

AP Assignment - 03 Name : Chirag Yadav UID : 22BCS16377 Section : 608-B

Q1-Given a linked list. Print all the elements of the linked list separated by space followed.

CODE-

```
class Solution {    void
printList(Node head) {
Node temp = head;
while (temp != null) {
    System.out.print(temp.data);
    if (temp.next != null)
    {
        System.out.print(" ");
    }
    temp = temp.next;
}
}
```

Solution:-

Output Window

Compilation Results

Custom Input

Y.O.G.I. (AI Bot)

Problem Solved Successfully

[Suggest Feedback](#)

Test Cases Passed

1112 / 1112

Attempts : Correct / Total

2 / 5

Accuracy : 40%

Time Taken

2.08

Q2-Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return the linked list **sorted** as well.

Code:-

```
class Solution {    public ListNode
deleteDuplicates(ListNode head) {
    if (head == null) return null;

    ListNode current = head;

    while (current != null && current.next != null) {
if (current.val == current.next.val) {
current.next = current.next.next;
        } else {
```

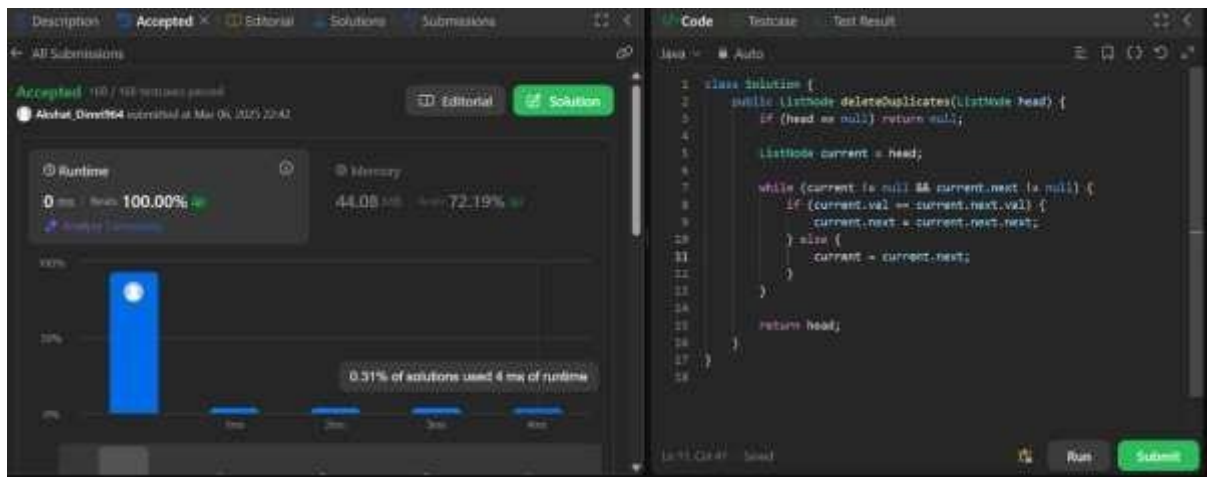
```

        current = current.next;
    }
}

return head;
}
}

```

Solution:-



Q3-Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Code:-

```

class Solution {
    public ListNode
reverseList(ListNode head) {
    ListNode prev = null;
    ListNode current = head;

    while (current != null) {
        ListNode nextNode = current.next;

