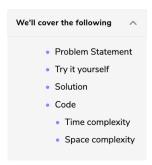
Kth Largest Number in a Stream (medium)



Problem Statement

Design a class to efficiently find the Kth largest element in a stream of numbers.

The class should have the following two things:

- 1. The constructor of the class should accept an integer array containing initial numbers from the stream and an integer 'K'.
- 2. The class should expose a function add(int num) which will store the given number and return the Kth largest number.

Example 1:

```
Input: [3, 1, 5, 12, 2, 11], K = 4
1. Calling add(6) should return '5'.
2. Calling add(13) should return '6'.
2. Calling add(4) should still return '6'.
```

Try it yourself

Try solving this question here:

```
import java.util.*;

class KthLargestNumberInStream {

public KthLargestNumberInStream(int[] nums, int k) {

public int add(int num) {

// TODO: Write your code here

}

public int add(int num) {

// TODO: Write your code here

return -1;

}

public static void main(String[] args) {

int[] input = new int[] { 3, 1, 5, 12, 2, 11 };

KthLargestNumberInStream kthLargestNumber = new KthLargestNumberInStream(input, 4);

System.out.println("4th largest number is: " + kthLargestNumber.add(6));

System.out.println("4th largest number is: " + kthLargestNumber.add(3));

System.out.println("4th largest number is: " + kthLargestNumber.add(4));

}

Run
Save Reset 12
```

Solution

This problem follows the **Top 'K' Elements** pattern and shares similarities with Kth Smallest number.

We can follow the same approach as discussed in the 'Kth Smallest number' problem. However, we will use a **Min Heap** (instead of a **Max Heap**) as we need to find the Kth largest number.

Code

riere is what our algorithm will look like:

```
Python3
                                   G C++
                                                    JS JS
👙 Java
       import java.util.*;
      class KthLargestNumberInStream {
   PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>((n1, n2) -> n1 - n2);
   final int k;
         public KthLargestNumberInStream(int[] nums, int k) {
            // add the numbers in the min heap
for (int i = 0; i < nums.length; i++)</pre>
               add(nums[i]);
         public int add(int num) {
            minHeap.add(num);
            // if heap has more than 'k' numbers, remove one number if \mbox{(minHeap.size() > this.k)}
              minHeap.poll();
            return minHeap.peek();
         public static void main(String[] args) {
   int[] input = new int[] { 3, 1, 5, 12, 2, 11 };
   KthLargestNumberInStream kthLargestNumber = new KthLargestNumberInStream(input, 4);
 Run
                                                                                                                             Save Reset []
```

Time complexity

The time complexity of the add() function will be O(logK) since we are inserting the new number in the heap.

Space complexity

The space complexity will be O(K) for storing numbers in the heap.

