LAB211Assignment

Type:	**Short Assignment**
Code:	**J1.S.P0006**
LOC:	**40**
Slot(s):	**1**

Title

Binary search algorithm.

Background Context

Generally, to find a value in unsorted array, we should look through elements of an array one by one, until searched value is found. In case of searched value is absent from array, we go through all elements. In average, complexity of such an algorithm is proportional to the length of the array.

Situation changes significantly, when array is sorted. If we know it, random access capability can be utilized very efficiently to find searched value quick. Cost of searching algorithm reduces to binary logarithm of the array length. For reference, $\log_2(1\ 000\ 000) \approx 20$. It means, that **in worst case**, algorithm makes 20 steps to find a value in sorted array of a million elements or to say, that it doesn't present it the array.

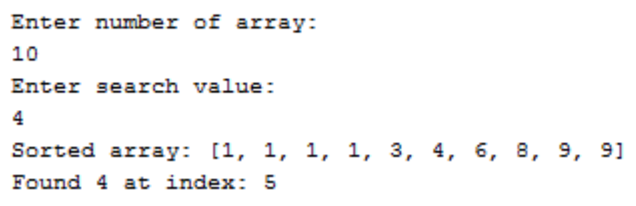
Program Specifications

Design a program that allows users to input the number of array. **Generate random** integer in number range input. After that allows users to input search number. Display **sorted array** and **index of search number** in array.

Function details:

1. Display a screen to prompt users to input a positive decimal number.
    - Users run the program, display a screen to ask users to enter a number of array and a search number.
    - Users input a positive decimal number. Then, perform **Function 2**.
2. Display the found index in array.
    - Generate random integer in number range for each array element.
    - Sort array
    - Display the index of search number in array.

Expectation of User interface:



```
Enter number of array:
10
Enter search value:
4
Sorted array: [1, 1, 1, 1, 3, 4, 6, 8, 9, 9]
Found 4 at index: 5
```

Guidelines

Algorithm

Algorithm is quite simple. It can be done either recursively or iteratively:

1. get the middle element;
2. if the middle element equals to the searched value, the algorithm stops;
3. otherwise, two cases are possible:
    - searched value is less than the middle element. In this case, go to the step 1 for the part of the array, before middle element.
    - searched value is greater than the middle element. In this case, go to the step 1 for the part of the array, after middle element.

Now we should define, when iterations should stop. First case is when searched element is found. Second one is when subarray has no elements. In this case, we can conclude, that searched value doesn't present in the array.

Examples

Example 1. Find 6 in {-1, 5, 6, 18, 19, 25, 46, 78, 102, 114}.

Step 1 (middle element is 19 > 6): -1 5 6 18 **19** 25 46 78 102 114

Step 2 (middle element is 5 < 6): -1 **5** 6 18 19 25 46 78 102 114

Step 3 (middle element is 6 == 6): -1 5 **6** 18 19 25 46 78 102 114

Example 2. Find 103 in {-1, 5, 6, 18, 19, 25, 46, 78, 102, 114}.

Step 1 (middle element is 19 < 103): -1 5 6 18 **19** 25 46 78 102 114

Step 2 (middle element is 78 < 103): -1 5 6 18 19 25 46 **78** 102 114

Step 3 (middle element is 102 < 103): -1 5 6 18 19 25 46 78 **102** 114

Step 4 (middle element is 114 > 103): -1 5 6 18 19 25 46 78 102 **114**

Step 5 (searched value is absent): -1 5 6 18 19 25 46 78 102 114