```

```

        current.next = prev;

    prev = current;

    current = nextNode;

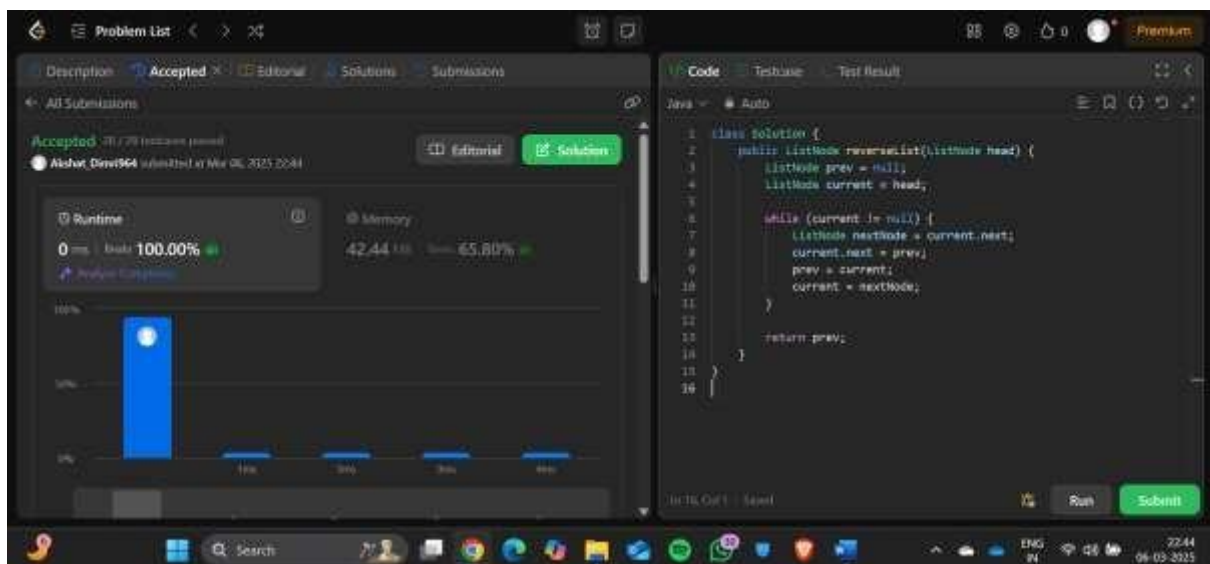
}

return prev;

}
}

```

Solution:-



Q4-You are given the head of a linked list. **Delete the middle node**, and return *the head of the modified linked list*.

The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor^{\text{th}}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

Code:-

```

class Solution {
    public ListNode
    deleteMiddle(ListNode head) {
        if (head ==
        null || head.next == null) {

```

```

        return null;
    }

    ListNode A = head, B = head, prev = null;

    while (B != null && B.next != null) {

        prev = A;

        A = A.next;
        B = B.next.next;

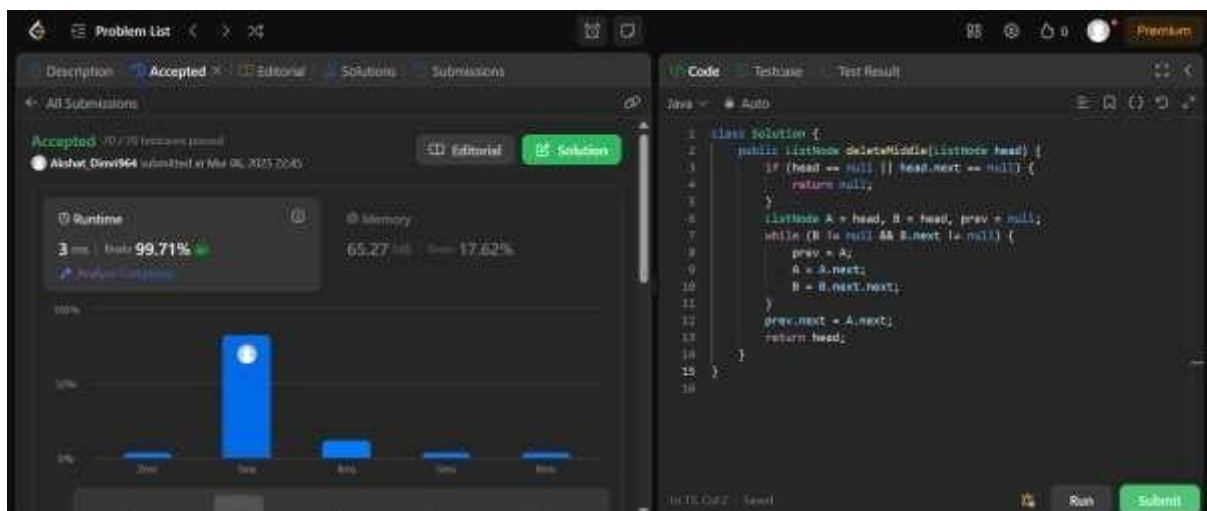
    }

    prev.next = A.next;

    return head;
}

