1) You have two arrays/lists as follows

int[] list1 = new int[] { 1, 2, 3, 4, 5 };
int[] list2 = new int[] { 3, 4, 5, 6, 7 };

a. Show the common elements in both lists. E.g the common elements are "3 4 5", because they are contained in both lists.
b. Show the elements from list 1, but is not found in list2. E.g "1 2"
c. Show the elements from list 2, but is not found in list1. E.g 6 7"

```
//************************************************************

using System;
using System.Collections.Generic;

public class Program
{
        public static void Main()
        {
                //These are the 2 arrays
        int[] list1 = new int[] { 1, 2, 3, 4, 5 };
        int[] list2 = new int[] { 3, 4, 5, 6, 7 };

        //Start Answer for (a)
        int[] list3 = getCommonElements(list1, list2);

        Console.WriteLine(string.Join(" ", list3));

           //End Answer for (a)

        //Start Answer for (b)

        int[] list4 = getLeftJoinElements(list1, list3);
        Console.WriteLine(string.Join(" ", list4));
        //End Answer for (b)


       //Start Answer for (c)

       int[] list5 = getLeftJoinElements(list2, list3);
       Console.WriteLine(string.Join(" ", list5));
       //End Answer for (c)

       Console.WriteLine("Press <ENTER> to continue");
       Console.ReadLine();
     }

     private static int[] getCommonElements(int[] list1, int[] list2) {
        var result = new List<int>();
        var dictionary = new Dictionary<int, bool>();
```

```
        foreach (var value in list1) {
            dictionary[value] = true;
        }

        foreach (var value in list2) {
            if (dictionary.ContainsKey(value)) {
                result.Add(value);
            }
        }

        return result.ToArray();
    }

    private static int[] getLeftJoinElements(int[] list1, int[] commonElements) {
        var result = new List<int>();

                        var dictionary = new Dictionary<int, bool>();
        foreach (var value in commonElements) {
            dictionary[value] = true;
        }

        foreach (var value in list1) {
            if (!dictionary.ContainsKey(value)) {
                result.Add(value);
            }
        }

        return result.ToArray();
    }
}
```

2) This question is made up a small program and 4 classes based on a Vehicle hierarchy

a. Change the class 'Entity' so that you cannot directly instantiate the class.
b. A vehicle model is linked to a vehicle make. For example A180 avantgarde model's parent is make MercedesBenz. Change the vehicle model class to reflect this
c. The VehicleModel will have many VehiclePrices (Array or list). Change the VehicleModel class to reflect this.
d. Add proper constructors to the classes VehicleMake and VehicleModel to enforce that the description is always set and is not null or empty. If it is null or empty you should throw an exception. The constructor of VehicleModel should also take in a parameter for the parent (VehicleMake).
e. Change the Entity base class to be more generic in that the Id property is not always be an int. Then adjust the classes inheriting from this base class accordingly. The ID type for VehicleMake and Vehicle model is still int but the ID type for VehiclePrice is Guid.
f. Add another constructor to the VehicleModel class that also takes a parameter called currentYearPrice (decimal). If this constructor is called then the array or list of vehicle prices should be instantiated if needed and you should check if this array already has an entry for the current year and if it does you should update that item's price to the currentYearPrice amount, if it is not in the array then you should add is.
g. In the Main method of the Program class create an instance of the VehicleMake class and set the description to be Mercedes-Benz.

h. In the Main method of the Program class create an instance of the VehicleModel class and use the above VehicleMake as 1 parameter and set the description to A180 and set the current year price to be $300000.

```
//*************************************************************

using System;
using System.Collections.Generic;

namespace ConsoleAppTestPart2
{
	public class Program
	{
		public static void Main(string[] args)
		{
			Console.WriteLine("Start");

			//implementation for g
			var vehicleMake = new VehicleMake("Mercedes-Benz");

	//implementation for h
			var vehicleModel = new VehicleModel(vehicleMake, "A180", 300000);

			Console.WriteLine("Complete");
			Console.ReadLine();
		}
	}

	/// <summary>
	/// base class for IDs
	/// </summary>
	public class Entity<T>
	{
		public T Id { get; set; }

		protected Entity(){}
	}

	/// <summary>
	/// examples of makes of vehicle are Mercedes-Benz, Audi, Toyota, etc
	/// </summary>
	public class VehicleMake: Entity<int>
	{
		private string description { get; set; }
		public string Description {
			get { return description; }
			set {
				if (string.IsNullOrEmpty(value)) {
					throw new Exception("Description can not be null or empty");
				} else {
					this.description = value;
```

```
                }
            }
        }

        private VehicleMake(){}
        public VehicleMake(string description){
            if (string.IsNullOrEmpty(description)) {
                throw new Exception("Description can not be null or empty");
            } else {
                this.description = description;
            }
        }
}

/// <summary>
/// examples of vehicle models are A 180 avantgarde or Corolla 1.6 GLE
/// </summary>
public class VehicleModel: Entity<int>
{
        public VehicleMake VehicleMake { get; set; }
        public List<VehiclePrice> VehiclePrices { get; set; }

        private string description { get; set; }
        public string Description {
            get { return description; }
            set {
                if (string.IsNullOrEmpty(value)) {
                    throw new Exception("Description can not be null or empty");
                } else {
                    this.description = value;
                }
            }
        }

        private VehicleModel(){}
        public VehicleModel(string description, VehicleMake vehicleMake){
            this.VehicleMake = vehicleMake;
            if (string.IsNullOrEmpty(description)) {
                throw new Exception("Description can not be null or empty");
            } else {
                this.description = description;
            }
        }

        public VehicleModel(VehicleMake vehicleMake, string description, decimal currentYearPrice)
        : this (description, vehicleMake) {
            var year = DateTime.Now.Year;
            if (VehiclePrices == null) VehiclePrices = new List<VehiclePrice>();

            var index = VehiclePrices.FindIndex(x => x.Year == year);
            if (index < 0) {
```

```
                        VehiclePrices.Add(
                                new VehiclePrice() {
                                        Id = Guid.NewGuid(),
                                        Year = year,
                                        Price = currentYearPrice
                                }
                        );
                } else {
                        var array = VehiclePrices.ToArray();
                        array[index].Price = currentYearPrice;
                        VehiclePrices.Clear();
                        VehiclePrices.AddRange(array);
                }
        }
    }

    /// <summary>
    /// This has the price or the vehicle for a given year
    /// </summary>
    public class VehiclePrice: Entity<Guid>
    {
        public int Year { get; set; }
        public decimal Price { get; set; }
    }

}
```