Sum of Elements (medium)

We'll cover the following

Problem Statement

Given an array, find the sum of all numbers between the K1'th and K2'th smallest elements of that array.

Example 1:

```
Input: [1, 3, 12, 5, 15, 11], and K1=3, K2=6
Output: 23
Explanation: The 3rd smallest number is 5 and 6th smallest number 15. The sum of numbers coming between 5 and 15 is 23 (11+12).
```

Example 2:

```
Input: [3, 5, 8, 7], and K1=1, K2=4
Output: 12
Explanation: The sum of the numbers between the 1st smallest number (3) and the 4th smallest number (8) is 12 (5+7).
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```java
import java.util.*;

class SumOfElements {

  public static int findSumOfElements(int[] nums, int k1, int k2) {
    // TODO: Write your code here
    return -1;
  }

  public static void main(String[] args) {
    int result = SumOfElements.findSumOfElements(new int[] { 1, 3, 12, 5, 15, 11 }, 3, 6);
    System.out.println("Sum of all numbers between k1 and k2 smallest numbers: " + result);

    result = SumOfElements.findSumOfElements(new int[] { 3, 5, 8, 7 }, 1, 4);
    System.out.println("Sum of all numbers between k1 and k2 smallest numbers: " + result);
  }
}

```

Run | Save | Reset

Solution

This problem follows the Top 'K' Numbers pattern, and shares similarities with Kth Smallest Number.

We can find the sum of all numbers coming between the K1'th and K2'th smallest numbers in the following steps:

1. First, insert all numbers in a min-heap.
2. Remove the first K1 smallest numbers from the min-heap

2. Remove the first `K1` smallest numbers from the min-heap.
3. Now take the next `K2-K1-1` numbers out of the heap and add them. This sum will be our required output.

Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```java
import java.util.*;

class SumOfElements {

  public static int findSumOfElements(int[] nums, int k1, int k2) {
    PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>((n1, n2) -> n1 - n2);
    // insert all numbers to the min heap
    for (int i = 0; i < nums.length; i++)
      minHeap.add(nums[i]);

    // remove k1 small numbers from the min heap
    for (int i = 0; i < k1; i++)
      minHeap.poll();

    int elementSum = 0;
    // sum next k2-k1-1 numbers
    for (int i = 0; i < k2 - k1 - 1; i++)
      elementSum += minHeap.poll();

    return elementSum;
  }

  public static void main(String[] args) {
    int result = SumOfElements.findSumOfElements(new int[] { 1, 3, 12, 5, 15, 11 }, 3, 6);
    System.out.println("Sum of all numbers between k1 and k2 smallest numbers: " + result);

    result = SumOfElements.findSumOfElements(new int[] { 3, 5, 8, 7 }, 1, 4);
    System.out.println("Sum of all numbers between k1 and k2 smallest numbers: " + result);
  }
}
```

Run Save Reset

Time complexity

Since we need to put all the numbers in a min-heap, the time complexity of the above algorithm will be $O(N * logN)$ where 'N' is the total input numbers.

Space complexity

The space complexity will be $O(N)$, as we need to store all the 'N' numbers in the heap.

Alternate Solution

We can iterate the array and use a max-heap to keep track of the top K2 numbers. We can, then, add the top `K2-K1-1` numbers in the max-heap to find the sum of all numbers coming between the K1'th and K2'th smallest numbers. Here is what the algorithm will look like:

Java Python3 C++ JS

```java
import java.util.*;

class SumOfElements {

  public static int findSumOfElements(int[] nums, int k1, int k2) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>((n1, n2) -> n2 - n1);
    // keep smallest k2 numbers in the max heap
    for (int i = 0; i < nums.length; i++) {
      if (i < k2 - 1) {
        maxHeap.add(nums[i]);
      } else if (nums[i] < maxHeap.peek()) {
        maxHeap.poll(); // as we are interested only in the smallest k2 numbers
        maxHeap.add(nums[i]);
      }
    }

    // get the sum of numbers between k1 and k2 indices
    // these numbers will be at the top of the max heap
    int elementSum = 0;
    for (int i = 0; i < k2 - k1 - 1; i++)
      elementSum += maxHeap.poll();

    return elementSum;
  }

  public static void main(String[] args) {
    int result = SumOfElements.findSumOfElements(new int[] { 1, 3, 12, 5, 15, 11 }, 3, 6);
    System.out.println("Sum of all numbers between k1 and k2 smallest numbers: " + result);
```

Time complexity

Since we need to put only the top K2 numbers in the max-heap at any time, the time complexity of the above algorithm will be $O(N * logK2)$.

Space complexity

The space complexity will be $O(K2)$, as we need to store the smallest 'K2' numbers in the heap.

Back
Maximum Distinct Elements (medium)
Next
Rearrange String (hard)
Mark as Completed
Report an Issue
Ask a Question