# C# Advanced Lab - Algorithms

This document defines **algorithmic problems** from the ["Advanced C#" Course @ Software University](http://softuni.bg/courses/advanced-csharp/). You are presented with some problems and certain steps you need to take in order to accomplish the tasks.

## Linear and Binary Search

There are two standard array searching algorithms - **Linear** and **Binary** Search.

* Linear search traverses the entire collection until the searched element is found.
* Binary search works only on **sorted collections**. It picks the **mid element** of the collection and checks if it's equal to the searched element.
  + If it's **equal**, returns the mid index.
  + If it's **smaller**, cuts the right half of the collection and repeats the same step.
  + If it's **larger**, cuts the left half of the collection and repeats the same step.

If **no such element is found**, both algorithms should return **-1** as result.

|  |
| --- |
| **Linear Search vs Binary Search Comparison** |
|  |
| **Animated gif***:*  [*https://blog.penjee.com/wp-content/uploads/2015/04/binary-and-linear-search-animations.gif*](https://blog.penjee.com/wp-content/uploads/2015/04/binary-and-linear-search-animations.gif) |

1 3 5 7 11 13 17 19 23 29 31 37 41 43 53

Your task is to implement both Linear Search and Binary Search algorithms using C#. Create respective methods for them - **LinearSearch(array, element)** and **BinarySearch(array, element)**.

The solution for **Linear search** is as follows:

* Iterate through each element of the collection
  + Check if its equal to the searched element. Return index if it is.
  + Continue iterating if not.
  + Return **-1** if no such element is found.

One solution for **Binary search** is as follows:

* Define **min=0** and **max=A.length-1**
* Until there is at least 1 element in range:
  + Get **mid** index in range **[min…max]** and check if **A[mid]** is equal to **search element**
    - If it's **equal**, return **mid** index
    - If it's **larger**, repeat iteration by **ignoring all elements** to the **left** of **mid**
    - If it's **smaller**, repeat iteration by **ignoring all elements** to the **right** of **mid**

In case of **duplicate elements**, both algorithms should return the **leftmost index** (see example 3 below).

### Constraints

* The input list will hold integers in the range [−2147483648 … 2147483647].
* You are **NOT allowed** to use **.IndexOf()**, **Array.BinarySearch()** or similar methods. Write **your own** Linear and Binary search algorithms.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| -2 0 3 5 213 8582 239191 985128  239191 | **6** |
| 0 1 2 3 4 5 6 6 7 8  -2 | **-1** |
| 3 9 10 12 13 13 13 13  13 | **4** |