

Grokking the Coding Interview: Patterns for Coding Questions

43% completed

(easy)

- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

- Introduction
- Order-Statistic Binary Search

Top 'K' Numbers (easy)

We'll cover the following

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

Problem Statement

Given an unsorted array of numbers, find the 'K' largest numbers in it.

Note: For a detailed discussion about different approaches to solve this problem, take a look at [Kth Smallest Number](#).

Example 1:

```
Input: [3, 1, 5, 12, 2, 11], K = 3
Output: [5, 12, 11]
```

Example 2:

```
Input: [5, 12, 11, -1, 12], K = 3
Output: [12, 11, 12]
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 const find_k_largest_numbers = function(nums, k) {
2   result = []
3   // TODO: Write your code here
4   return result;
5 };
6
7
8 console.log('Here are the top K numbers: ${find_k_largest_numbers([3, 1, 5, 12, 2, 11], 3)}')
9 console.log('Here are the top K numbers: ${find_k_largest_numbers([5, 12, 11, -1, 12], 3)}')
10
```

RUNSAVERESET↺

Solution

A brute force solution could be to sort the array and return the largest K numbers. The time complexity of such an algorithm will be $O(N * \log N)$ as we need to use a sorting algorithm like [Timsort](#) if we use Java's `Collection.sort()`. Can we do better than that?

The best data structure that comes to mind to keep track of top 'K' elements is [Heap](#). Let's see if we can use a heap to find a better algorithm.

If we iterate through the array one element at a time and keep 'K' largest numbers in a heap such that each time we find a larger number than the smallest number in the heap, we do two things:

1. Take out the smallest number from the heap, and
2. Insert the larger number into the heap.

This will ensure that we always have 'K' largest numbers in the heap. The most efficient way to repeatedly find the smallest number among a set of numbers will be to use a min-heap. As we know, we can find the smallest number in a min-heap in constant time $O(1)$, since the smallest number is always at the root of the heap. Extracting the smallest number from a min-heap will take $O(\log N)$ (if the heap has 'N' elements) as the heap needs to readjust after the removal of an element.

Let's take Example-1 to go through each step of our algorithm:

Given array: [3, 1, 5, 12, 2, 11], and K=3

1. First, let's insert 'K' elements in the min-heap.
2. After the insertion, the heap will have three numbers [3, 1, 5] with '1' being the root as it is the smallest element.
3. We'll iterate through the remaining numbers and perform the above-mentioned two steps if we find a

- Order Agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

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 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

Pattern : 0/1 Knapsack (Dynamic Programming)

