Previous Next

23. Merge k Sorted List

Sept. 9, 2017 | 407.4K views

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
```

Solution

Approach 1: Brute Force

Intuition & Algorithm

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

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

Python

```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
```

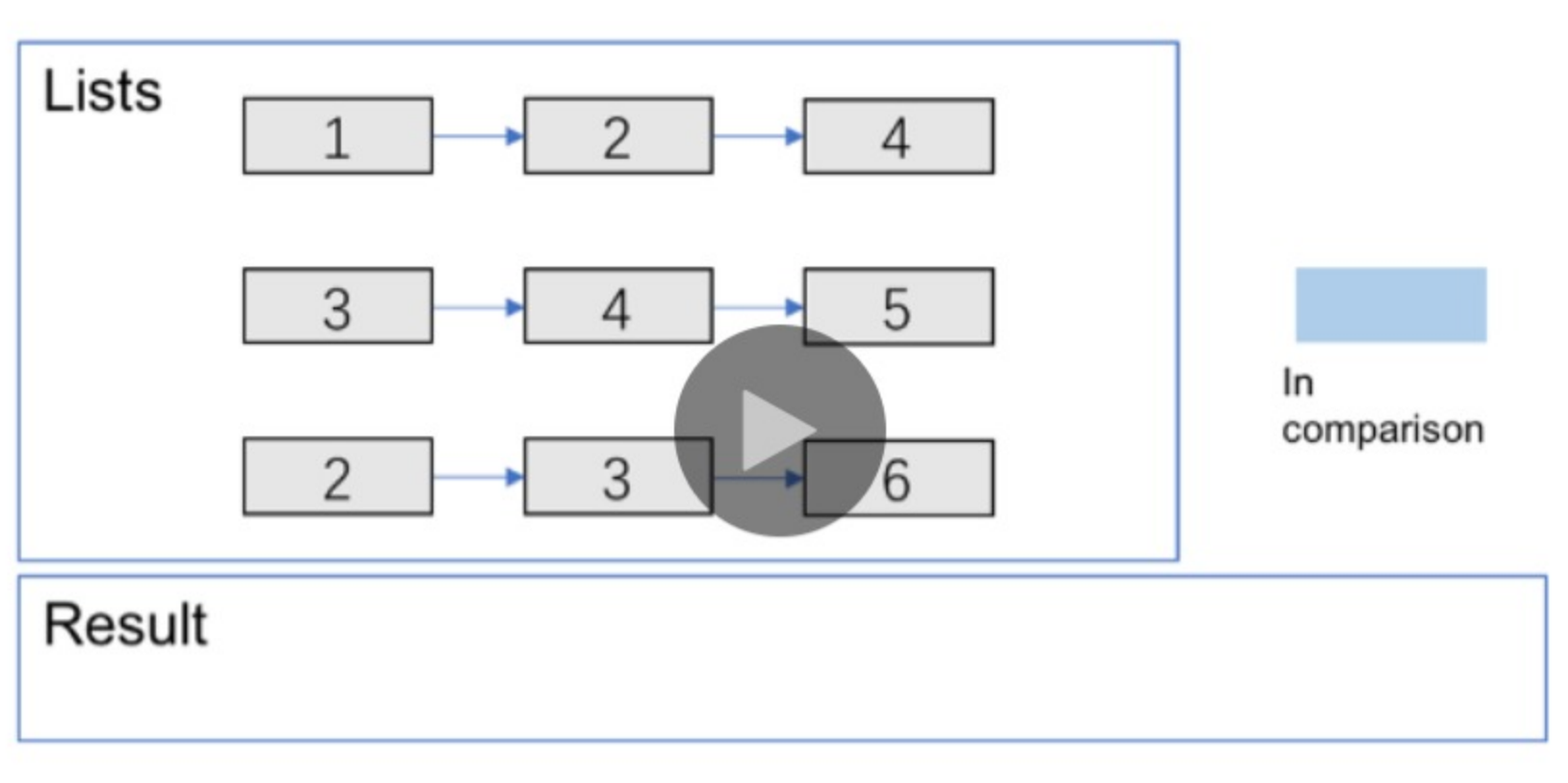
Complexity Analysis

- Time complexity : $O(N \log N)$ where N is the total number of nodes.
  - Collecting all the values costs $O(N)$ time.
  - A stable sorting algorithm costs $O(N \log N)$ time.
  - Iterating for creating the linked list costs $O(N)$ time.
- Space complexity : $O(N)$.
  - Sorting cost $O(N)$ space (depends on the algorithm you choose).
  - Creating a new linked list costs $O(N)$ space.

Approach 2: Compare one by one

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 the selected nodes.



Complexity Analysis

- Time complexity : $O(kN)$ where k is the number of linked lists.
  - Almost every selection of node in final linked costs $O(k)$ ($\mathrm{k-1}$ times comparison).
  - There are N nodes in the final linked list.
- Space complexity :
  - $O(n)$ Creating 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 the new linked list.

Approach 3: Optimize Approach 2 by Priority Queue

Algorithm

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

Python

