Kth largest/smallest element in an array
Problem Statement: Given an unsorted array, print Kth Largest and Smallest Element from an unsorted array.
Examples:

Example 1:

Input: Array = [1,2,6,4,5,3] , K = 3

Output: kth largest element = 4, kth smallest element = 3

Example 2:

Input: Array = [1,2,6,4,5] , k = 3

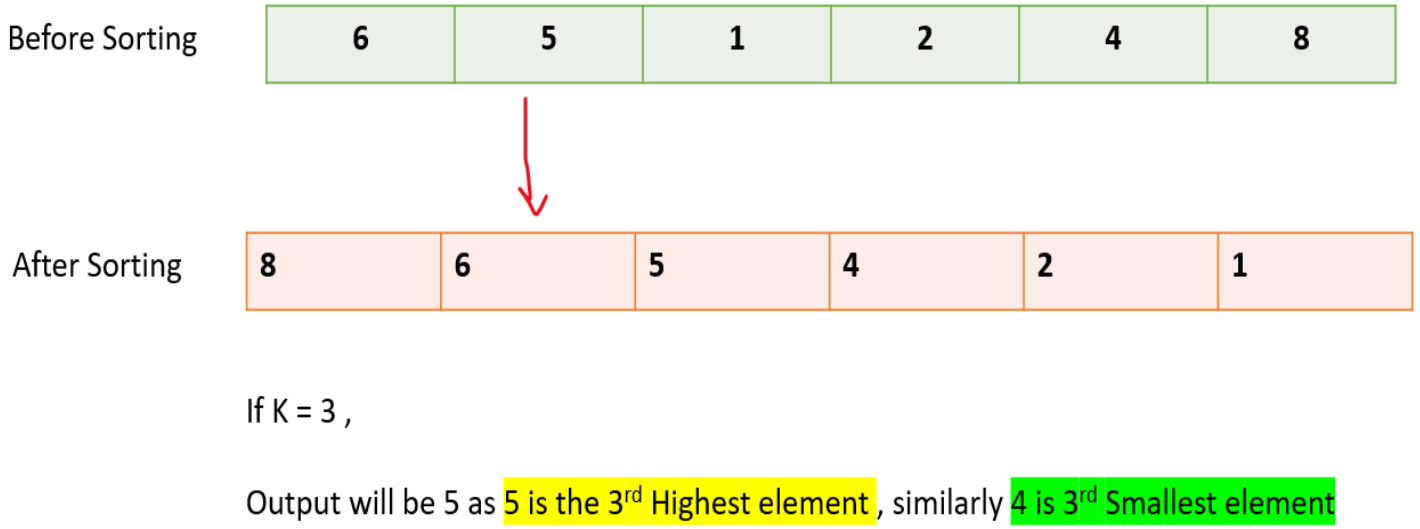
Output : kth largest element = 4, kth smallest element = 4

Solution

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1: Sorting the Array

- The most naive approach is to sort the given array in descending order.
- The index of kth Largest element = k-1 (zero-based indexing)
- The index of kth Smallest element = n-k
- The array can also be sorted in ascending order.
- The index of kth Largest element = n-k
- The index of kth Smallest element = k-1 (zero based indexing)



If K = 3 ,

Output will be 5 as 5 is the 3rd Highest element , similarly 4 is 3rd Smallest element

Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include <bits/stdc++.h>
using namespace std ;

class Solution {

public:

void kth_Largest_And_Smallest_By_AscendingOrder(vector<int>&arr, int k) {

        sort(arr.begin(), arr.end())  ;
        int n = arr.size()  ;

        cout << "kth Largest element " << arr[n - k] << ", " <<
        "kth Smallest element " << arr[k - 1];
    }
} ;
int main() {

    vector<int>arr = {1, 2, 6, 4, 5, 3}  ;
```

```
    Solution obj ;

    obj.kth_Largest_And_Smallest_By_AscendingOrder(arr, 3) ;

