

Sum of Elements (medium)

We'll cover the following ^

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

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++

```
1 def find_sum_of_elements(nums, k1, k2):
2     # TODO: Write your code here
3     return 1
```



```

3     return -1
4
5
6 def main():
7
8     print("Sum of all numbers between k1 and k2 smallest numbers: " +
9           str(find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)))
10    print("Sum of all numbers between k1 and k2 smallest numbers: " +
11          str(find_sum_of_elements([3, 5, 8, 7], 1, 4)))
12
13
14 main()
15

```



Solution

This problem follows the Top 'K' Numbers

(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5728885882748928/>) pattern, and shares similarities with Kth Smallest Number

(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5696381570252800/>).

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:

Java

Python3

C++

JS

```

1  from heapq import *
2
3
4  def find_sum_of_elements(nums, k1, k2):
5      minHeap = []
6      # insert all numbers to the min heap
7      for num in nums:
8          heappush(minHeap, num)
9
10     # remove k1 small numbers from the min heap
11     for _ in range(k1):
12         heappop(minHeap)
13
14     elementSum = 0
15     # sum next k2-k1-1 numbers
16     for _ in range(k2 - k1 - 1):
17         elementSum += heappop(minHeap)
18

```



```

19     return elementSum
20
21
22 def main():
23
24     print("Sum of all numbers between k1 and k2 smallest numbers: " +
25           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: " +
27           str(find_sum_of_elements([3, 5, 8, 7], 1, 4)))
28

```



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.

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

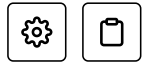
```

1  from heapq import *
2
3
4  def find_sum_of_elements(nums, k1, k2):
5      maxHeap = []
6      # keep smallest k2 numbers in the max heap
7      for i in range(len(nums)):
8          if i < k2 - 1:
9              heappush(maxHeap, -nums[i])
10         elif nums[i] < -maxHeap[0]:
11             heappop(maxHeap) # as we are interested only in the smallest k2 numbers
12             heappush(maxHeap, -nums[i])
13
14     # get the sum of numbers between k1 and k2 indices
15     # these numbers will be at the top of the max heap
16     elementSum = 0
17     for _ in range(k2 - k1 - 1):
18         elementSum += -heappop(maxHeap)
19
20     return elementSum
21
22
23 def main():
24
25     print("Sum of all numbers between k1 and k2 smallest numbers: " +
26           str(find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)))
27     print("Sum of all numbers between k1 and k2 smallest numbers: " +
28           str(find sum of elements([3, 5, 8, 7], 1, 4)))

```



Time complexity



Since we need to put only the top K^2 numbers in the max-heap at any time, the time complexity of the above algorithm will be $O(N * \log K^2)$.

Space complexity

The space complexity will be $O(K^2)$, as we need to store the smallest ' K^2 ' numbers in the heap.

[← Back](#)[Next →](#)

Maximum Distinct Elements (medium)

Rearrange String (hard)

☒ **Mark as Completed**



Report an
Issue



Ask a Question

(https://discuss.educative.io/tag/sum-of-elements-medium__pattern-top-k-elements__grokking-the-coding-interview-patterns-for-coding-questions)