LeetCode 23

https://leetcode.com/problems/merge-k-sorted-lists/description/

Yifeng Zeng

Description

23. Merge k Sorted Lists

Idea Report

We want to merge k already sorted lists ListNode[] into one single linked list. We need to find the smallest ListNode in these k lists one at a time, add it to the output. Also we need to remove the smallest from the input and then again find the smallest ListNode from the updated input. The primitive idea is to loop the k nodes and find the smallest one, the time complexity is O(nk*k) , where n is the average number of nodes for each linked list because for each node we need to loop k nodes to find the smallest.

To speed up, we are actually looking for the smallest node from k nodes, then we can put the k nodes in a container and pick the smallest one from this container. The minHeap can find the smallest element from k elements so we can use a minHeap instead of looping through the k nodes. Each time we poll out the smallest node, add it to output, and if it's .next node is not null, we add .next node back into the minHeap. This way the time complexity is O(nklogk).

Code

```
public class Solution {
  public ListNode mergeKLists(ListNode[] lists) {
      if (lists == null || lists.length == 0) {
          return null;
      }

      Queue<ListNode> pq = new PriorityQueue<>((a, b) -> (a.val - b.val));
      for (ListNode list : lists) {
          if (list != null) {
              pq.offer(list);
          }
      }
```

```
        ListNode dummy = new ListNode(0);
        ListNode head = dummy;
        while (!pq.isEmpty()) {
            ListNode cur = pq.poll();
            head.next = cur;
            head = head.next;
            if (cur.next != null) {
                pq.offer(cur.next);
            }
        }

        return dummy.next;
    }
}
```

Another faster way is to merge these k lists pair by pair. Suppose if we have 4 lists A,B,C,D, we select 2 lists A,B to merge them together into E, and we merge the next 2 lists C,D to merge them together into F. And then we merge E and F to the final result. In this way, we reduce the input size by half, to the time complexity is O(nlogk), where n is the average length of each list because to merge 2 lists use O(n) time, and we need to merge logk times.

Code

```
class Solution {

    public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) {
            return null;
        }

        return mergeKLists(lists, 0, lists.length - 1);
    }

    private ListNode mergeKLists(ListNode[] lists, int start, int end) {
        if (start == end) {
            return lists[start];
        }

        int mid = (end - start) / 2 + start;
        ListNode left = mergeKLists(lists, start, mid);
        ListNode right = mergeKLists(lists, mid + 1, end);
        return merge(left, right);
    }

    private ListNode merge(ListNode left, ListNode right) {
        ListNode dummy = new ListNode(0);
```

```
        ListNode head = dummy;
        while (left != null && right != null) {
            if (left.val < right.val) {
                head.next = left;
                left = left.next;
            } else {
                head.next = right;
                right = right.next;
            }
            head = head.next;
        }
        if (left != null) {
            head.next = left;
        }
        if (right != null) {
            head.next = right;
        }
        return dummy.next;
    }
}
```

Summary

- Looking for smallest/largest one in k elements, we can use a heap.
- For a large input with a parallel task, we can consider divid input into two halves to solve the two sub problem.