

Grokking the Coding Interview: Patterns for Coding Questions

86% completed



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 ▾

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

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

We'll cover the following ▴

- K Pairs with Largest Sums (Hard)
- Solution
- Code
 - Time complexity
 - Space complexity

K Pairs with Largest Sums (Hard)

Given two sorted arrays in descending order, find 'K' pairs with the largest sum where each pair consists of numbers from both the arrays.

Example 1:

```
Input: L1=[9, 8, 2], L2=[6, 3, 1], K=3
Output: [9, 3], [9, 6], [8, 6]
Explanation: These 3 pairs have the largest sum. No other pair has a sum larger than any of the se.
```

Example 2:

```
Input: L1=[5, 2, 1], L2=[2, -1], K=3
Output: [5, 2], [5, -1], [2, 2]
```

Solution

This problem follows the **K-way merge** pattern and we can follow a similar approach as discussed in [Merge K Sorted Lists](#).

We can go through all the numbers of the two input arrays to create pairs and initially insert them all in the heap until we have 'K' pairs in **Min Heap**. After that, if a pair is bigger than the top (smallest) pair in the heap, we can remove the smallest pair and insert this pair in the heap.

We can optimize our algorithms in two ways:

1. Instead of iterating over all the numbers of both arrays, we can iterate only the first 'K' numbers from both arrays. Since the arrays are sorted in descending order, the pairs with the maximum sum will be constituted by the first 'K' numbers from both the arrays.
2. As soon as we encounter a pair with a sum that is smaller than the smallest (top) element of the heap, we don't need to process the next elements of the array. Since the arrays are sorted in descending order, we won't be able to find a pair with a higher sum moving forward.

Code

Here is what our algorithm will look like:

```
1  const Heap = require('./collections/heap'); //http://www.collectionsjs.com
2
3
4  function find_k_largest_pairs(nums1, nums2, k) {
5      const minHeap = new Heap([], null, ((a, b) => b[0] - a[0]));
6
7      for (i = 0; i < Math.min(k, nums1.length); i++) {
8          for (j = 0; j < Math.min(k, nums2.length); j++) {
9              if (minHeap.length < k) {
10                 minHeap.push([nums1[i] + nums2[j], i, j]);
11             } else {
12                 // if the sum of the two numbers from the two arrays is smaller than the smallest(top)
13                 // element of the heap, we can 'break' here. Since the arrays are sorted in the
14                 // descending order, we'll not be able to find a pair with a higher sum moving forward
15                 if (nums1[i] + nums2[j] < minHeap.peek()[0]) {
16                     break;
17                 } else { // we have a pair with a larger sum, remove top and insert this pair in the heap
18                     minHeap.pop();
19                     minHeap.push([nums1[i] + nums2[j], i, j]);
20                 }
21             }
22         }
23     }
24
25     const result = [];
26     minHeap.forEach((a) => {
27         result.push([nums1[a[1]], nums2[a[2]]]);
28     });
29 }
```

RUN

SAVE

RESET



Close

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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)

Conclusions

Where to Go from Here

Mark Course as Completed

Output5.997s

Pairs with largest sum are: [9, 3] [9, 6] [8, 6]

Time complexity ⚙

Since, at most, we'll be going through all the elements of both arrays and we will add/remove one element in the heap in each step, the time complexity of the above algorithm will be $O(N * M * \log K)$ where 'N' and 'M' are the total number of elements in both arrays, respectively.

If we assume that both arrays have at least 'K' elements then the time complexity can be simplified to $O(K^2 \log K)$, because we are not iterating more than 'K' elements in both arrays.

Space complexity ⚙

The space complexity will be $O(K)$ because, at any time, our **Min Heap** will be storing 'K' largest pairs.

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

← Back

Next →

Problem Challenge 1Introduction

✔

Mark as Completed

🗨

Report an Issue

🗨

Ask a Question