Approach #1 Brute Force [Accepted]

Intuition & Algorithm

- Traverse all linked lists and collect values of nodes into an array.
- Sort and iterate this array to get the proper value of nodes.
- Create a new sorted linked list and extend it with new nodes.

As for sorting, you can refer here for more about sorting algorithms.

Python

```
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        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
```

Complexity Analysis

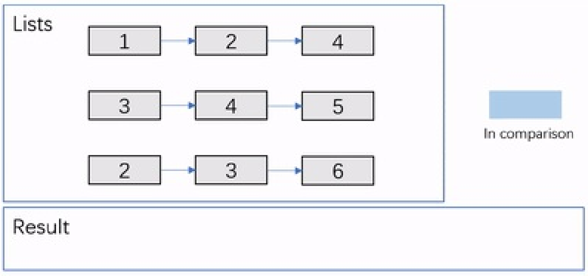
- Time complexity : $O(N\log N)$ where N is the total number of nodes.
  - Collecting all values costs $O(N)$ time.
  - A stable sorting algorithm costs $O(N\log N)$ time.
  - Iterating for creating linked list costs $O(N)$ time.

- Space complexity : $O(N)$.
  - Sorting cost $O(N)$ space (depends on the algorithm you choose).
  - Create a new linked list costs $O(N)$ space.

Approach #2 Compare one by one [Accepted]

Algorithm

- Compare every k nodes (head of every linked list) and get the node with the smallest value.
- Extend the final sorted linked list with seleted nodes.



Complexity Analysis

- Time complexity : $O(kN)$ where k is the number of linked lists.
    - Almost every selection of node in final linked costs $O(k)$ (k-1 times comparison).
    - There're N nodes in the final linked list.

- Space complexity :
    - $O(n)$ Create a new linked list costs $O(N)$ space.
    - $O(1)$ It's not hard to apply in-place method: connect selected nodes instead of creating new nodes to fill new linked list.

Approach #3 Optimize approach 2 by Priority Queue [Accepted]

Algorithm

Almost the same as the one above but optimize the **comparasion process** by **priority queue**. You can refer here for more information about it.

Python

```
from Queue import PriorityQueue

class Solution(object):
    def mergeKLists(self, lists):
        """
```

```
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        head = point = ListNode(0)
        q = PriorityQueue()
        for l in lists:
            if l:
                q.put((l.val, l))
        while not q.empty():
            val, node = q.get()
            point.next = ListNode(val)
            point = point.next
            node = node.next
            if node:
                q.put((node.val, node))
        return head.next
```

Complexity Analysis

- Time complexity : $O(N\log k)$ where k is the number of linked lists.
    - The comparasion cost will be reduced to $O(\log k)$ for every pop and insertion to priority queue. But finding the node with the smallest value just costs $O(1)$ time.
    - There're N nodes in the final linked list.

- Space complexity :
    - $O(n)$ Create a new linked list costs $O(N)$ space.
    - $O(k)$ The code I present applies in-place method which cost $O(1)$ space. And the priority queue (often implemented with heaps) costs $O(k)$ space (it's far less than N in most situation)

Approach #4 Merge lists one by one [Accepted]

Algorithm

Convert merge k lists problem to merge 2 lists (k-1) times. Here is the merge 2 lists problem page.

Complexity Analysis

- Time complexity : $O(kN)$ where k is the number of linked lists.
    - We can merge two sorted linked list in $O(n)$ time where n is the total number of nodes in two lists.
    - Sum up the merge process and we can get: $O(\sum_{i=1}^{k-1}(i*(N/k)+N/k)) = O(kN)$

- Space complexity : $O(1)$
    - We can merge two sorted linked list in $O(1)$ space.

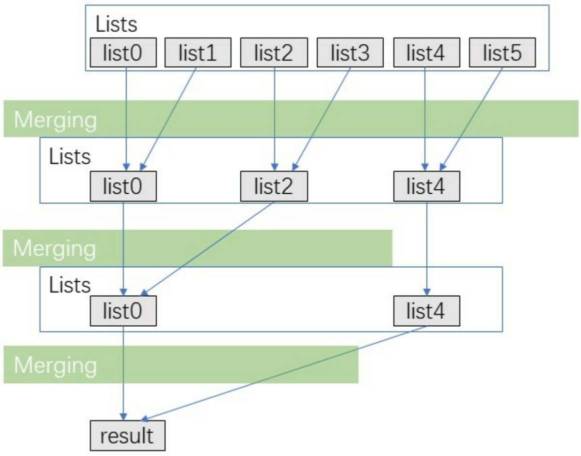
Approach #5 Merge with Divide And Conquer [Accepted]

Intuition & Algorithm

This approach walk alongside the one above but improved a lot. We don't need to traverse most nodes many times repeatly:

- Pair up k lists and merge each pair.
- After first pairing, k lists are merged into k/2 lists with average 2N/k length, then k/4, k/8 and so on.
- Repeat this procedure until we get the final sorted linked list.

Thus, we'll traverse almost N nodes per pairing and merging, and repeat this procedure about $\log_{2}{k}$ times.



Python

```
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        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 lists
```

```
    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
```

Complexity Analysis

- Time complexity : $O(N\log k)$ where k is the number of linked lists.
  - We can merge two sorted linked list in $O(n)$ time where n is the total number of nodes in two lists.
  - Sum up the merge process and we can get: $O(\sum*{i=1}^{log*{2}{k}}N = O(N\log k)$

- Space complexity : $O(1)$
  - We can merge two sorted linked list in $O(1)$ space.