```

Solution:-



Q5-You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Code:-

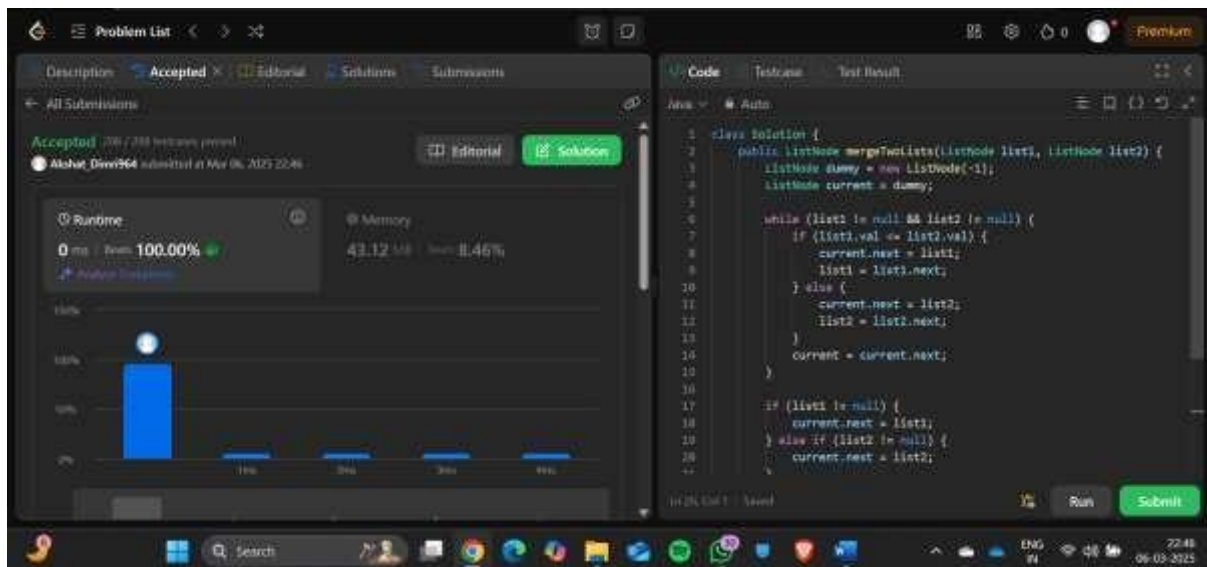
```
class Solution {    public ListNode mergeTwoLists(ListNode list1,
ListNode list2) {
    ListNode dummy = new ListNode(-1);
    ListNode current = dummy;

    while (list1 != null && list2 != null) {
        if (list1.val <= list2.val) {
current.next = list1;          list1
= list1.next;
        } else {
current.next = list2;
list2 = list2.next;
        }
        current = current.next;
    }

    if (list1 != null) {
current.next = list1;    }
    else if (list2 != null) {
current.next = list2;
    }

    return dummy.next;
}
}
```

Solution:-



Q6-Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter.**

Return true if there is a cycle in the linked list. Otherwise, return false.

Code:-

```
public class Solution {    public boolean
hasCycle(ListNode head) {    if (head == null ||
head.next == null) return false;

    ListNode slow = head, fast = head;

    while (fast != null && fast.next != null) {
slow = slow.next;        fast =
fast.next.next;

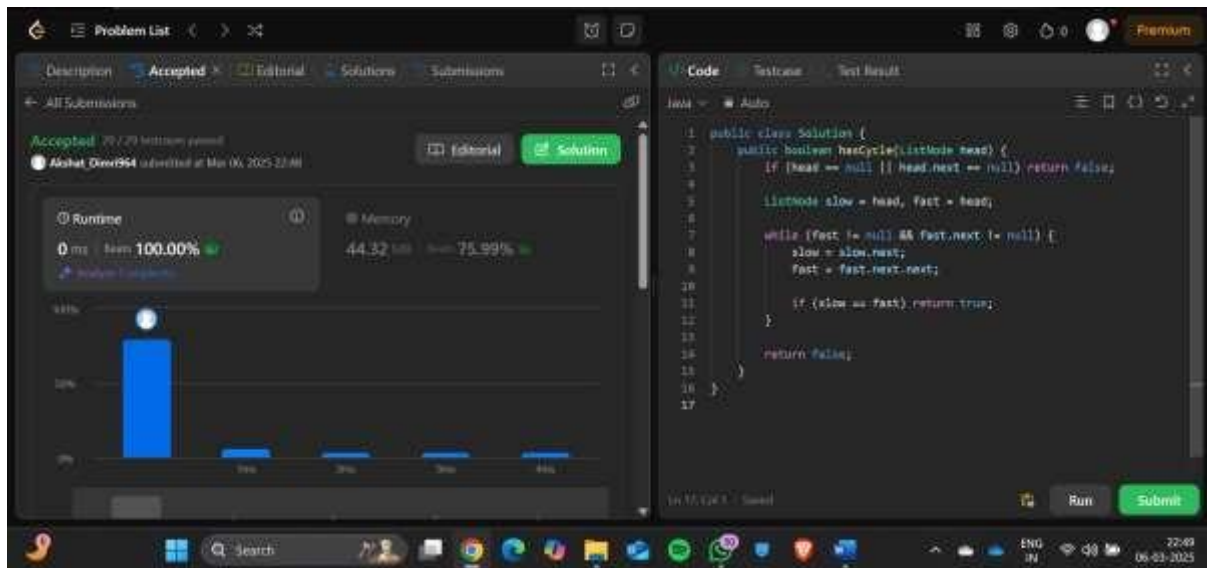
    if (slow == fast) return true;
    }
```

```

        return false;
    }
}

