Merge K Sorted Lists

Question:

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

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

Example 2:

```
Input: lists = []
Output: []
```

Example 3:

```
Input: lists = [[]]
Output: []
```

Approach 1:

My first approach was to insert all the elements of each linked list into an array.
Then sort the array and finally store all the values from the sorted array into a new linked list.

Solution 1:

```
def mergeKLists(self, lists):
        self.nodes = []
        head = point = ListNode(0)
        for l in lists:
                while l:
                        self.nodes.append(l.val)
                        l = l.next
        for x in sorted(self.nodes):
                point.next = ListNode(x)
                point = point.next
        return head.next
```

Time Complexity: O(N*logN) as sorting dominates the entire process.

Space Complexity: O(N)

Approach 2:

Using merging with divide and conquer to optimize the code.

Solution 2:

```
def mergeKLists(self, lists):
        amount = len(lists)
        interval = 1
        while interval < amount:
                for i in range(0, amount - interval, interval * 2):
                        lists[i] = self.merge2Lists(lists[i], lists[i + interval])
                interval *= 2
        return lists[0] if amount > 0 else None

def merge2Lists(self, l1, l2):
        head = point = ListNode(0)
        while l1 and l2:
                if l1.val <= l2.val:
                        point.next = l1
                        l1 = l1.next
                else:
                        point.next = l2
                        l2 = l1
                        l1 = point.next.next
                point = point.next
        if not l1:
                point.next=l2
        else:
                point.next=l1
        return head.next
```

Time Complexity: O(N*logk).

Space Complexity: O(1)