Kth Largest Number in a Stream (medium)

We'll cover the following

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:

Java | Python3 | JS | C++

```
class KthLargestNumberInStream {
  constructor(nums, k) {
    // TODO: Write your code here
    this.k = k;
  }

  add(num) {
    // TODO: Write your code here
    return -1;
  }
};


kthLargestNumber = new KthLargestNumberInStream([3, 1, 5, 12, 2, 11], 4);
console.log(`4th largest number is: ${kthLargestNumber.add(6)}`)
console.log(`4th largest number is: ${kthLargestNumber.add(13)}`)
console.log(`4th largest number is: ${kthLargestNumber.add(4)}`)

```

RUN SAVE RESET

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

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Heap = require('./collections/heap'); //http://www.collectionsjs.com


class KthLargestNumberInStream {
  constructor(nums, k) {
    this.minHeap = new Heap([], null, ((a, b) => b - a));
    this.k = k;

    // add the numbers in the min heap
    nums.forEach((num) => {
      this.add(num);
    });
  }
```

```
14
15  add(num) {
16    // add the new number in the min heap
17    this.minHeap.push(num);
18
19    // if heap has more than 'k' numbers, remove one number
20    if (this.minHeap.length > this.k) {
21      this.minHeap.pop();
22    }
23
24    // return the 'Kth largest number
25    return this.minHeap.peek();
26  }
27 }
28
```

RUN SAVE RESET

Close

Output 4.134s

```
4th largest number is: 5
4th largest number is: 6
4th largest number is: 6
```

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.

Back
Frequency Sort (medium)

Next
'K' Closest Numbers (medium)

Mark as Completed

Report an Issue Ask a Question