```

Solution:-



Q7-Given the head of a linked list, rotate the list to the right by k places.

Code:-

```

class Solution {
    public ListNode rotateRight(ListNode head,
int k) {
        if (head == null || head.next == null || k == 0)
return head;

```

```

        int length = 1;

```

```

        ListNode tail = head;

```

```

        while (tail.next != null) {

```

```

            tail = tail.next;
            length++;

```



```

    }

    k %= length;    if (k
== 0) return head;

    tail.next = head;    int
stepsToNewHead = length - k;
    ListNode newTail = head;

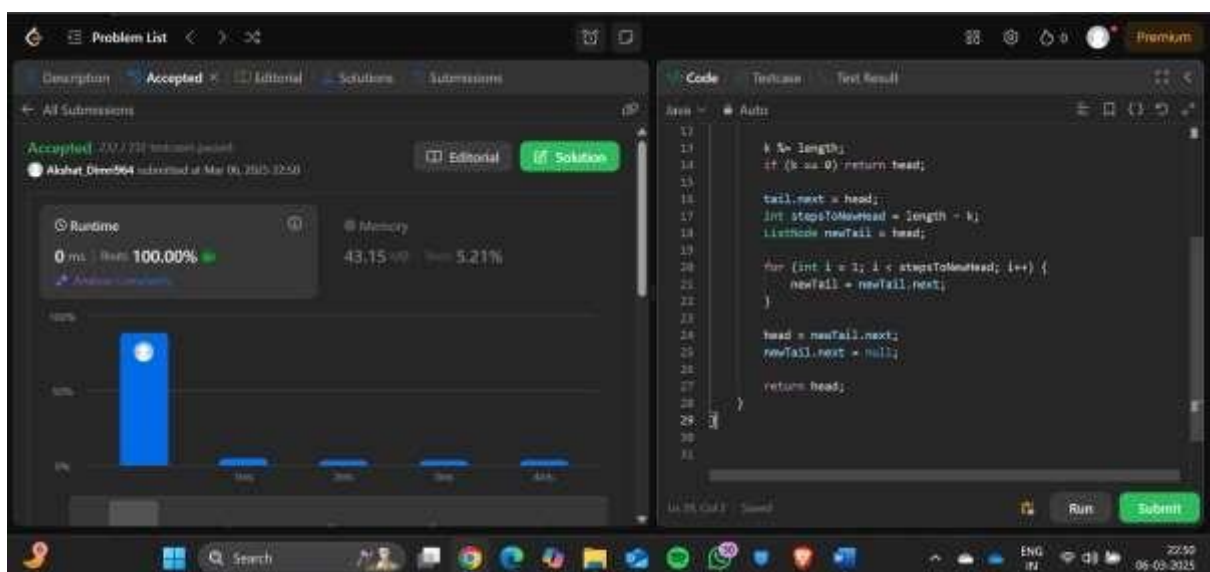
    for (int i = 1; i < stepsToNewHead; i++) {
newTail = newTail.next;
    }

    head = newTail.next;
newTail.next = null;

    return head;
}
}

