:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Runtime.InteropServices;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(".NET runtime version "+Environment.Version);
            Console.WriteLine("Operating System " +
RuntimeInformation.OSDescription);
            Console.WriteLine("CPU Architecture
"+RuntimeInformation.OSArchitecture);
            Console.WriteLine("Framework Description " +
RuntimeInformation.FrameworkDescription);
            var s = RuntimeInformation.FrameworkDescription;
            switch (s)
            {
                case string r when r.Contains(".NET"):
                    Console.WriteLine("Modern .NET Runtime");
                    break;
                default:
                    Console.WriteLine("Legacy Runtime");
                    break;
            }
                Console.ReadKey();
        }
    }

}
```



Question 2:

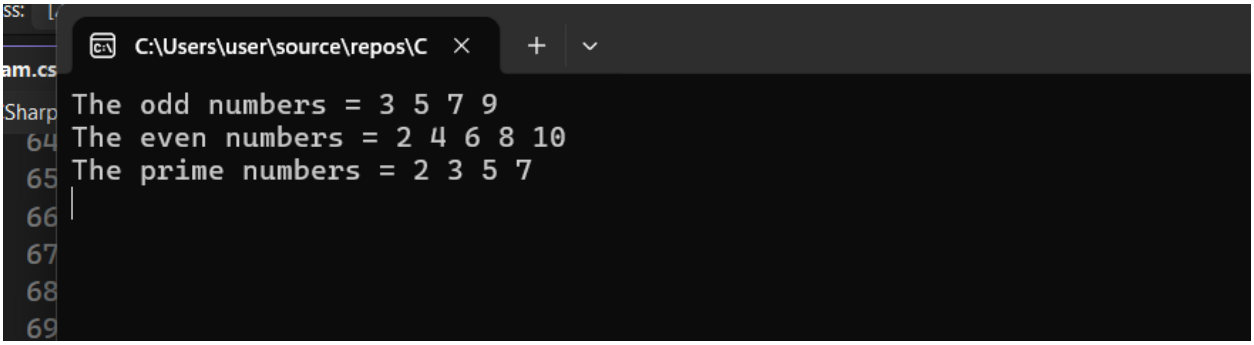
```
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Feature Login = new Feature("Login", true,1.2);
            Login.DisplayCanRun();
            Feature Export = new Feature("Export", false,1.1);
            Export.DisplayCanRun();
            Feature AdminPanel = new Feature("AdminPanel", true,1.3);
            AdminPanel.DisplayCanRun();
            Console.ReadKey();
        }


    }

    public class Feature
    {
        const double Version = 1.2;
        private string name;
        private bool enabled;
        private readonly double minimumRequiredVersion;
        public Feature(string name, bool enabled, double minimumRequiredVersion)
        {
            this.name = name;
            this.enabled = enabled;
            this.minimumRequiredVersion = minimumRequiredVersion;
        }
        public bool CanRun()
        {
            return (enabled && minimumRequiredVersion <= Version);
        }
        public void DisplayCanRun()
        {
            Console.Write(name + " ");
            Console.Write(CanRun() ? "Can Run" : "Cannot Run");
            Console.WriteLine();
        }
    }
}
```



Question 3:

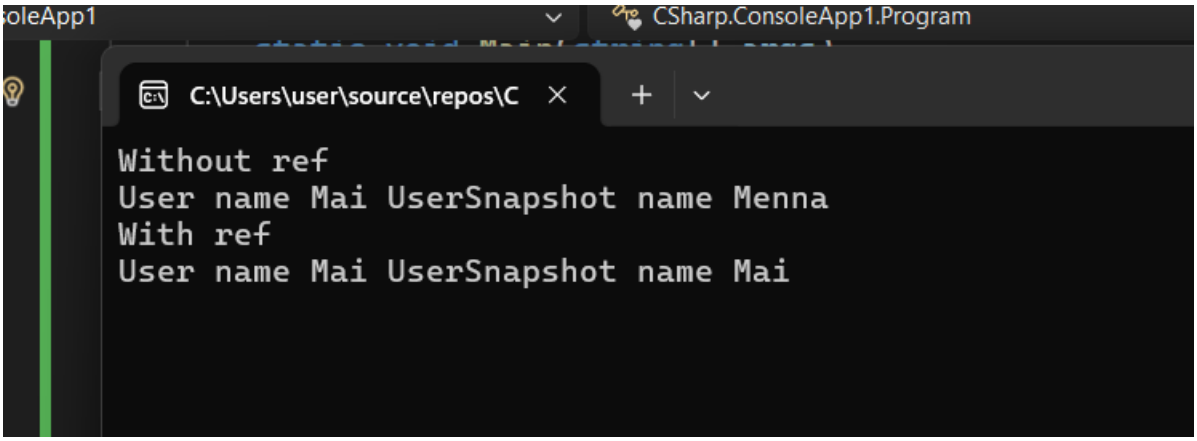
```
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> numbers = new List<int> { 2, 3, 4, 5, 6, 7, 8, 9, 10 };

