148. Sort List

https://leetcode.com/problems/sort-list/

February 24, 2022

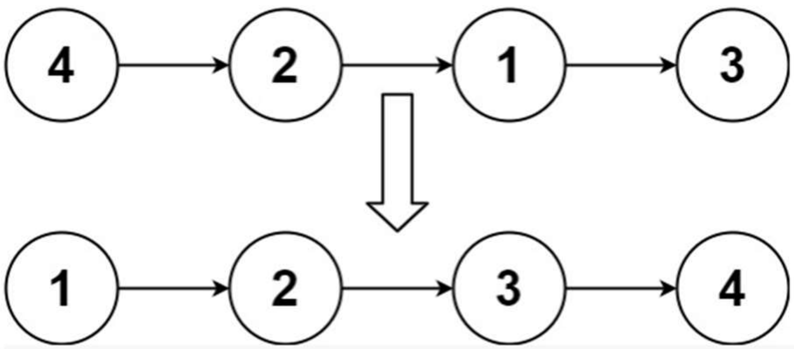
-Priyanshu Arya

148. Sort List

Medium 6491 213 Add to List Share

Given the `head` of a linked list, return *the list after sorting it in* ***ascending order***.

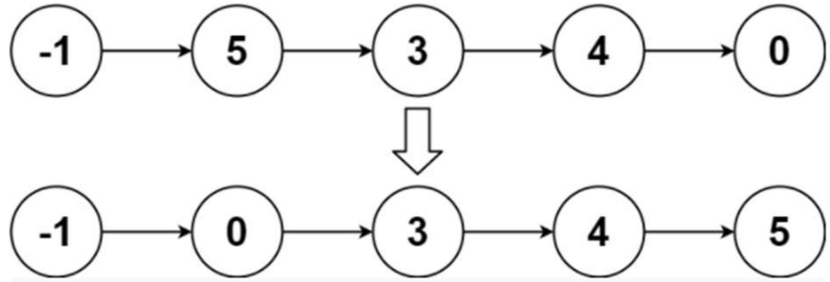
Example 1:




```
Input: head = [4,2,1,3]
Output: [1,2,3,4]
```

Ques. Sort the Linked List

Example 2:




```
Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]
```

Example 3:

```
Input: head = []
Output: []
```

Constraints:

- The number of nodes in the list is in the range `[0, 5 * 10^4]`.
- `-10^5 <= Node.val <= 10^5`

Follow up: Can you sort the linked list in `O(n logn)` time and `O(1)` memory (i.e. constant space)?

Algorithm	Best Case	Average Case	Worst Case	Space Complexity
Quick Sort	O(n logn)	O(n logn)	O(n^2)	O(1)
Bottom Up Merge Sort	O(n logn)	O(n logn)	O(n logn)	O(1)
Top Down Merge Sort	O(n logn)	O(n logn)	O(n logn)	O(n)

Quicksort is also one of the efficient algorithms with the average time complexity of O(*n*log*n*). But the worst-case time complexity is O(*n*^2). **Also, variations of the quick sort like randomized quicksort are not efficient for the linked list because unlike arrays, random access in the linked list is not possible in O(1) time.** If we sort the linked list using quicksort, we would end up using the head as a pivot element which may not be efficient in all scenarios.

That's the reason we are going to use merge sort for sorting a Linked List.

Merge - Sort

Merge sort Algorithm follows Divide and Conquer Strategy.

In Divide and Conquer Strategy we have 2 phases.

1- Divide Phase - Divide the problem into Sub problems

2- Conquer Phase - Repeatedly Solve each Sub problem independently and combine the result to form the original problem.

