

Grokking the Coding Interview: Patterns for Coding Questions

84% 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

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

Merge K Sorted Lists (medium)

We'll cover the following

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

Problem Statement

Given an array of 'K' sorted LinkedLists, merge them into one sorted list.

Example 1:

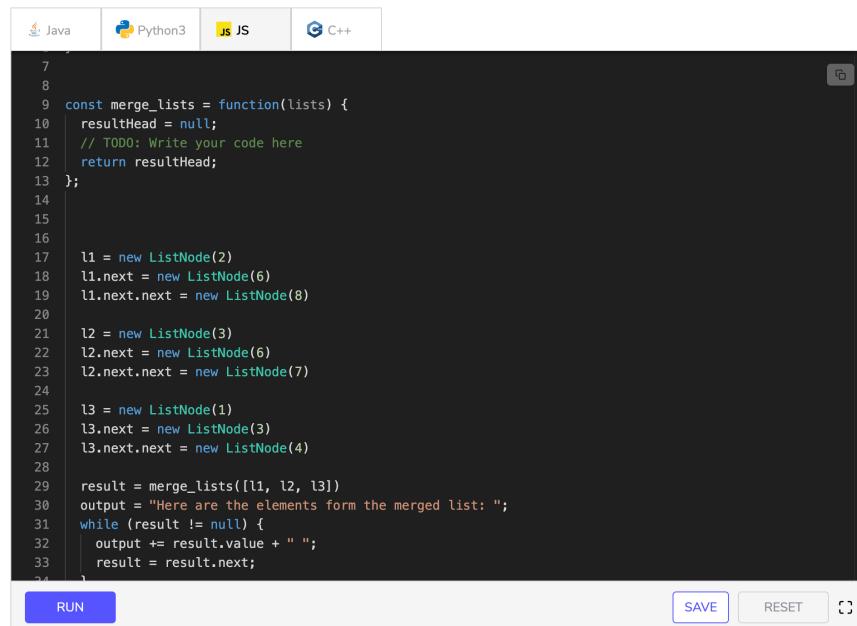
```
Input: L1=[2, 6, 8], L2=[3, 6, 7], L3=[1, 3, 4]
Output: [1, 2, 3, 3, 4, 6, 6, 7, 8]
```

Example 2:

```
Input: L1=[5, 8, 9], L2=[1, 7]
Output: [1, 5, 7, 8, 9]
```

Try it yourself

Try solving this question here:



```

7
8
9 const merge_lists = function(lists) {
10   resultHead = null;
11   // TODO: Write your code here
12   return resultHead;
13 };
14
15
16
17 l1 = new ListNode(2)
18 l1.next = new ListNode(6)
19 l1.next.next = new ListNode(8)
20
21 l2 = new ListNode(3)
22 l2.next = new ListNode(6)
23 l2.next.next = new ListNode(7)
24
25 l3 = new ListNode(1)
26 l3.next = new ListNode(3)
27 l3.next.next = new ListNode(4)
28
29 result = merge_lists([l1, l2, l3])
30 output = "Here are the elements form the merged list: "
31 while (result != null) {
32   output += result.value + " ";
33   result = result.next;
34 }

```



Solution

A brute force solution could be to add all elements of the given 'K' lists to one list and sort it. If there are a total of 'N' elements in all the input lists, then the brute force solution will have a time complexity of $O(N * \log N)$ as we will need to sort the merged list. Can we do better than this? How can we utilize the fact that the input lists are individually sorted?

If we have to find the smallest element of all the input lists, we have to compare only the smallest (i.e. the first) element of all the lists. Once we have the smallest element, we can put it in the merged list. Following a similar pattern, we can then find the next smallest element of all the lists to add it to the merged list.

The best data structure that comes to mind to find the smallest number among a set of 'K' numbers is a [Heap](#). Let's see how we can use a heap to find a better algorithm.

1. We can insert the first element of each array in a [Min Heap](#).
2. After this, we can take out the smallest (top) element from the heap and add it to the merged list.
3. After removing the smallest element from the heap, we can insert the next element of the same list into the heap.
4. We can repeat steps 2 and 3 to populate the merged list in sorted order.