            Numbers num =new Numbers();
            num.DisplayNumbers(numbers);
            Console.ReadKey();
        }


    }
    public class NumberResult
    {
        public List<int> evennumders = new List<int>();
        public List<int> oddnumders = new List<int>();
        public List<int> primenumders = new List<int>();
    }
    public class Numbers
    {
        public NumberResult GetNumbers(List<int> l)
        {
            NumberResult result = new NumberResult();

            foreach (var x in l)
            {
                if (x % 2 == 0)
                    result.evennumders.Add(x);
                else result.oddnumders.Add(x);

                if(Prime(x))
                    result.primenumders.Add(x);
            }
            return result;
        }
        private bool Prime (int x)
        {
            bool check = true;
            if (x <= 1) check = false;
            else
            {
                for (int i = 2; i < x; i++)
                {
                    if (x % i == 0)
                    {
                        check = false;
                        break;
                    }
                }
```

```
            }
            return check;
        }
        public void DisplayNumbers(List<int> l)
        {
            var v=GetNumbers(l);

            Console.Write("The odd numbers = ");
            foreach (var x in v.oddnumders)
            {
                Console.Write(x + " ");
            }
            Console.WriteLine();
            Console.Write("The even numbers = ");
            foreach (var x in v.evennumders)
            {
                Console.Write(x + " ");
            }
            Console.WriteLine();
            Console.Write("The prime numbers = ");
            foreach (var x in v.primenumders)
            {
                Console.Write(x + " ");
            }
            Console.WriteLine();
        }
    }
}
```



Question 4:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Runtime.InteropServices;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            User user = new User();
            UserSnapshot usersnapshot = new UserSnapshot();
            ModifyData(user, usersnapshot);
            Console.WriteLine("Without ref");
            Console.WriteLine($"User name {user.Name} UserSnapshot name
{usersnapshot.Name}");

            ModifyDataRef(user, ref usersnapshot);
            Console.WriteLine("With ref");
            Console.WriteLine($"User name {user.Name} UserSnapshot name
{usersnapshot.Name}");


            Console.ReadKey();
        }

        static void ModifyData(User user, UserSnapshot userSnapshot)
        {
            user.Name = "Mai";
            userSnapshot.Name = "Mai";

        }

        static void ModifyDataRef(User user, ref UserSnapshot userSnapshot)
        {
            user.Name = "Mai";
            userSnapshot.Name = "Mai";

        }
    }

    public class User
    {
        public string Name = "Menna";

    }

    public struct UserSnapshot
    {
        public string Name;
        public UserSnapshot()
        {
            Name = "Menna";
        }
    }

}
```



What changed? Why?

The Name of Struct ‘UserSnapshot’ did not actually change when passed to the method normally because struct is value type any modification inside the method is not applied to the original struct without using ref keyword.

Stack vs Heap

Stack:

Fast memory allocation

Stores local variables and value types

Memory is automatically released when the method ends

Heap:

Slower than stack

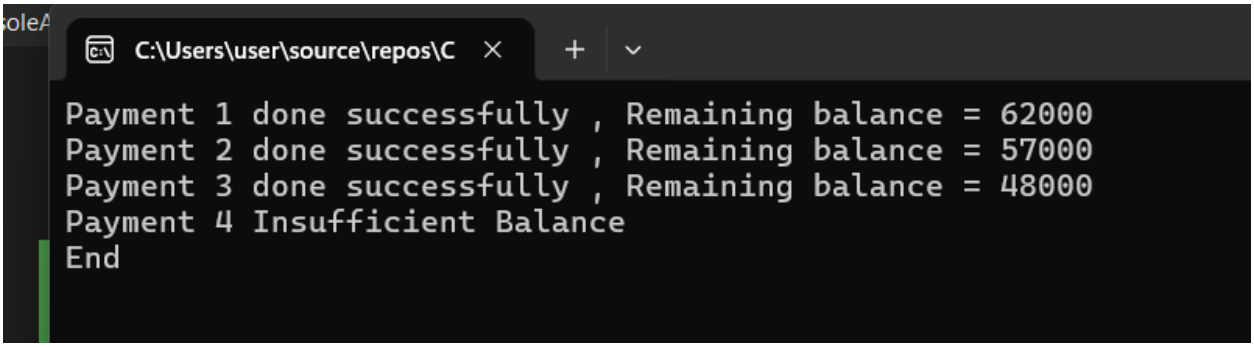
Stores objects and class instances (reference types)