```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 comparison 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 are N nodes in the final linked list.
- Space complexity :
  - $O(n)$ Creating a new linked list costs $O(n)$ space.
  - $O(k)$ The code above 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 situations).

Approach 4: Merge lists one by one

Algorithm

Convert merge k lists problem to merge 2 lists ($\mathrm{k-1}$) 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: $O(\sum_{i=1}^{k-1}(i*(\frac{N}{k})+\frac{N}{k}))=O(kN)$.
- Space complexity : $O(1)$
  - We can merge two sorted linked list in $O(1)$ space.

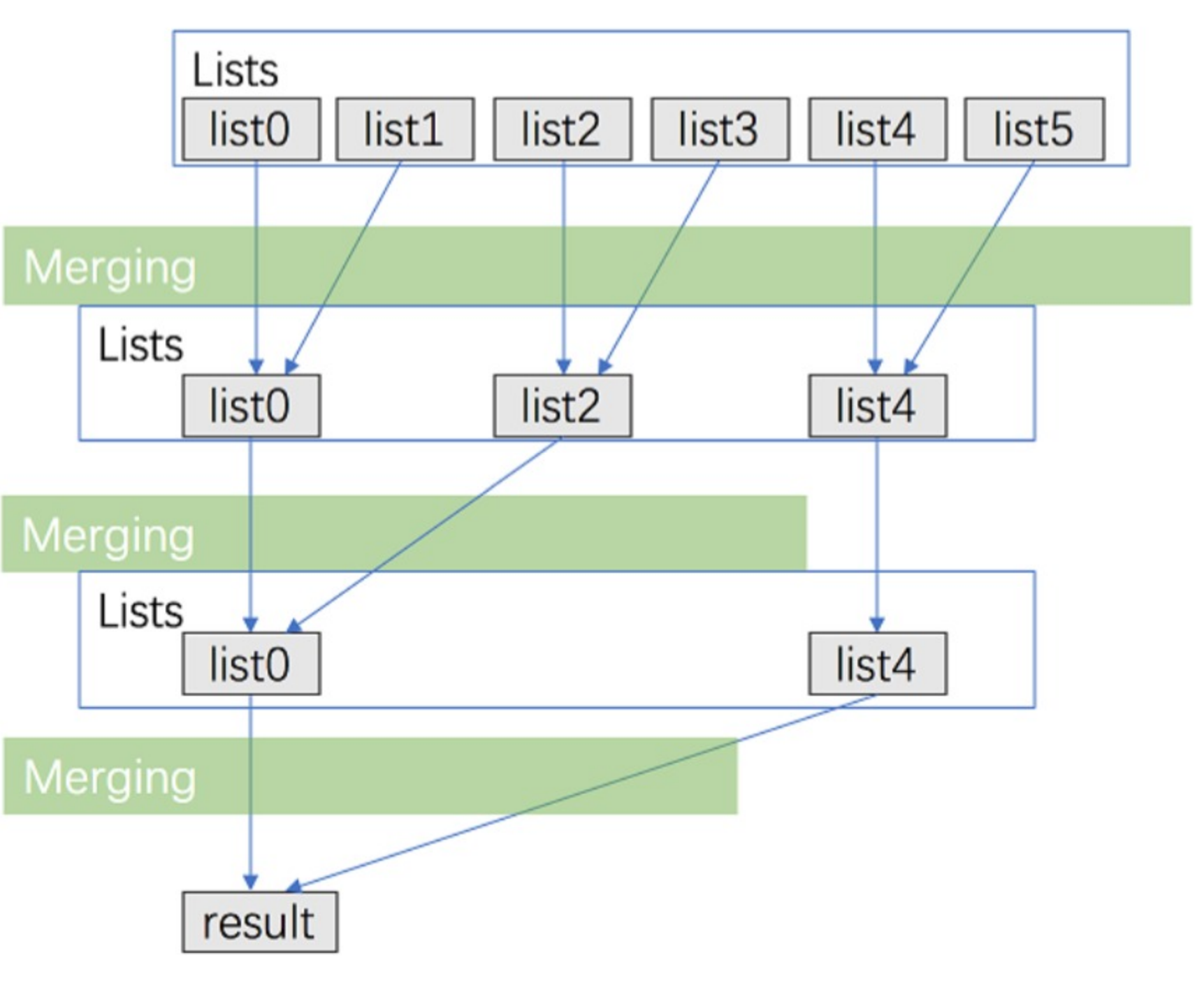
Approach 5: Merge with Divide And Conquer