- 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

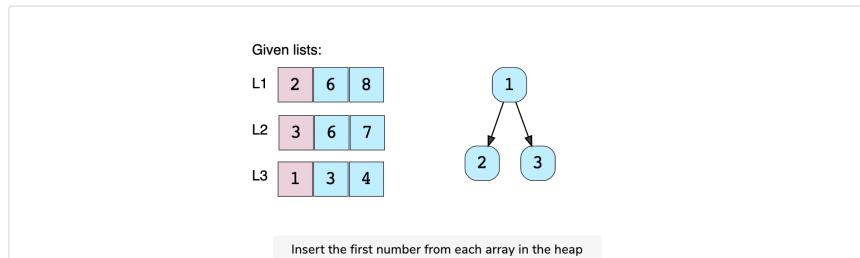
- Where to Go from Here

Mark Course as Completed

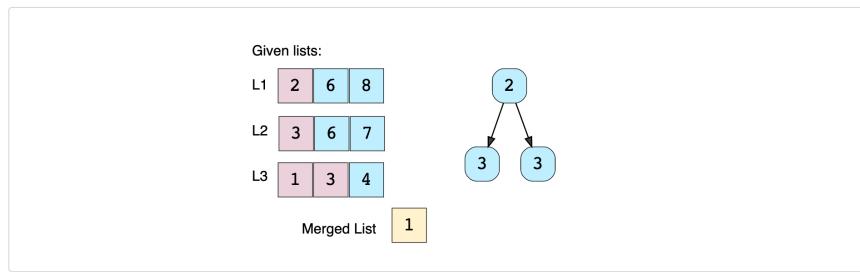
Let's take the Example-1 mentioned above to go through each step of our algorithm:

Given lists: L1=[2, 6, 8], L2=[3, 6, 7], L3=[1, 3, 4]

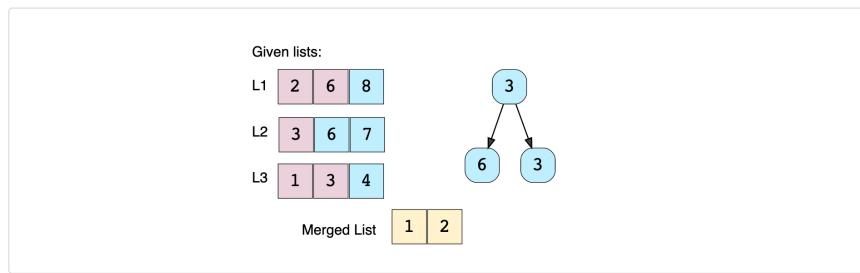
1. After inserting the 1st element of each list, the heap will have the following elements:



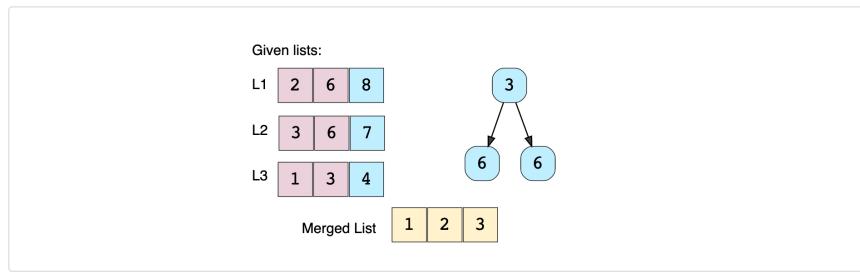
2. We'll take the top number from the heap, insert it into the merged list and add the next number in the heap.



3. Again, we'll take the top element of the heap, insert it into the merged list and add the next number to the heap.



4. Repeating the above step, take the top element of the heap, insert it into the merged list and add the next number to the heap. As there are two 3s in the heap, we can pick anyone but we need to take the next element from the corresponding list to insert in the heap.



We'll repeat the above step to populate our merged array.

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 class ListNode {
4   constructor(value, next = null) {
5     this.value = value;
6     this.next = next;
7   }
8 }
9
10 function merge_lists(lists) {
11   const minHeap = new Heap([], null, ((a, b) => b.value - a.value));
12
13   // put the root of each list in the min heap
14   lists.forEach((a) => {
15     if (a != null) {
16       minHeap.push(a);
17     }
18   });
19
20   let mergedList = [];
21
22   while (minHeap.size() > 0) {
23     const currentMin = minHeap.pop();
24     mergedList.push(currentMin.value);
25
26     if (currentMin.next != null) {
27       minHeap.push(currentMin.next);
28     }
29   }
30
31   return mergedList;
32 }

```

```

15     if (a !== null) {
16         minHeap.push(a);
17     }
18 });
19
20 // take the smallest(top) element form the min-heap and add it to the result
21 // if the top element has a next element add it to the heap
22 let resultHead = null,
23     resultTail = null;
24 while (minHeap.length > 0) {
25     const node = minHeap.pop();
26     if (resultHead === null) {
27         resultHead = resultTail = node;
28     } else {

```

RUN

SAVE

RESET



Close

Output

4.304s

```
Here are the elements from the merged list: 1 2 3 3 4 6 6 7 8
```

Time complexity

Since we'll be going through all the elements of all arrays and will be removing/adding one element to the heap in each step, the time complexity of the above algorithm will be $O(N * \log K)$, where 'N' is the total number of elements in all the 'K' input arrays.

Space complexity

The space complexity will be $O(K)$ because, at any time, our min-heap will be storing one number from all the 'K' input arrays.

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

Back

Next

Introduction

Kth Smallest Number in M Sorted List...

Mark as Completed

Report an Issue **Ask a Question**



Explore



Tracks



My Courses



Edpresso



Refer a Friend



Create