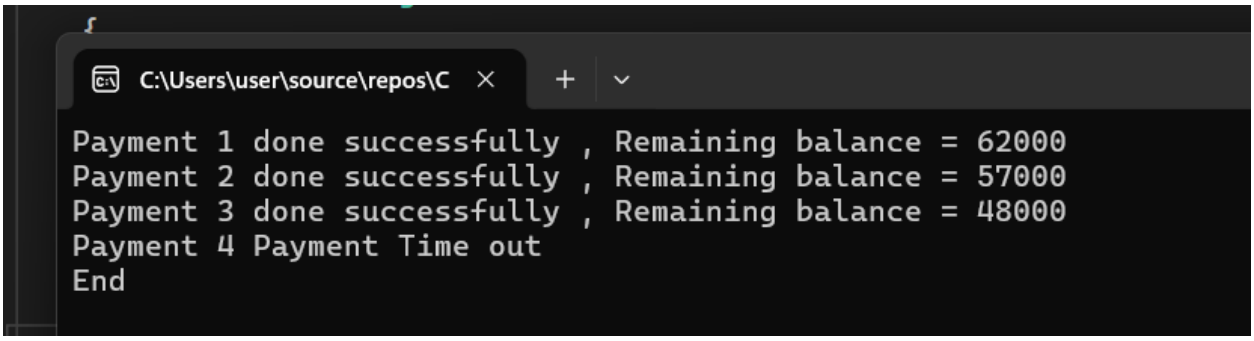
Managed by the Garbage Collector

Question 5:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Runtime.InteropServices;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            PaymentSystem payment = new PaymentSystem(1001, "Menna", 2299, 70000);
            try
            {
                payment.MakePayment(8000,7);
                payment.MakePayment(5000, 7);
                payment.MakePayment(9000, 7);
                //payment.MakePayment(90000, 7);
                payment.MakePayment(9000, 11);
            }

            catch (InsufficientBalanceException ex)
            {
                Console.WriteLine(ex.Message);
            }
            catch (PaymentTimeoutException ex)
            {
                Console.WriteLine(ex.Message);
            }

            finally
            {
                Console.WriteLine("End");

            }
             Console.ReadKey();
        }
    }
    public class PaymentSystem
    {
        const int MaxTime = 10;
        private static int OperationCounter = 0;
        public int ID { get; set; }
        public string Name { get; set; }
        public int AccountNumber { get; set; }
        public int Balance { get; set; }
        public PaymentSystem(int id,string name,int accountnumber,int balance)
        {
            ID = id;
            Name = name;
            AccountNumber = accountnumber;
            Balance = balance;
        }
        public void MakePayment(int balance,int paymenttime)
        {
            int currentOperation = ++OperationCounter;
            if (balance > Balance)
            {
```

```
                throw new InsufficientBalanceException($"Payment {currentOperation}
Insufficient Balance");

