Kth Smallest Number (easy)

We'll cover the following

Problem Statement

Given an unsorted array of numbers, find Kth smallest number in it.

Please note that it is the Kth smallest number in the sorted order, not the Kth distinct element.

Note: For a detailed discussion about different approaches to solve this problem, take a look at Kth Smallest Number.

Example 1:

```
Input: [1, 5, 12, 2, 11, 5], K = 3
Output: 5
Explanation: The 3rd smallest number is '5', as the first two smaller numbers are [1, 2].
```

Example 2:

```
Input: [1, 5, 12, 2, 11, 5], K = 4
Output: 5
Explanation: The 4th smallest number is '5', as the first three small numbers are [1, 2, 5].
```

Example 3:

```
Input: [5, 12, 11, -1, 12], K = 3
Output: 11
Explanation: The 3rd smallest number is '11', as the first two small numbers are [5, -1].
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```java
import java.util.*;

class KthSmallestNumber {

  public static int findKthSmallestNumber(int[] nums, int k) {
    // TODO: Write your code here
    return -1;
  }

  public static void main(String[] args) {
    int result = KthSmallestNumber.findKthSmallestNumber(new int[] { 1, 5, 12, 2, 11, 5 }, 3);
    System.out.println("Kth smallest number is: " + result);

    // since there are two 5s in the input array, our 3rd and 4th smallest numbers should be a '5'
    result = KthSmallestNumber.findKthSmallestNumber(new int[] { 1, 5, 12, 2, 11, 5 }, 4);
    System.out.println("Kth smallest number is: " + result);

    result = KthSmallestNumber.findKthSmallestNumber(new int[] { 5, 12, 11, -1, 12 }, 3);
    System.out.println("Kth smallest number is: " + result);
  }
}

```

RUN SAVE RESET

Close

Output 3.005s

```
Kth smallest number is: -1
Kth smallest number is: -1
Kth smallest number is: -1
```

Solution

This problem follows the Top 'K' Numbers pattern but has two differences:

1. Here we need to find the Kth `smallest` number, whereas in Top 'K' Numbers we were dealing with 'K' `largest` numbers.
2. In this problem, we need to find only one number (Kth smallest) compared to finding all 'K' largest numbers.

We can follow the same approach as discussed in the 'Top K Elements' problem. To handle the first difference mentioned above, we can use a max-heap instead of a min-heap. As we know, the root is the biggest element in the max heap. So, since we want to keep track of the 'K' smallest numbers, we can compare every number with the root while iterating through all numbers, and if it is smaller than the root, we'll take the root out and insert the smaller number.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```java
import java.util.*;

class KthSmallestNumber {

  public static int findKthSmallestNumber(int[] nums, int k) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>((n1, n2) -> n2 - n1);
    // put first k numbers in the max heap
    for (int i = 0; i < k; i++)
      maxHeap.add(nums[i]);

    // go through the remaining numbers of the array, if the number from the array is smaller than the
    // top (biggest) number of the heap, remove the top number from heap and add the number from array
    for (int i = k; i < nums.length; i++) {
      if (nums[i] < maxHeap.peek()) {
        maxHeap.poll();
        maxHeap.add(nums[i]);
      }
    }

    // the root of the heap has the Kth smallest number
    return maxHeap.peek();
  }

  public static void main(String[] args) {
    int result = KthSmallestNumber.findKthSmallestNumber(new int[] { 1, 5, 12, 2, 11, 5 }, 3);
    System.out.println("Kth smallest number is: " + result);

    // since there are two 5s in the input array, our 3rd and 4th smallest numbers should be a '5'
```

RUN | SAVE | RESET

Close

Output 1.515s

```
Kth smallest number is: 5
Kth smallest number is: 5
Kth smallest number is: 11
```

Time complexity

The time complexity of this algorithm is $O(K * logK + (N - K) * logK)$, which is asymptotically equal to $O(N * logK)$

Space complexity

The space complexity will be $O(K)$ because we need to store 'K' smallest numbers in the heap.

An Alternate Approach

Alternatively, we can use a **Min Heap** to find the Kth smallest number. We can insert all the numbers in the min-heap and then extract the top 'K' numbers from the heap to find the Kth smallest number. Initializing the min-heap with all numbers will take $O(N)$ and extracting 'K' numbers will take $O(KlogN)$. Overall, the time complexity of this algorithm will be $O(N + KlogN)$ and the space complexity will be $O(N)$.

Back
Top 'K' Numbers (easy)
Next
'K' Closest Points to the Origin (easy)
Mark as Completed
Report an Issue
Ask a Question

Create