    return 0 ;
}
```

Output: kth Largest element 4, kth Smallest element 3

Time complexity: O(nlogn)

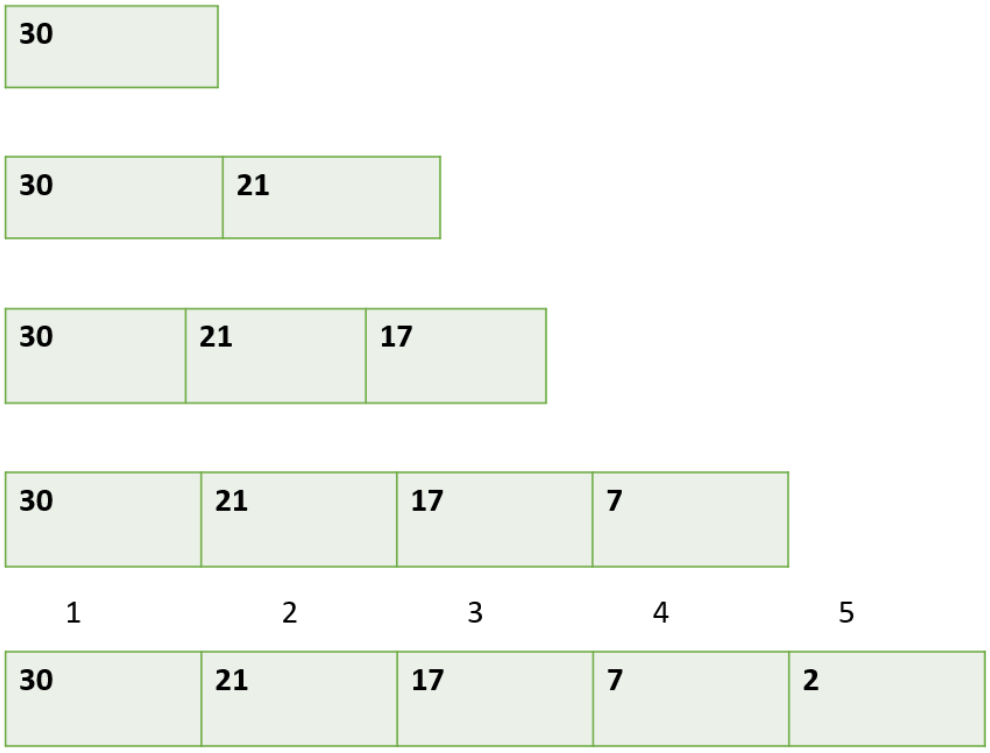
Space complexity: O(1)

Solution 2: Using Heap

●The idea is to construct a max-heap of elements. Since the top element of the max heap is the largest element of the heap, we will remove the top K-1 elements from the heap. The top element will be Kth's Largest element.
●To get the Kth Smallest element, we will use a min-heap. After the removal of the top k-1 elements, the Kth Smallest element is top of the Priority queue.

Let the array be [17,7,2,30,21] and k = 3

Generate a Max-Heap



Now , extract k-1 values

30	21	17	7	1

17	7	1

Print the Top value , The top value is Kth largest.

Here, 17 is the 3rd Largest element

Similarly, for the smallest kth element we will be using Min-Heap. After, extracting the top k-1 values will be having Kth Smallest element.
Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include <bits/stdc++.h>
using namespace std ;

class Solution {

public:

    void kth_Largest_MaxHeap(vector<int>&arr, int k) {

        priority_queue<int>pq ;
        int n = arr.size()  ;

        for (int i = 0; i < arr.size(); i++) {
            pq.push(arr[i])  ;
        }

        int f = k - 1 ;

        while (f > 0) {
            pq.pop()  ;
            f-- ;
        }

        cout << "Kth Largest element " << pq.top() << "\n"  ;
    }

    void kth_Smallest_MinHeap(vector<int>&arr, int k) {

        priority_queue<int, vector<int>, greater<int>>pq ;
        int n = arr.size()  ;

        for (int i = 0; i < arr.size(); i++) {
            pq.push(arr[i])  ;
        }

        int f = k - 1 ;

        while (f > 0) {
            pq.pop()  ;
```

```
            f-- ;
        }

        cout << "Kth Smallest element " << pq.top() << "\n"  ;
    }
} ;
int main() {

    vector<int>arr = {1, 2, 6, 4, 5, 3}  ;

    Solution obj ;
    obj.kth_Largest_MaxHeap(arr, 3)  ;
    obj.kth_Smallest_MinHeap(arr, 3)  ;

    return 0 ;
}
```

