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 l
argest 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

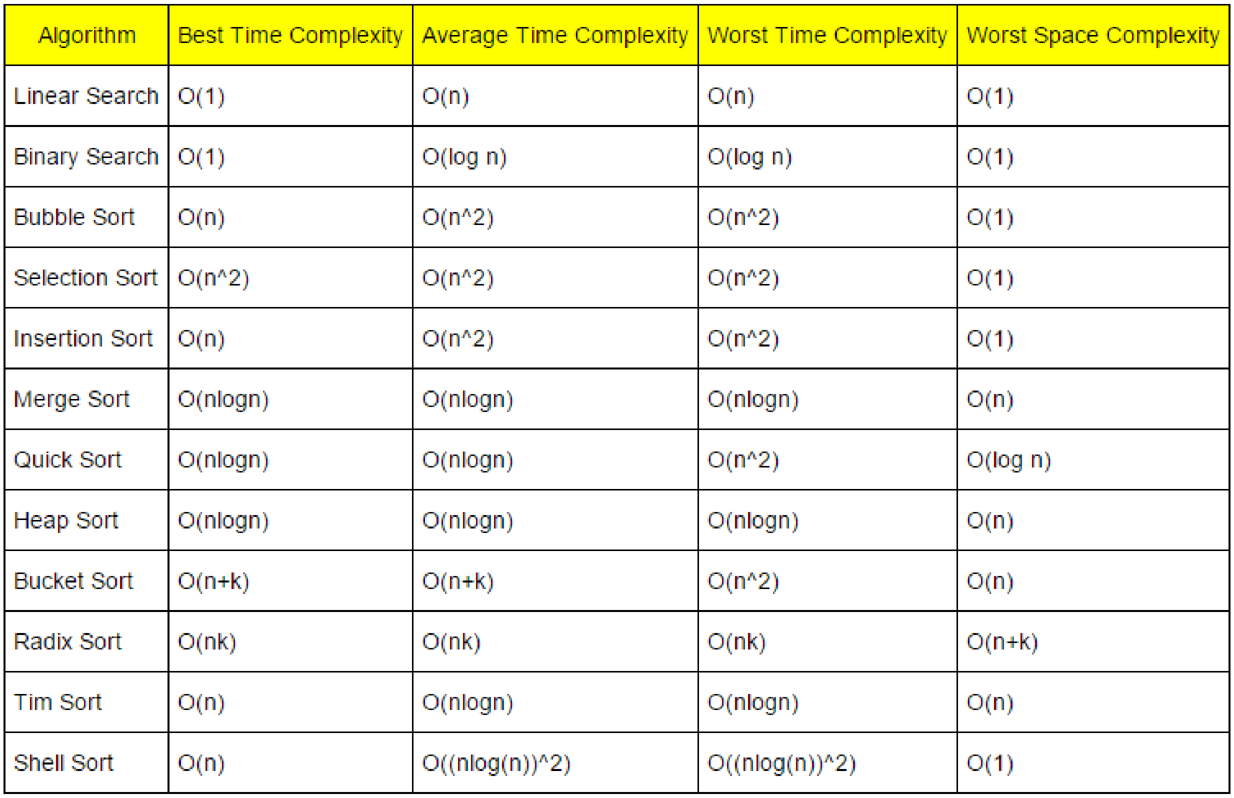
```
Brute Force:(Sorting Techniques)
we can sort it and return the second last element
Time complexity: based on the sorting technique used.
    - Inbuilt in Java (Tim sort) and cpp - O(nlogn)
Auxiliary Space - O(1)

Better approach:(Two passes through array)
1. First find the maximum element in one pass through the arr
ay
2. then in the second pass find the second largest element by
comparing every element if it is less than the first maximum
and greater than the second maximum then it becomes the secon
d 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 store 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