Merge k Sorted Lists

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

Solution 1

If someone understand how priority queue works, then it would be trivial to walk through the codes.

My question: is that possible to solve this question under the same time complexity without implementing the priority queue?

```
public class Solution {
    public ListNode mergeKLists(List<ListNode> lists) {
        if (lists==null||lists.size()==0) return null;

        PriorityQueue<ListNode> queue= new PriorityQueue<ListNode>(lists.size(),n
ew Comparator<ListNode>(){
            @Override
            public int compare(ListNode o1,ListNode o2){
                if (o1.val<o2.val)
                    return -1;
                else if (o1.val==o2.val)
                    return 0;
                else
                    return 1;
            }
        });

        ListNode dummy = new ListNode(0);
        ListNode tail=dummy;

        for (ListNode node:lists)
            if (node!=null)
                queue.add(node);

        while (!queue.isEmpty()){
            tail.next=queue.poll();
            tail=tail.next;

            if (tail.next!=null)
                queue.add(tail.next);
        }
        return dummy.next;
    }
}
```

written by reeclapple original link here

Solution 2

```
ListNode *mergeKLists(vector<ListNode *> &lists) {
    if(lists.empty()){
        return nullptr;
    }
    while(lists.size() > 1){
        lists.push_back(mergeTwoLists(lists[0], lists[1]));
        lists.erase(lists.begin());
        lists.erase(lists.begin());
    }
    return lists.front();
}
ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
    if(l1 == nullptr){
        return l2;
    }
    if(l2 == nullptr){
        return l1;
    }
    if(l1->val <= l2->val){
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    }
    else{
        l2->next = mergeTwoLists(l1, l2->next);
        return l2;
    }
}
```

The second function is from Merge Two Sorted Lists.

The basic idea is really simple. We can merge first two lists and then push it back. Keep doing this until there is only one list left in vector. Actually, we can regard this as an iterative divide-and-conquer solution.

written by zxyperfect original link here

Solution 3

```java
public static ListNode mergeKLists(ListNode[] lists){
    return partion(lists,0,lists.length-1);
}

public static ListNode partion(ListNode[] lists,int s,int e){
    if(s==e)  return lists[s];
    if(s<e){
        int q=(s+e)/2;
        ListNode l1=partion(lists,s,q);
        ListNode l2=partion(lists,q+1,e);
        return merge(l1,l2);
    }else
        return null;
}

//This function is from Merge Two Sorted Lists.
public static ListNode merge(ListNode l1,ListNode l2){
    if(l1==null) return l2;
    if(l2==null) return l1;
    if(l1.val<l2.val){
        l1.next=merge(l1.next,l2);
        return l1;
    }else{
        l2.next=merge(l1,l2.next);
        return l2;
    }
}
```

written by mouqi123 original link here