Output:

Kth Largest element 4

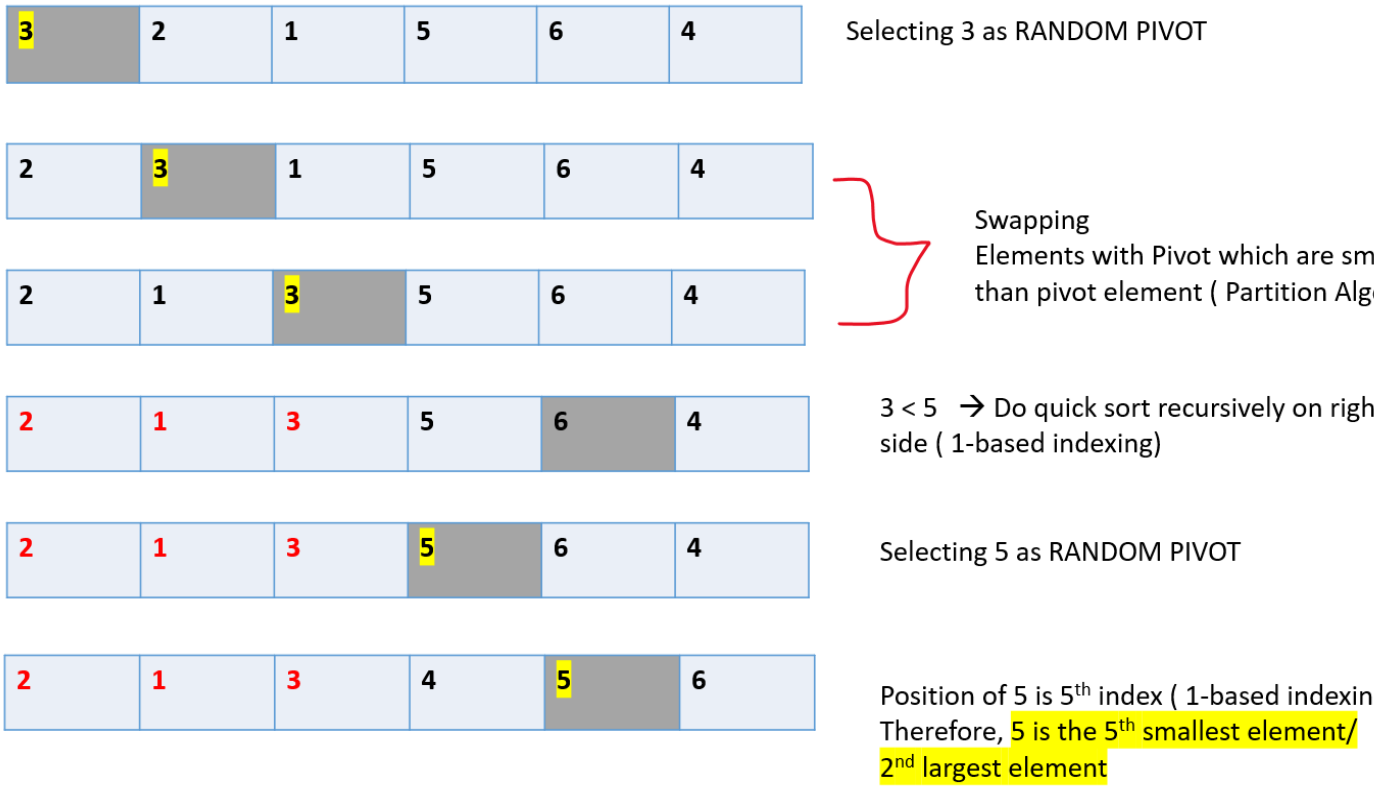
Kth Smallest element 3

Time complexity: O(k+(n-k)*log(k)) , n = size of array

Space complexity: O(k)