```

Solution:-



The screenshot shows a code editor with a C++ solution for a linked list problem. The code is accepted, with 0ms runtime and 100.00% memory usage. The code implements a function to rotate a linked list by k steps.

```

13
14
15 k %= length;
16 if (k == 0) return head;
17
18 tail.next = head;
19 int stepsToNewHead = length - k;
20 ListNode newTail = head;
21
22 for (int i = 1; i < stepsToNewHead; i++) {
23     newTail = newTail.next;
24 }
25
26 head = newTail.next;
27 newTail.next = null;
28
29 return head;
30
31

```

The code is accepted, with 0ms runtime and 100.00% memory usage. The code implements a function to rotate a linked list by k steps.

Q8-Given the head of a linked list, return *the list after sorting it in **ascending order***.

Code:-

```
class Solution {    public ListNode sortList(ListNode
head) {        if (head == null || head.next == null)
return head;

        ListNode mid = getMid(head);
        ListNode left = sortList(head);
        ListNode right = sortList(mid);

        return merge(left, right);
    }

    private ListNode getMid(ListNode head) {
        ListNode slow = head, fast = head, prev = null;
        while (fast != null && fast.next != null) {
            prev = slow;        slow = slow.next;

            fast = fast.next.next;
        }
        if (prev != null) prev.next = null;
        return slow;
    }

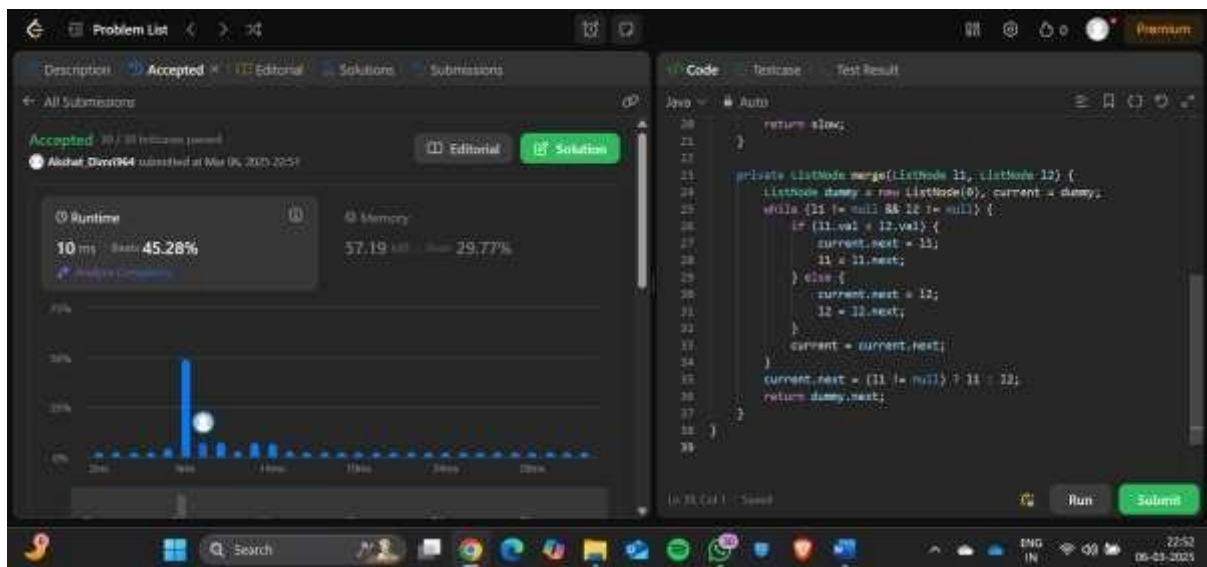
    private ListNode merge(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0), current = dummy;
        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
```

```

current.next = l1;          l1 =
l1.next;          } else {
    current.next = l2;
l2 = l2.next;
    }
    current = current.next;
}
current.next = (l1 != null) ? l1 : l2;
return dummy.next;
}
}

```

Solution:-



The screenshot shows a coding platform interface. On the left, the 'Accepted' status is confirmed with a runtime of 10 ms and memory usage of 57.19 KB. The right panel displays the Java code for merging two sorted linked lists. The code uses a dummy node and a while loop to merge the two lists.

```

20 return slow;
21 }
22
23 private ListNode merge(ListNode l1, ListNode l2) {
24     ListNode dummy = new ListNode(0); current = dummy;
25     while (l1 != null && l2 != null) {
26         if (l1.val < l2.val) {
27             current.next = l1;
28             l1 = l1.next;
29         } else {
30             current.next = l2;
31             l2 = l2.next;
32         }
33         current = current.next;
34     }
35     current.next = (l1 != null) ? l1 : l2;
36     return dummy.next;
37 }
38

```

Q9 - 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.