Merge Sort
- Top-Down (Recursive Method)
- Bottom Up (Iterative Method)

Top-Down Merge Sort (Recursive)

In this we - split List into sub Lists of equal sizes
- Sorts each sublist independently, and eventually merge all sorted List back

Algorithm:

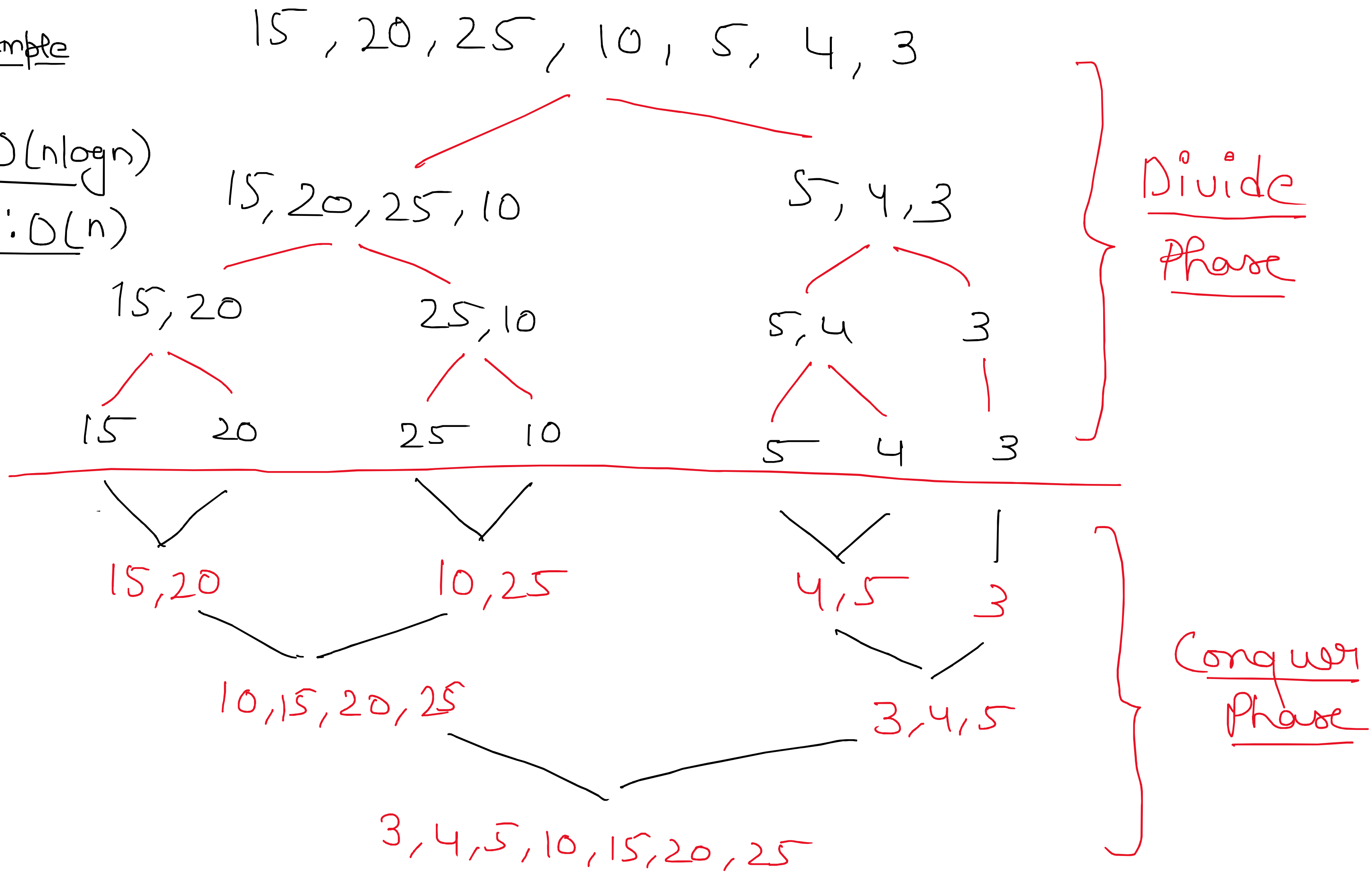
1) Recursively split the List into two equal halves, until there is only one node in the Linked List.

2) Recursively sort each sublist and combine it into a single sorted List.