            }
            if(paymenttime > MaxTime)
            {
                throw new PaymentTimeoutException($"Payment {currentOperation}
Payment Time out");
            }
            Balance -= balance;
            Console.WriteLine($"Payment {currentOperation} done successfully ,
Remaining balance = {Balance}");
        }
    }
    public class InsufficientBalanceException : Exception
    {
        public InsufficientBalanceException(string message) : base(message) { }
    }
    public class PaymentTimeoutException : Exception
    {
        public PaymentTimeoutException(string message) : base(message) { }
    }
}
```

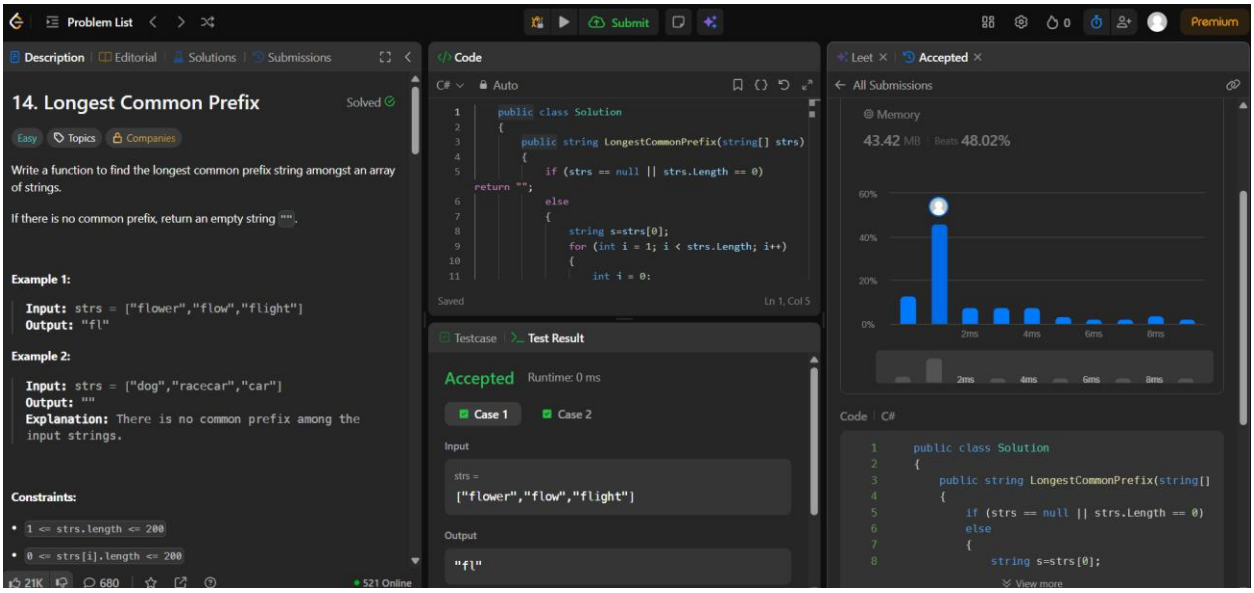




Part 2:

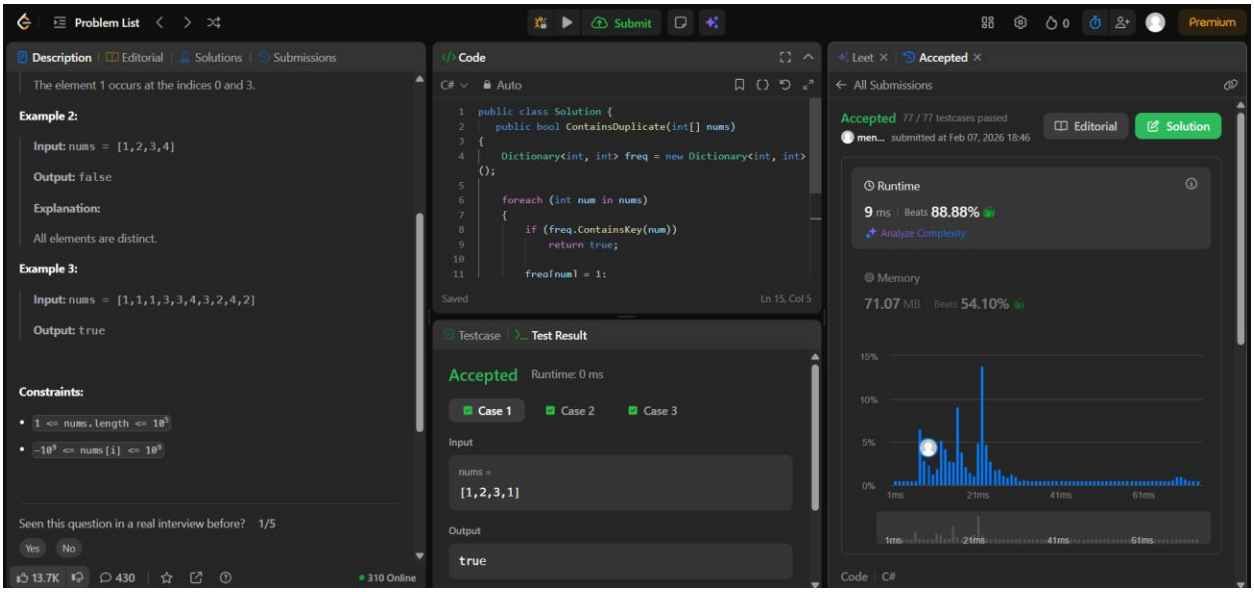
Problem 1:

```
public class Solution
{
    public string LongestCommonPrefix(string[] strs)
    {
        if (strs == null || strs.Length == 0) return "";
        else
        {
            string s=strs[0];
            for (int i = 1; i < strs.Length; i++)
            {
                int j = 0;
                while (j < s.Length && j < strs[i].Length && s[j] == strs[i][j] )
                {
                    j++;
                }
                s=s.Substring(0,j);
                if (s == "")
                    break;
            }
            return s;
        }

    }
}
```



Problem 2:

```
public class Solution
{
    public bool ContainsDuplicate(int[] nums)
    {
        Dictionary<int, int> freq = new Dictionary<int, int>();

        foreach (int num in nums)
        {
            if (freq.ContainsKey(num))
                return true;

            freq[num] = 1;
        }

        return false;
    }
}
```



Problem 3:

```
public class Solution
 {
     public bool IsAnagram(string s, string t)
     {
         if (s.Length != t.Length)
             return false;
         int[]freq = new int[26];
         for (int i = 0; i < s.Length; i++)
         {
             freq[s[i] - 'a']++;
             freq[t[i] - 'a']--;
         }
         for (int i = 0; i < 26; i++)
         {
             if (freq[i] > 0)
                 return false;
         }
         return true;
     }
 }
```