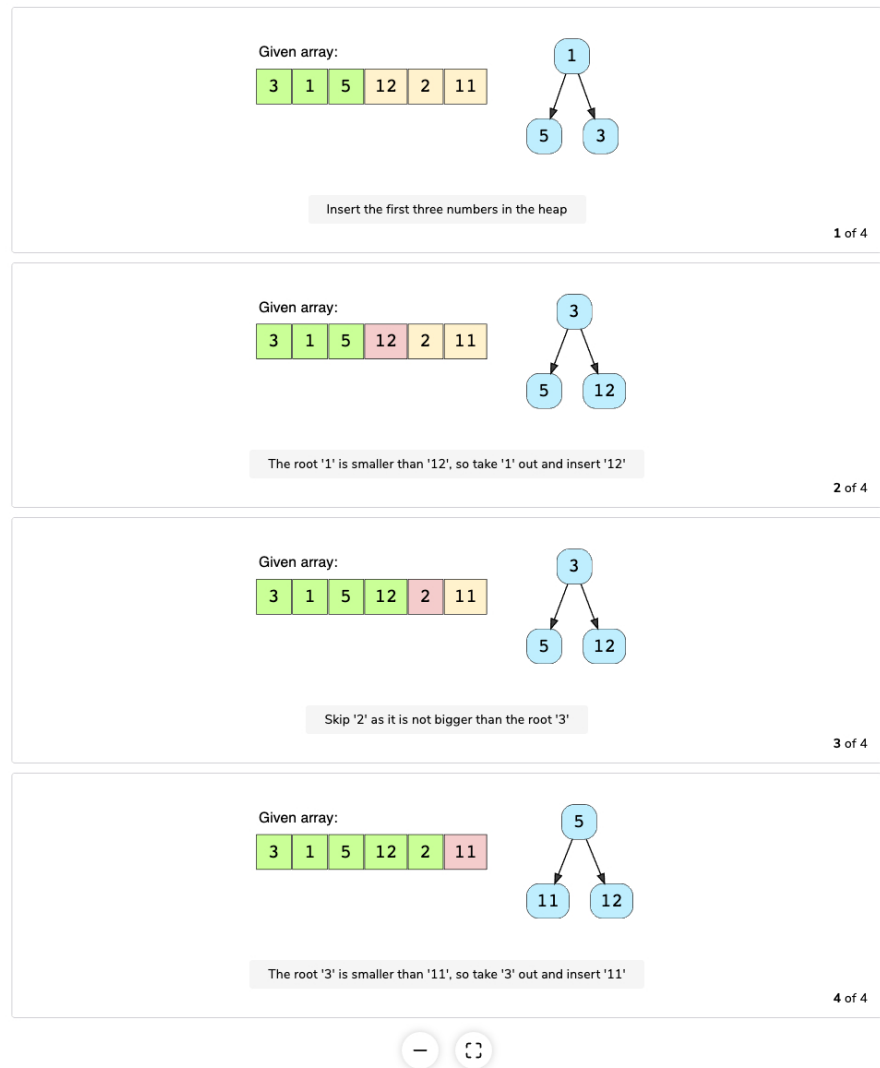
number larger than the root of the heap.

- The 4th number is '12' which is larger than the root (which is '1'), so let's take out '1' and insert '12'. Now the heap will have [3, 5, 12] with '3' being the root as it is the smallest element.
- The 5th number is '2' which is not bigger than the root of the heap ('3'), so we can skip this as we already have top three numbers in the heap.
- The last number is '11' which is bigger than the root (which is '3'), so let's take out '3' and insert '11'.

Finally, the heap has the largest three numbers: [5, 12, 11]

As discussed above, it will take us $O(\log K)$ to extract the minimum number from the min-heap. So the overall time complexity of our algorithm will be $O(K * \log K + (N - K) * \log K)$ since, first, we insert 'K' numbers in the heap and then iterate through the remaining numbers and at every step, in the worst case, we need to extract the minimum number and insert a new number in the heap. This algorithm is better than $O(N * \log N)$.

Here is the visual representation of our algorithm:



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_numbers(nums, k) {
5      const minHeap = new Heap([], null, ((a, b) => b - a));
6      // put first 'K' numbers in the min heap
7      for (i = 0; i < k; i++) {
8          minHeap.push(nums[i]);
9      }
10
11     // go through the remaining numbers of the array, if the number from the array is bigger than the
12     // top(i.e., smallest) number of the min-heap, remove the top number from heap and add the number from
13     for (i = k; i < nums.length; i++) {
14         if (nums[i] > minHeap.peek()) {
15             minHeap.pop();
16             minHeap.push(nums[i]);
17         }
18     }
19 }

```

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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)

Conclusions

Where to Go from Here

Mark Course as Completed

19

20

21

22

23

24

25

26

// the heap has the top 'K' numbers, return them in a list

return minHeap.toArray();

}

console.log(`Here are the top K numbers: \${find_k_largest_numbers([3, 1, 5, 12, 2, 11], 3)}`);

console.log(`Here are the top K numbers: \${find_k_largest_numbers([5, 12, 11, -1, 12], 3)}`);

RUN

SAVE

RESET

Close

Output

5.252s

Here are the top K numbers: 5,12,11

Here are the top K numbers: 11,12,12

Time complexity

As discussed above, the time complexity of this algorithm is $O(K * \log K + (N - K) * \log K)$, which is asymptotically equal to $O(N * \log K)$

Space complexity

The space complexity will be $O(K)$ since we need to store the top 'K' 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](#)

MARK AS COMPLETED

Back

Next

Introduction

Kth Smallest Number (easy)

Report an Issue Ask a Question