For Example

Time: O(nlogn)

Space: O(n)



Algorithm:

merge sort(head):

mid node = mid list (head)

left = merge-sort (head)

right = merge-sort (mid node)

return merge(left, right)

Recurrence Relation

$$T(n) = \begin{cases} 1 & , n < 1 \\ \underbrace{2T(n/2)}_{\text{call left \& right}} + \underbrace{O(n)}_{\text{merge left \& right}} & n > 1 \end{cases}$$

Find middle of Linked List

(Fast $ Slow Pointer Approach)

Algorithm:

```
midList(head):
    fast, slow = head, head
    while fast != None and fast.next != None:
        slow = slow.next
        fast = fast.next.next
    return slow
```

Time: O(n)

Space: O(1)

[Find Middle Of Linked List]

Merge two Sorted Linked List (Iterative)

Algorithm

Time: O(n)

Space: O(1)

```
merge(list1, list2):
    dummy = curr = ListNode()
    while List1 and list2:
        if list1.val < list2.val:
            curr.next = list1
            list1 = list1.next
        else
            curr.next = list2
            list2 = list2.next
    if list1:
        curr.next = list1
    else:
        curr.next = list2
    return dummy.next
```

Loop run untill both have element

check if list1 is completed or list 2 and link all other remaining node link

return head of merge node List

[Merge two sorted linked lists]()

Merge two Sorted Linked List (Recursively)

Algorithm

Time : O(n)

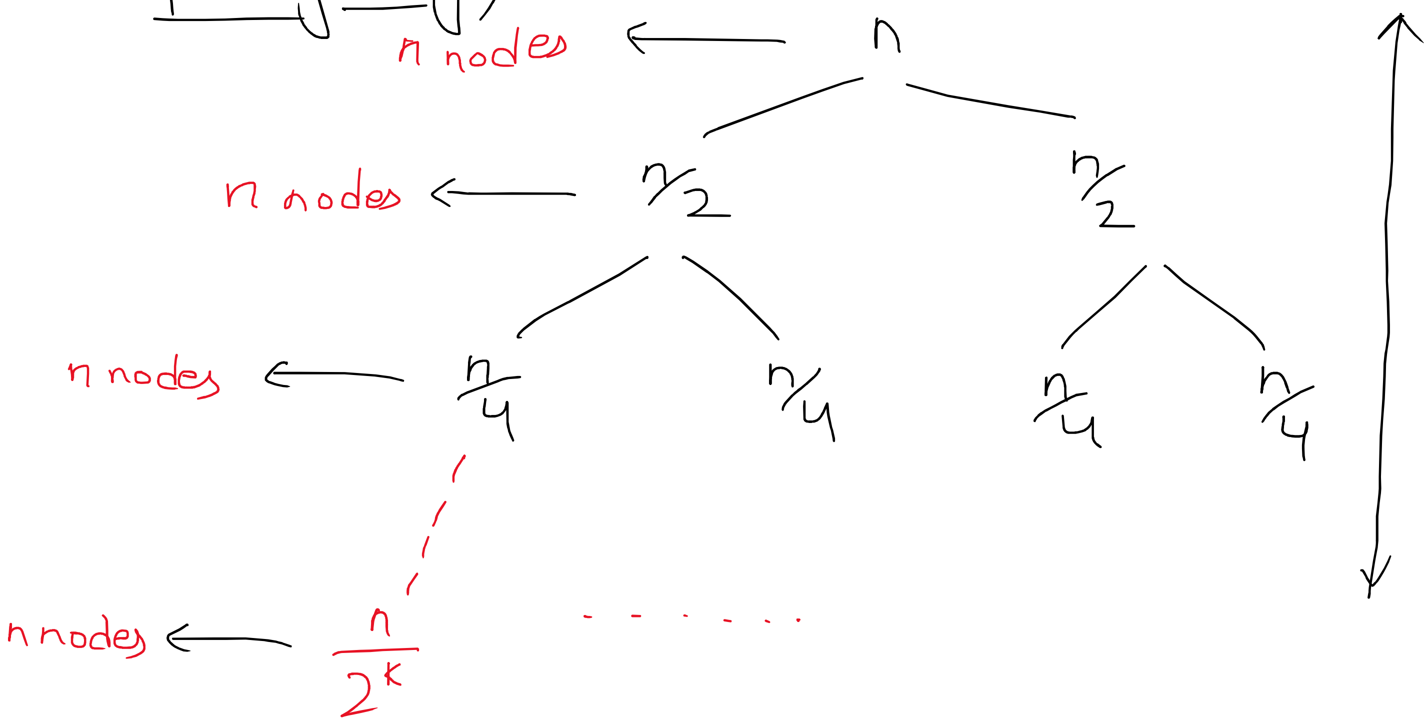
Space : O(1)

```
merge(list1, list2):
    if not list1 or not list2:
        return list1 or list2

    if list1.val < list2.val:
        list1.next = merge(list1.next, list2)
        return list1
    else
        list2.next = merge(list1, list2.next)
        return list2
```

