

Kth Smallest Number (easy)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity
- An Alternate Approach

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++

```
1 import java.util.*;
2
3 class KthSmallestNumber {
4
5     public static int findKthSmallestNumber(int[] nums, int k) {
6         // TODO: Write your code here
7         return -1;
8     }
9
10    public static void main(String[] args) {
11        int result = KthSmallestNumber.findKthSmallestNumber(new int[] { 1, 5, 12, 2, 11, 5 }, 3);
12        System.out.println("Kth smallest number is: " + result);
13
14        // since there are two 5s in the input array, our 3rd and 4th smallest numbers should be a '5'
15        result = KthSmallestNumber.findKthSmallestNumber(new int[] { 1, 5, 12, 2, 11, 5 }, 4);
16        System.out.println("Kth smallest number is: " + result);
17
18        result = KthSmallestNumber.findKthSmallestNumber(new int[] { 5, 12, 11, -1, 12 }, 3);
19        System.out.println("Kth smallest number is: " + result);
20    }
21 }
22
```

Run

Save

Reset



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

```
1 import java.util.*;
2
3 class KthSmallestNumber {
4
5     public static int findKthSmallestNumber(int[] nums, int k) {
6         PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>((n1, n2) -> n2 - n1);
7         // put first k numbers in the max heap
8         for (int i = 0; i < k; i++)
9             maxHeap.add(nums[i]);
10
11         // go through the remaining numbers of the array, if the number from the array is smaller than the
12         // top (biggest) number of the heap, remove the top number from heap and add the number from array
13         for (int i = k; i < nums.length; i++) {
14             if (nums[i] < maxHeap.peek()) {
15                 maxHeap.poll();
16                 maxHeap.add(nums[i]);
17             }
18         }
19
20         // the root of the heap has the Kth smallest number
21         return maxHeap.peek();
22     }
23
24     public static void main(String[] args) {
25         int result = KthSmallestNumber.findKthSmallestNumber(new int[] { 1, 5, 12, 2, 11, 5 }, 3);
26         System.out.println("Kth smallest number is: " + result);
27
28         // since there are two 5s in the input array, our 3rd and 4th smallest numbers should be a '5'
```

Run Save Reset

Time complexity

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

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 $O(K \log N)$. Overall, the time complexity of this algorithm will be $O(N + K \log N)$ and the space complexity will be $O(N)$.

← Back

Next →

Top 'K' Numbers (easy)

'K' Closest Points to the Origin (easy)

✓ Mark as Completed

Report an Issue Ask a Question

