This assessed exercise has 2 assessed tasks: Task A and Task B (next page)

Task A

Task A is composed by two parts: 1 and 2. Before completing the task please read the "hints to complete the task" that you find below.

Part 1)

Show how to calculate the **Big O notation** for the segment of C# code shown below

```
int n = 50;
int r = 20;
int dummy = 0;

Console.WriteLine("Creating a nested loop");

for (int i = 0; i < n; i++)
{
    Console.WriteLine("Inside the first loop");

    dummy++;

    for (int j = 0; j < n; j++)
    {

            r = r + dummy;
            Console.WriteLine("Inside the second loop " + r);


    }

}
```

Part 2)

The code below implements the *Selection Sort* for an array of int

```
static public void SelectionSort(int[] a)
{
   for (int i = 0; i < a.Length -1 ; i++)
   {
      int smallest = i;
      for (int j = i + 1; j < a.Length; j++)
      {
        if (a[j] < a[smallest])
              smallest = j;
      }
      swap(ref a[i], ref a[smallest]);
    }
}
```

where *swap(ref int x, ref int y)* is the method that exchanges the values of x and y

```
static void swap(ref int x, ref int y)
    {
       int temp = x;
       x = y;
       y = temp;
    }
```

It is very useful to implement generic functions that can be used sort different defined data types.

Modify the code above to implement in C# **Selection Sort** to be a **generic** function *SelecSortGen* that can sort an array of ***any*** IComparable objects.

Implement a class *Book* with members as *ISBN* (string), *Title* (string) and *Author* (string). Use the starting code "GenSelectSort" you can find on Moodle (you will need to complete it)

- Show that you can use the implemented function *SelecSortGen* to sort an **array of int**, an **array of strings** and an **array of books** (in this case by either using the ISBN, the title or the author).

Hints to complete the task: For Part 1 watch the recorded videos on Big O notation. As you can see in the videos you need to start the calculation of the Big O notation by computing how many times each line of the code will be encountered as the code runs. The videos should be sufficient to solve this part; however we are also going to see more examples of Big O notation in the webinars.
For part 2 use the starting code "GenSelectSort" that you can find on Moodle; remember how you use IComparable and CompareTo that you have seen in the previous weeks. You can also watch the recorded videos in "Sorting Algorithms" to see the idea behind the selection sort. We are also going to study and implement a variety of sorting algorithms in the webinars.

Task B

In a company there is a **single** computer lab which needs to be shared among different requests.
As there are restrictions the use of the computer lab must be **exclusive**: the computer lab can only satisfy **at most one** request at a time.

Implement in C# an efficient application (with a user interface) which allows to:

- **Insert** requests for the use of the computer lab. Each request is composed by:
    - **ID** of the request (assume each request has an unique ID, which is an int),
    - **starting** and **finishing times** of the request (eg., a request could be : ID = "2964" , starting time = 12.00, finishing time = 17.00).

- **Display** the **largest possible set of requests** that can be satisfied.

The application should allow to insert requests in an arbitrarily manner (i.e., not necessarily in the order of starting or finishing time).

The user interface should be a GUI - use a Windows Forms application (you could choose to display the result in a textbox or listbox). Before completing the task please read the "hints to complete the task" that you find below.

Example

Given the following requests for the computer lab

ID = 1, starting time = 12:00, finishing time = 13:45

ID = 2, starting time = 12:30, finishing time = 12:45

ID = 3, starting time = 16:30, finishing time = 16:45

ID = 4, starting time = 13:40, finishing time = 15:35

ID = 5, starting time = 12:10, finishing time = 14:30

ID = 6, starting time = 12:05, finishing time = 15:10

It is possible to see that some requests have overlapping times (eg, the request with ID = 1 and the request with ID = 2).

However, as the computer lab must be used in an exclusive way, i.e., can only satisfy at most one request at a time, then the **largest possible set of requests** that can be satisfied has 3 elements which correspond to the requests with these IDs:
ID =2, ID = 3, ID = 4

Hints to complete the task: You could create a class Request (that contains ID, starting and finishing time of a request) and store the received requests in an array of objects of type Request. For simplicity, assume that it is not necessary to consider the date of the request but only the starting and finishing time. Do not implement an algorithm which tries all the possibilities (that would be very inefficient!) but think to a Greedy algorithm. To complete this task watch the recorded materials on "Greedy Algorithms". The video should be sufficient for you to solve this task; however we are also going to see more on greedy algorithms in the webinars.