[Merge two sorted linked lists](#)

Complexity Analysis



$$\text{Height} = \frac{n}{2^k}$$

Assume

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

taking $\log_2$ both side

$$= k \log_2 2$$

$$\boxed{k = \log_2 n} \leftarrow \text{Height}$$

Complexity = Total no. of nodes at every level * Height of tree

$$= n * \log_2 n$$

Time: $\boxed{O(n \log_2 n)}$ Space: $\boxed{O(n)}$

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def sortList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head is None or head.next is None:
            return head
        mid = self.getmidlist(head)
        left = self.sortList(head)
        right = self.sortList(mid)

        return self.merge(left, right)


    def merge(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:
        if not list1 or not list2:
            return list1 or list2

        if list1.val < list2.val:
            list1.next = self.merge(list1.next, list2)
            return list1
        else:
            list2.next = self.merge(list1, list2.next)
            return list2


    def getmidlist(self, head: Optional[ListNode]) -> Optional[ListNode]:
        fast, slow = head.next, head

        while fast != None and fast.next != None:
            fast = fast.next.next
            slow = slow.next

        start = slow.next
        slow.next = None
        return start
```

merge two sorted list

Get mid of List

splitting the node from mid part

Bottom-Up Merge Sort (Iterative)

The Bottom up Approach for Merge sort starts by splitting the problem into the smallest subproblems and iteratively merge the result to solve the original problem.

- First, the list is split into sublists of size 1 and merged iteratively in sorted order. The merged List is solved similarly.
