CS 180 Discussion 1A

Heap related questions

- Merge k Sorted Lists [leetcode 23]
- Find Median from Data Stream [leetcode 295]
- Find K Pairs with Smallest Sums [leetcode 373]
- The Skyline Problem [leetcode 218]

Merge k Sorted Lists

Merge *k* sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Example:

```
Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6
```

Merge k Sorted Lists

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
```

```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        PriorityQueue<ListNode> queue = new
PriorityQueue<>((a, b)->(a.val - b.val));
        for(ListNode head: lists) {
            if(head != null) {
                queue.offer(head);
            }
        }
        ListNode fake = new ListNode(0);
        ListNode tail = fake;
        while(!queue.isEmpty()) {
            ListNode cur = queue.poll();
            tail.next = cur;
            tail = tail.next;
            if(cur.next != null) {
                queue.add(cur.next);
            }
        }
        return fake.next;
    }
}
```

Find Median from Data Stream

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

For example,
`[2,3,4]`, the median is `3`

`[2,3]`, the median is `(2 + 3) / 2 = 2.5`

Design a data structure that supports the following two operations:

- void addNum(int num) - Add an integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

Find Median from Data Stream

```
class MedianFinder {
    private PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b)->(b - a));
    private PriorityQueue<Integer> minHeap = new PriorityQueue<>();
    public void addNum(int num) {
        maxHeap.offer(num);
        minHeap.offer(maxHeap.poll());
        if(maxHeap.size() < minHeap.size()) {
            maxHeap.offer(minHeap.poll());
        }
    }
    public double findMedian() {
        if(maxHeap.size() == minHeap.size()) {
            return ((double)maxHeap.peek() + minHeap.peek()) / 2.0;
        } else {
            return (double)maxHeap.peek();
        }
    }
}
```

Find K Pairs with Smallest Sums

You are given two integer arrays **nums1** and **nums2** sorted in ascending order and an integer **k**.

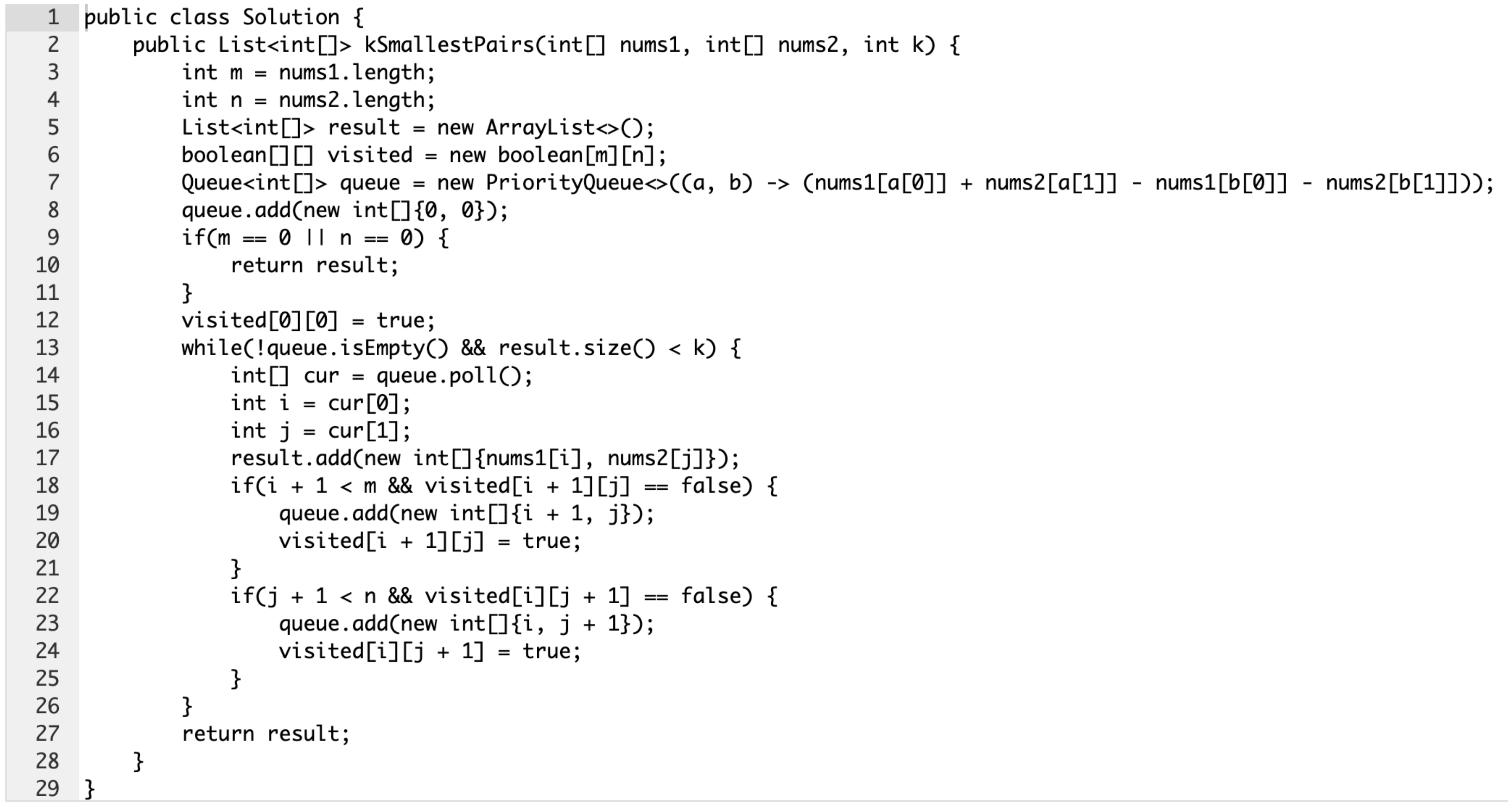
Define a pair **(u,v)** which consists of one element from the first array and one element from the second array.

Find the k pairs **$(u_1,v_1),(u_2,v_2)$ ...(u_k,v_k)** with the smallest sums.

Example 1:

```
Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
Output: [[1,2],[1,4],[1,6]]
Explanation: The first 3 pairs are returned from the sequence:
             [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
Example 2:
Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
Output: [1,1],[1,1]
Explanation: The first 2 pairs are returned from the sequence:
             [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
```

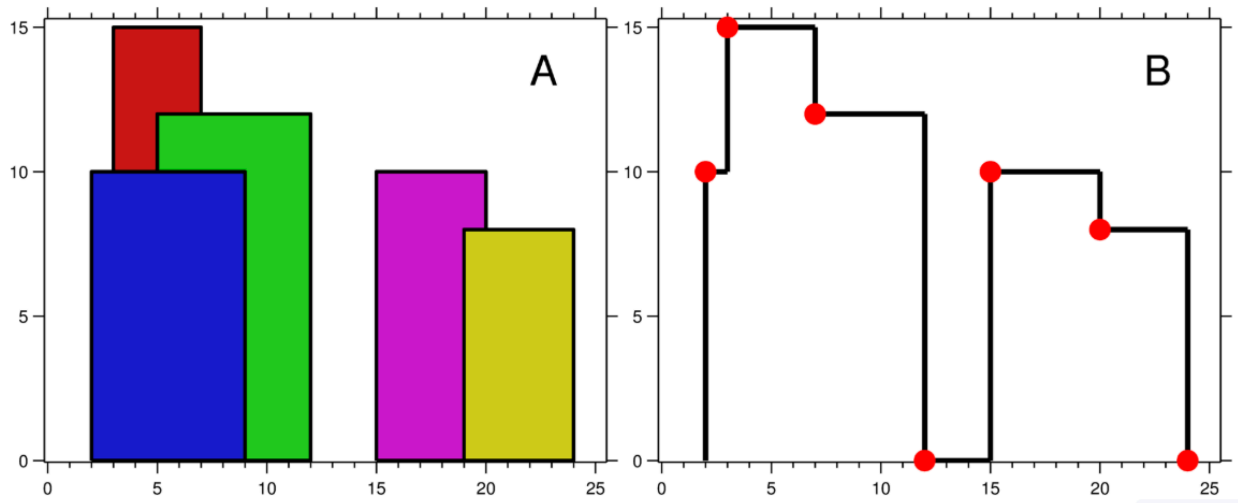
Find K Pairs with Smallest Sums



```java
public class Solution {
    public List<int[]> kSmallestPairs(int[] nums1, int[] nums2, int k) {
        int m = nums1.length;
        int n = nums2.length;
        List<int[]> result = new ArrayList<>();
        boolean[][] visited = new boolean[m][n];
        Queue<int[]> queue = new PriorityQueue<>((a, b) -> (nums1[a[0]] + nums2[a[1]] - nums1[b[0]] - nums2[b[1]]));
        queue.add(new int[]{0, 0});
        if(m == 0 || n == 0) {
            return result;
        }
        visited[0][0] = true;
        while(!queue.isEmpty() && result.size() < k) {
            int[] cur = queue.poll();
            int i = cur[0];
            int j = cur[1];
            result.add(new int[]{nums1[i], nums2[j]});
            if(i + 1 < m && visited[i + 1][j] == false) {
                queue.add(new int[]{i + 1, j});
                visited[i + 1][j] = true;
            }
            if(j + 1 < n && visited[i][j + 1] == false) {
                queue.add(new int[]{i, j + 1});
                visited[i][j + 1] = true;
            }
        }
        return result;
    }
}
```

The Skyline Problem

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Now suppose you are **given the locations and height of all the buildings** as shown on a cityscape photo (Figure A), write a program to **output the skyline** formed by these buildings collectively (Figure B).



The geometric information of each building is represented by a triplet of integers `[Li, Ri, Hi]`, where `Li` and `Ri` are the x coordinates of the left and right edge of the ith building, respectively, and `Hi` is its height. It is guaranteed that `0 ≤ Li, Ri ≤ INT_MAX`, `0 < Hi ≤ INT_MAX`, and `Ri - Li > 0`. You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

For instance, the dimensions of all buildings in Figure A are recorded as: `[ [2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8] ]` .

The Skyline Problem

```
class Solution {
    public List<int[]> getSkyline(int[][] buildings) {
        List<int[]> result = new ArrayList<>();
        List<int[]> height = new ArrayList<>();
        for(int[] building: buildings) {
            height.add(new int[]{building[0], -building[2]});
            height.add(new int[]{building[1], building[2]});
        }
        Collections.sort(height, (a, b) -> (a[0] == b[0] ? a[1] - b[1]: a[0] - b[0]));
        Queue<Integer> queue = new PriorityQueue<>((a, b) -> (b - a));
        queue.offer(0);
        int pre = 0;
        for(int[] h: height) {
            if(h[1] < 0) {
                queue.offer(-h[1]);
            } else {
                queue.remove(h[1]);
            }
            int cur = queue.peek();
            if(pre != cur) {
                result.add(new int[]{h[0], cur});
                pre = cur;
            }
        }
        return result;
    }
}
```