Intuition & Algorithm

This approach walks alongside the one above but is improved a lot. We don't need to traverse most nodes many times repeatedly

- Pair up k lists and merge each pair.
- After the 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

```python
1  class Solution(object):
2      def mergeKLists(self, lists):
3          """
4          :type lists: List[ListNode]
5          :rtype: ListNode
6          """
7          amount = len(lists)
8          interval = 1
9          while interval < amount:
10             for i in range(0, amount - interval, interval * 2):
11                 lists[i] = self.merge2Lists(lists[i], lists[i + interval])
12             interval *= 2
13         return lists[0] if amount > 0 else lists
14
15     def merge2Lists(self, l1, l2):
16         head = point = ListNode(0)
17         while l1 and l2:
18             if l1.val <= l2.val:
19                 point.next = l1
20                 l1 = l1.next
21             else:
22                 point.next = l2
23                 l2 = l1
24                 l1 = point.next.next
25             point = point.next
26         if not l1:
27             point.next=l2
```

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: $O\big(\sum_{i=1}^{log_2 k} N\big) = O(N \log k)$
- Space complexity : $O(1)$
  - We can merge two sorted linked lists in $O(1)$ space.

Rate this article:

Previous Next

Comments: 141

Sort By

Type comment here... (Markdown is supported)

Preview Post

windliang ★ 1001 October 14, 2018 2:36 PM

All Approaches with java.
Approach 1

```
public ListNode mergeKLists(ListNode[] lists) {
```

Read More

246 Share Reply

SHOW 13 REPLIES

jinjiren ★ 337 February 3, 2019 3:30 PM

Approach 3 does not work in Python3 for 2 reasons:

1. python3 use `from queue import PriorityQueue`, instead of `from Queue`. (the case of 'Q')
2. TypeError '<' not supported between instances of 'ListNode' and 'ListNode':

Read More

94 Share Reply

SHOW 13 REPLIES

sudiptakarmakar ★ 40 March 29, 2018 5:57 PM

Why are we switching the list pointers on line 23 and 24 of solution 5?

40 Share Reply

SHOW 1 REPLY

filipp ★ 101 April 24, 2019 8:14 PM

Honestly, why this problem is "hard"? Medium at most IMO.

101 Share Reply

SHOW 12 REPLIES

sfdye ★ 850 September 21, 2018 2:25 PM

Another Python 3 version using `heapq` as priority queue.

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

Read More

25 Share Reply

SHOW 2 REPLIES

KillBug ★ 53 July 7, 2018 9:23 PM

PriorityQueue is changed in python 3, you have to define a extra class to make tuple compare

```
from queue import PriorityQueue
```

Read More

20 Share Reply

SHOW 1 REPLY

alexishe ★ 234 November 9, 2018 1:55 AM

why time complexity could be O(Nlogk) for priority queue?

11 Share Reply

SHOW 7 REPLIES

ndfhsledk ★ 16 January 29, 2019 9:18 AM

the time complexity of approach 5 is wrong.

k pairs, every 2 list merge to 1. The total number of merge times is k-1.

16 Share Reply

SHOW 9 REPLIES

christophermray1 ★ 40 September 4, 2018 11:03 AM

A fifth solution: divide and conquer, recursive:
This does the same thing as solution 4, but we use a recursive approach to choosing which lists to pair rather than iterating and forming pairs.

Read More

20 Share Reply

SHOW 1 REPLY

maddy9 ★ 47 December 30, 2018 12:29 AM

In Appraoch 3 why there is a need to create a new ListNode. when we can use the existing node. `point.next = node`

9 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 5 6 ... 14 15 ›