- The process Continues untill we sort the entire list.

Follow up question askes us to do it in `O(1)` memory, and it is possible to do it, using bottom-up merge sort, which is much more difficult to implement during interview limits. What I expect that if you just explain the idea, without implementing this will be already quite good.

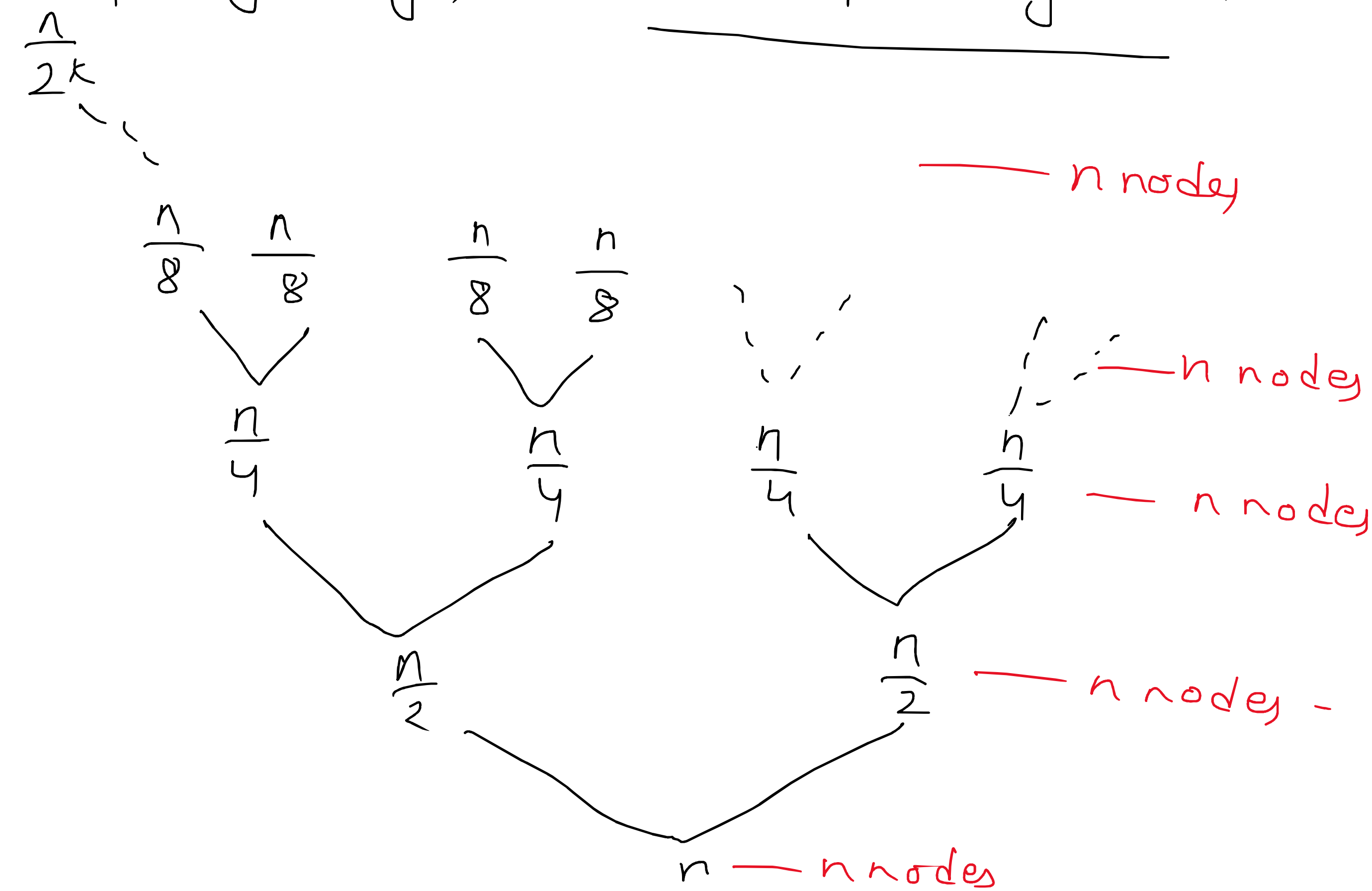
For Example N = 6

15 → 20 → 5 → 4 → 2 → 50

Iteration 1, Size 1	15	20	5	4	2	50
	15	20	4	5	2	50
Iteration 2, Size 3	15	20	4	5	2	50
	4	15	20	2	5	50
Iteration 3, Size 6	4	15	20	2	5	50
	2	4	5	15	20	50

Sorted List 2 → 4 → 5 → 15 → 20 → 50

Complexity Analysis Bottom-Up Merge Sort



Height = k

Let us Assume

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

taking $\log_2$ both Side

$$\log_2 n = \log_2 2^k$$

$$\log_2 n = k \log_2 2$$

$$k = \log_2 n$$

Time Complexity = Total no. of nodes at every level * Height of Tree

$$= n * \log_2 n$$

$$\text{Time} = O(n \log_2 n)$$

A bottom up merge sort for linked list uses a small (25 to 32) fixed size array of references (or pointers) to nodes, which would meet the **O(1)** space requirement. } Space : $O(1)$

If you have any doubt you can ping me back in comments.

References : http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/7-Sort/merge-sort5.html

https://leetcode.com/problems/sort-list/solution/

https://www.interviewbit.com/tutorial/merge-sort-algorithm/#:~:text=The%20Bottom%2DUp%20merge%20sort,of%20sorted%20array%20is%20achieved.

Thank you

If you like Please share this and feel free to connect for any queries.

GitHub: https://github.com/priyanshu-arya/DSA/tree/master/Leetcode%201
Discord: https://discord.gg/qPer56TP
Mail: priyanshuarya2482000@gmail.com