Solution 3: Using Quickselect Algorithm

- Choose any random element as PIVOT.
- Use **Partition Algorithm** to partition the given array into 2 parts and place the PIVOT at its correct position (called as index).
- **Partition Algorithm:** All the elements are compared to the PIVOT, and the elements less than the PIVOT are shifted toward the left side of the array and greater ones are shifted toward the right side of the array.
- Now since all elements right to the PIVOT are greater and left to the PIVOT are smaller, compare the index with N-k (where N = size of the array) and recursively find the part to find the Kth largest element.
- The worst-case time complexity of this method is O(n^2) but its Average time complexity is O(n).



Code for kth largest element:

- C++ Code
- Java Code
- Python Code

```cpp
#include <bits/stdc++.h>
using namespace std ;

int partition(vector<int>& arr, int left, int right) {
    int pivot = arr[left] ;
    int l = left + 1 ;
    int r = right;
    while (l <= r) {
        if (arr[l] < pivot && arr[r] > pivot) {
            swap(arr[l], arr[r]);
            l++ ;
            r-- ;
        }
        if (arr[l] >= pivot) {
            l++;
        }
        if (arr[r] <= pivot) {
            r--;
        }
    }
    swap(arr[left], arr[r]);
    return r;
}

int kth_Largest_Element(vector<int>& arr, int k) {
    int left = 0, right = arr.size() - 1, kth;
    while (1) {
        int idx = partition(arr, left, right);
        if (idx == k - 1) {
            kth = arr[idx];
            break;
        }
        if (idx < k - 1) {
            left = idx + 1;
        } else {
            right = idx - 1;
        }
    }
    return kth;
}



int main() {

    vector<int>arr ;
    arr.push_back(12) ;
    arr.push_back(3)  ;
    arr.push_back(5) ;
    arr.push_back(7)  ;
    arr.push_back(4) ;
    arr.push_back(19)  ;
    arr.push_back(26) ;

    int n = arr.size(), k = 1;
    cout << "Kth Largest element is " << kth_Largest_Element(arr, k);
    return 0 ;
}
```

Output: Kth Largest element is 26

Time complexity: O(n) , where n = size of the array

Space complexity: O(1)

code for Kth Smallest element:

- C++ Code
- Java Code
- Python Code

```cpp
#include<bits/stdc++.h>
using namespace std;

int partition(vector<int>&arr, int l, int r)
{
	int f = arr[r] ;
	int i = l;

	for (int j = l; j <= r - 1; j++) {
		if (arr[j] <= f) {
			swap(arr[i], arr[j]) ;
			i++;
		}
	}
	swap(arr[i], arr[r]);
	return i;
}


int kth_Smallest_Element(vector<int>&arr, int l, int r, int k)
{
	if (k <= r - l + 1 && k > 0) {

		int ind = partition(arr, l, r);


		if (ind - l == k - 1) {
			return arr[ind];
		}
		if (ind - l > k - 1) {
			return kth_Smallest_Element(arr, l, ind - 1, k);
		}


		return kth_Smallest_Element(arr, ind + 1, r, k - ind + l - 1);
	}
	return INT_MAX;
}

int main()
{

	vector<int>arr ;
	arr.push_back(12) ;
	arr.push_back(3)  ;
	arr.push_back(5) ;
	arr.push_back(7)  ;
	arr.push_back(4) ;
	arr.push_back(19)  ;
	arr.push_back(26) ;

	int n = arr.size(), k = 1;
	cout << "Kth smallest element is " << kth_Smallest_Element(arr, 0, n - 1, k);
	return 0;
}
```

Output: Kth smallest element is 3

Time complexity: O(n) ,where n = size of the array

Space complexity: O(1)