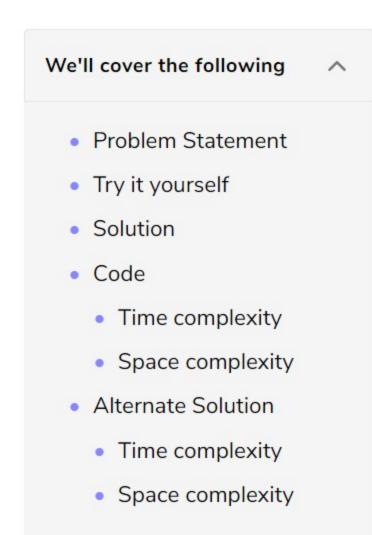


Q Search Course

Sum of Elements (medium)



Problem Statement

Input: [1, 3, 12, 5, 15, 11], and K1=3, K2=6

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

₿

Example 1:

```
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:

```
Python3
                                      @ C++
Java
                          JS JS
    def find_sum_of_elements(nums, k1, k2):
      # TODO: Write your code here
      return -1
    def main():
      print("Sum of all numbers between k1 and k2 smallest numbers: " +
            str(find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)))
      print("Sum of all numbers between k1 and k2 smallest numbers: " +
10
            str(find_sum_of_elements([3, 5, 8, 7], 1, 4)))
11
12
13
14 main()
15
Run
                                                                                                       Reset
                                                                                              Save
```

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.
- 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:

```
Python3
                          G C++
                                      JS JS
Java
 1 from heapq import *
    def find sum of elements(nums, k1, k2):
      minHeap = []
      # insert all numbers to the min heap
      for num in nums:
       heappush(minHeap, num)
      # remove k1 small numbers from the min heap
      for _ in range(k1):
11
       heappop(minHeap)
12
13
      elementSum = 0
14
      # sum next k2-k1-1 numbers
15
      for _ in range(k2 - k1 - 1):
       elementSum += heappop(minHeap)
17
      return elementSum
21
    def main():
22
      print("Sum of all numbers between k1 and k2 smallest numbers: " +
24
            str(find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)))
25
      print("Sum of all numbers between k1 and k2 smallest numbers: " +
            str(find_sum_of_elements([3, 5, 8, 7], 1, 4)))
27
29
30 main()
                                                                                                                 ::3
Run
                                                                                                        Reset
                                                                                              Save
```

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 * log N) where 'N' is the total input numbers.

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

Alternate Solution

Space complexity

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:

```
Python3
                            G C++
                                         JS JS
  Java
   1 from heapq import *
   4 def find_sum_of_elements(nums, k1, k2):
        maxHeap = []
        # keep smallest k2 numbers in the max heap
        for i in range(len(nums)):
          if i < k2 - 1:
            heappush(maxHeap, -nums[i])
          elif nums[i] < -maxHeap[0]:</pre>
   10
            heappop(maxHeap) # as we are interested only in the smallest k2 numbers
   11
            heappush(maxHeap, -nums[i])
   12
   13
        # get the sum of numbers between k1 and k2 indices
        # these numbers will be at the top of the max heap
        elementSum = 0
        for _ in range(k2 - k1 - 1):
  17
          elementSum += -heappop(maxHeap)
         return elementSum
  20
   21
  22
      def main():
   24
        print("Sum of all numbers between k1 and k2 smallest numbers: " +
              str(find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)))
   26
         print("Sum of all numbers between k1 and k2 smallest numbers: " +
              str(find_sum_of_elements([3, 5, 8, 7], 1, 4)))
  31 main()
                                                                                                 Save
                                                                                                          Reset
   Run
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 * log K2). Space complexity

← Back

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

```
Rearrange String (hard)
Maximum Distinct Elements (medium)
                                                                                                           ✓ Mark as Completed
                                                                                           Propert an Issue Ask a Question
```

Next ->