Solution Review: Problem Challenge 1

We'll cover the following

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:

Java | Python3 | C++ | JS

```java
import java.util.*;

class LargestPairs {

  public static List<int[]> findKLargestPairs(int[] nums1, int[] nums2, int k) {
    PriorityQueue<int[]> minHeap = new PriorityQueue<int[]>((p1, p2) -> (p1[0] + p1[1]) - (p2[0] + p2[1])
    for (int i = 0; i < nums1.length && i < k; i++) {
      for (int j = 0; j < nums2.length && j < k; j++) {
        if (minHeap.size() < k) {
          minHeap.add(new int[] { nums1[i], nums2[j] });
        } else {
          // if the sum of the two numbers from the two arrays is smaller than the smallest (top) element
          // the heap, we can 'break' here. Since the arrays are sorted in the descending order, we'll no
          // able to find a pair with a higher sum moving forward.
          if (nums1[i] + nums2[j] < minHeap.peek()[0] + minHeap.peek()[1]) {
            break;
          } else { // else: we have a pair with a larger sum, remove top and insert this pair in the heap
            minHeap.poll();
            minHeap.add(new int[] { nums1[i], nums2[j] });
```

```
            }
          }
        }
      }
      return new ArrayList<>(minHeap);
    }

    public static void main(String[] args) {
      int[] l1 = new int[] { 9, 8, 2 };
```

Run Save Reset

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 $O(N * M * logK)$ 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 $O(K^2logK)$, 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.

Back

Problem Challenge 1

Next

Introduction

Mark as Completed

Report an Issue Ask a Question