The Top K Problem

**The Top K Problem - Approach 1**

Use the Heap data structure to obtain Top K’s largest or smallest elements.

Solution of the Top K largest elements:

1. Construct a Max Heap.
2. Add all elements into the Max Heap.
3. Traversing and deleting the top element (using pop() or poll() for instance), and store the value into the result array T.
4. Repeat step 3 until we have removed the K largest elements.

Solution of the Top K smallest elements:

1. Construct a Min Heap.
2. Add all elements into the Min Heap.
3. Traversing and deleting the top element (using pop() or poll() for instance), and store the value into the result array T.
4. Repeat step 3 until we have removed the K smallest elements.

**Complexity Analysis:**

Time complexity: O(K log N +N)

* Steps one and two require us to construct a Max Heap which requires O(N) time using the previously discussed heapify method. Each element removed from the heap requires O(logN) time; this process is repeated K times. Thus the total time complexity is O(KlogN+N).

Space complexity: O(N)

After step 2, the heap will store all N elements.

**The Top K Problem - Approach 2**

Use the **Heap** data structure to obtain Top K’s largest or smallest elements.

Solution of the Top K largest elements:

1. Construct a Min Heap with size K.
2. Add elements to the Min Heap one by one.
3. When there are K elements in the “Min Heap”, compare the current element with the top element of the Heap:
4. If the current element is no larger than the top element of the Heap, drop it and - proceed to the next element.
5. If the current element is larger than the Heap’s top element, delete the Heap’s top element, and add the current element to the Min Heap.
6. Repeat Steps 2 and 3 until all elements have been iterated.

Now the K elements in the Min Heap are the K largest elements.

Solution of the Top K smallest elements:

1. Construct a Max Heap with size K.
2. Add elements to the Max Heap one by one.
3. When there are K elements in the “Max Heap”, compare the current element with the top element of the Heap:
4. If the current element is no smaller than the top element of the Heap, drop it and proceed to the next element.
5. If the current element is smaller than the top element of the Heap, delete the top element of the Heap, and add the current element to the Max Heap.
6. Repeat Steps 2 and 3 until all elements have been iterated.

Now the K elements in the Max Heap are the K smallest elements.

**Complexity Analysis:**

Time complexity: O(NlogK)

* Steps one and two will require O(KlogK) time if the elements are added one by one to the heap, however using the heapify method, these two steps could be accomplished in O(K) time. Steps 3 and 4 will require O(logK) time each time an element must be replaced in the heap. In the worst-case scenario, this will be done N−K times. Thus the total time complexity is O((N−K)log K + Klog K) which simplifies to O(NlogK).

Space complexity: O(K)

* The heap will contain at most K elements at any given time.