

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++

```
1 class KthLargestNumberInStream:
2     def __init__(self, nums, k):
3         # TODO: Write your code here
4         self.k = k
5
6     def add(self, num):
7         # TODO: Write your code here
8         return -1
9
10
11 def main():
12
13     kthLargestNumber = KthLargestNumberInStream([3, 1, 5, 12, 2, 11], 4)
14     print("4th largest number is: " + str(kthLargestNumber.add(6)))
15     print("4th largest number is: " + str(kthLargestNumber.add(13)))
16     print("4th largest number is: " + str(kthLargestNumber.add(4)))
17
18
19 main()
20
21
```

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

```
1 from heapq import *
2
3
4 class KthLargestNumberInStream:
5     minHeap = []
6
7     def __init__(self, nums, k):
8         self.k = k
9         # add the numbers in the min heap
10        for num in nums:
11            self.add(num)
12
13    def add(self, num):
14        # add the new number in the min heap
15        heappush(self.minHeap, num)
16
17        # if heap has more than 'k' numbers, remove one number
18        if len(self.minHeap) > self.k:
19            heappop(self.minHeap)
20
21        # return the 'kth largest number
22        return self.minHeap[0]
23
24
25 def main():
26
27     kthLargestNumber = KthLargestNumberInStream([3, 1, 5, 12, 2, 11], 4)
28     print("4th largest number is: " + str(kthLargestNumber.add(6)))
29     print("4th largest number is: " + str(kthLargestNumber.add(13)))
30     print("4th largest number is: " + str(kthLargestNumber.add(4)))
31
```

Run

Save

Reset

Time complexity

The time complexity of the `add()` function will be $O(\log K)$ 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

Next

Frequency Sort (medium)

'K' Closest Numbers (medium)

Mark as Completed

Report an Issue

Ask a Question