

Grokking the Coding Interview: Patterns for Coding Questions

79% completed

 Search Course

Introduction

Pattern: Sliding Window

Pattern: Two Pointers

Pattern: Fast & Slow pointers

Pattern: Merge Intervals

Pattern: Cyclic Sort

Pattern: In-place Reversal of a LinkedList

Pattern: Tree Breadth First Search

Pattern: Tree Depth First Search

Pattern: Two Heaps

Pattern: Subsets

Pattern: Modified Binary Search

Pattern: Bitwise XOR

Pattern: Top 'K' Elements

-  Introduction
-  Top 'K' Numbers (easy)
-  Kth Smallest Number (easy)
-  'K' Closest Points to the Origin (easy)
-  Connect Ropes (easy)
-  Top 'K' Frequent Numbers (medium)
-  Frequency Sort (medium)
-  Kth Largest Number in a Stream (medium)
-  'K' Closest Numbers (medium)
-  Maximum Distinct Elements (medium)
-  Sum of Elements (medium)
-  Rearrange String (hard)
-  Problem Challenge 1
-  Solution Review: Problem

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 const find_sum_of_elements = function(nums, k1, k2) {
2   // TODO: Write your code here
3   return -1;
4 };
5
6
7 console.log(`Sum of all numbers between k1 and k2 smallest numbers: ${find_sum_of_elements([1, 3, 12, 5,
8, console.log(`Sum of all numbers between k1 and k2 smallest numbers: ${find_sum_of_elements([3, 5, 8, 7],
9

```

RUN
SAVE
RESET
COPY

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:

 Java	 Python3	 C++	 JS
--	---	---	--

```

1 const Heap = require('../collections/heap'); //http://www.collectionsjs.com
2
3
4 function find_sum_of_elements(nums, k1, k2) {
5   // insert all numbers in a min heap
6   const minHeap = new Heap(nums, null, ((a, b) => b - a));
7
8   // remove k1 small numbers from the min heap
9   for (i = 0; i < k1; i++) {
10     minHeap.pop();
11   }

```

- Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: K-way merge ^

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

```

12 let elementSum = 0;
13 // sum next k2-k1-1 numbers
14 for (i = 0; i < k2 - k1 - 1; i++) {
15   elementSum += minHeap.pop();
16 }
17
18
19 return elementSum;
20 }
21
22
23 console.log(`Sum of all numbers between k1 and k2 smallest numbers: ${
24   find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)}`);
25 console.log(`Sum of all numbers between k1 and k2 smallest numbers: ${
26   find_sum_of_elements([3, 5, 8, 7], 1, 4)}`);
27

```

RUN

SAVE

RESET

Close

Output

4.433s

```
Sum of all numbers between k1 and k2 smallest numbers: 23
Sum of all numbers between k1 and k2 smallest numbers: 12
```

Pattern : 0/1 Knapsack ^ (Dynamic Programming)

- Introduction
- 0/1 Knapsack (medium)
- Equal Subset Sum Partition (medium)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph) ^

- Introduction
- Topological Sort (medium)
- Tasks Scheduling (medium)
- Tasks Scheduling Order (medium)
- All Tasks Scheduling Orders (hard)
- Alien Dictionary (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Miscellaneous ^

- Kth Smallest Number (hard)



Explore



Tracks



My Courses



Edpresso



Refer a Friend

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 JS

1 const Heap = require('../collections/heap'); //http://www.collectionsjs.com
2
3 function find_sum_of_elements(nums, k1, k2) {
4   const maxHeap = new Heap();
5   // keep smallest k2 numbers in the max heap
6   for (i = 0; i < nums.length; i++) {
7     if (i < k2 - 1) {
8       maxHeap.push(nums[i]);
9     } else if (nums[i] < maxHeap.peek()) {
10      // as we are interested only in the smallest k2 numbers
11      maxHeap.pop();
12      maxHeap.push(nums[i]);
13    }
14  }
15
16 // get the sum of numbers between k1 and k2 indices
17 // these numbers will be at the top of the max heap
18 let elementSum = 0;
19 for (i = 0; i < k2 - k1 - 1; i++) {
20   elementSum += maxHeap.pop();
21 }
22
23 return elementSum;
24 }
25
26
27 console.log(`Sum of all numbers between k1 and k2 smallest numbers: ${
28   find_sum_of_elements([1, 3, 12, 5, 15, 11], 3, 6)}`);
29

```

RUN

SAVE

RESET

Close

Output

7.895s

```
Sum of all numbers between k1 and k2 smallest numbers: 23
Sum of all numbers between k1 and k2 smallest numbers: 12
```

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

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

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#)

[Back](#)

Maximum Distinct Elements (medium)

[Next](#)

Rearrange String (hard)

[Mark as Completed](#)

Report an Issue Ask a Question

+

Create