15 - Second Largest

Question

Given an array of positive integers arr[], return the second largest element from the array. If the second largest element doesn't exist then return -1.

Note: The second largest element should not be equal to the largest element.

Solution

```
class Solution {
    public int getSecondLargest(int[] arr) {
            // Code Here
            int max1 = Integer.MIN_VALUE;
            int max2 = Integer.MIN_VALUE;
        for(int i=0;i<arr.length;i++){

            if(arr[i]>max1){

                max2 = max1;
                max1 = arr[i];
            }
            else if(arr[i]>max2 && arr[i]!=max1){
                max2 = arr[i];
            }

        }

        if(max1 == max2 || max2 == Integer.MIN_VALUE){
            return -1;
```

```
        }

        return max2;
    }
}
```

Complexities

```
Time Complexity: O(n)

Auxiliary Space: O(1)
```

Resource :

https://www.geeksforgeeks.org/find-second-largest-element-array/

Notes

```
Understanding Problem :
 We have an input arr, which has positive numbers, we have
to find the second-largest element

Approaches :

Thinking: to find the smallest or largest number from an ar
ray, we have to sort the array

Brute Force :
1.Sort the array
2.Return the element from the last index-1

Code:
class solution{

    public static int secondLargest(int[] arr,int n){
        Arrays.sort(arr);
        return arr[n-2];
    }
```

```
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int[n];

        for(int i=0;i<n;i++){
            arr[i] = sc.nextInt();
        }
        Int second_max = secondLargest(arr,n);
    }
}

// But there is a test case where the last two element is s
ame, then it doesn’t become the second largest.
To handle that we have write a for loop

for(int i=n-2;i>0;i–){
    if(arr[i] !=arr[i+1]){
        return arr[i];
    }
}
return -1;

Time complexity: O(n log n)
O(nlogn)- for sorting and O(n) - for traversing array rever
se, so complexity becomes O(nlogn)+O(n)
Space complexity : O(1)


Better Approach:

To reduce the time complexity
1.First find the largest element in the array by passing th
rought the array
2.Then find the second largest element by passing through t
he array by considering the element is not equal largest el
ement and greater than the remaining elements
```

Code:

```
    public static int secondLargest(int[] arr,int n){
        int firstmax = -1;
        for(int i=0;i<n;i++){
            if(arr[i]>firstmax){
                firstmax = arr[i];
            }
        }
        //now we have the largest element
        int secondmax = -1;
        for(int i=0;i<n;i++){
            if(arr[i]>secondmax && arr[i] != firstmax){
                secondmax = arr[i];
            }
        }
        return secondmax;
    }

    Time complexity : O(n)
    o(n) - For first pass and o(n) for second pass. The Tim
e complexity is O(n)+O(n) = o(n);
    Space complexity: O(1)
```

Another Better Approach :
Instead of traversing through the array twice, we can find the second largest by traversing once.

1.initialize two variables one is firstmax and the other is secondmax initialized to -1;
2. If the element is greater than firstmax then we store th e firstmax in secondmax and assign the element to firstmax
3. But there may be a case where the element is lessthan fi rstmax but greaterthan secondmax then we have to assign the element to secondmax
4. at end return the second max

Code:

```
    public static int secondLargest(int[] arr,int n){
        int firstmax = -1,secondmax=-1;
        for(int i=0;i<n;i++){
            if(arr[i]>firstmax){
                secondmax = firstmax;
                firstmax = arr[i];
            }
else if(arr[i]>secondmax && arr[i]<firstmax){
    secondmax = arr[i];
}
        }
        return secondmax;
    }
Time Complexity : O(n);
Space complexity : O(1);

-------------------------------------------------------------
-------------------------

Brute Force:(Sorting Techniques)
we can sort it and return the second last element
Time complexity: based on the sorting technique used.
    - Inbuilt in Java (Tim sort) and cpp - O(nlogn)
Auxiliary Space - O(1)

Better approach:(Two passes through array)
1. First find the maximum element in one pass through the a
rray
2. then in the second pass find the second largest element
by comparing every element if it is less than the first max
imum and greater than the second maximum then it becomes th
e second maximum.
Time complexity: O(n)
Auxiliary Space - O(1)

optimal approach:(one pass through array)
1. if the element is greater than the first maximum, then s
tore the first maximum in the second maximum and it becomes
```

```
the first maximum.
2. if not then check the element is greater than the second
maximum and not equal to the second maximum, then store it
in the second maximum

Time complexity: O(n)
Auxiliary Space - O(1)
```

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

LinkedIn Post

https://www.linkedin.com/feed/update/urn:li:activity:7263203769851523072/

X Post

https://x.com/SatyaPilla6/status/1857439026912837926