Code:-

```

public class Solution {    public ListNode
mergeKLists(ListNode[] lists) {

```

```

        PriorityQueue<ListNode> minHeap = new PriorityQueue<>(Comparator.comparingInt(node ->
node.val));    for (ListNode node : lists) {        if (node != null) {            minHeap.add(node);
        }
    }

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

    while (!minHeap.isEmpty()) {
        ListNode node = minHeap.poll();
        current.next = node;
        current = current.next;

        if (node.next != null) {
            minHeap.add(node.next);
        }
    }

    return dummy.next;
}

```

Solution:-

Problem List

Accepted

Editorial

Solutions

Submissions

All Submissions

Accepted

100 / 104 testcases passed

Editorial

Solution

Abhishek_Divya964 submitted at Mar 06, 2025 22:51

Runtime

6 ms

Fast: 26.59%

Memory

44.46 MB

Less: 62.66%

Profile Visualization

100%

0ms

50ms

100ms

150ms

200ms

250ms

300ms

350ms

400ms

450ms

500ms

550ms

600ms

650ms

700ms

750ms

800ms

850ms

900ms

950ms

1000ms

1050ms

1100ms

1150ms

1200ms

1250ms

1300ms

1350ms

1400ms

1450ms

1500ms

1550ms

1600ms

1650ms

1700ms

1750ms

1800ms

1850ms

1900ms

1950ms

2000ms

2050ms

2100ms

2150ms

2200ms

2250ms

2300ms

2350ms

2400ms

2450ms

2500ms

2550ms

2600ms

2650ms

2700ms

2750ms

2800ms

2850ms

2900ms

2950ms

3000ms

3050ms

3100ms

3150ms

3200ms

3250ms

3300ms

3350ms

3400ms

3450ms

3500ms

3550ms

3600ms

3650ms

3700ms

3750ms

3800ms

3850ms

3900ms

3950ms

4000ms

4050ms

4100ms

4150ms

4200ms

4250ms

4300ms

4350ms

4400ms

4450ms

4500ms

4550ms

4600ms

4650ms

4700ms

4750ms

4800ms

4850ms

4900ms

4950ms

5000ms

5050ms

5100ms

5150ms

5200ms

5250ms

5300ms

5350ms

5400ms

5450ms

5500ms

5550ms

5600ms

5650ms

5700ms

5750ms

5800ms

5850ms

5900ms

5950ms

6000ms

6050ms

6100ms

6150ms

6200ms

6250ms

6300ms

6350ms

6400ms

6450ms

6500ms

6550ms

6600ms

6650ms

6700ms

6750ms

6800ms

6850ms

6900ms

6950ms

7000ms

7050ms

7100ms

7150ms

7200ms

7250ms

7300ms

7350ms

7400ms

7450ms

7500ms

7550ms

7600ms

7650ms

7700ms

7750ms

7800ms

7850ms

7900ms

7950ms

8000ms

8050ms

8100ms

8150ms

8200ms

8250ms

8300ms

8350ms

8400ms

8450ms

8500ms

8550ms

8600ms

8650ms

8700ms

8750ms

8800ms

8850ms

8900ms

8950ms

9000ms

9050ms

9100ms

9150ms

9200ms

9250ms

9300ms

9350ms

9400ms

9450ms

9500ms

9550ms

9600ms

9650ms

9700ms

9750ms

9800ms

9850ms

9900ms

9950ms

10000ms

Code

Testcase

Test Result

```
1 10. stackoverflow.com: java.lang.OutOfMemoryError: Java heap space
2  if (node != null) {
3      minHeap.add(node);
4  }
5
6
7
8
9
10 listNode dummy = new ListNode(0);
11 ListNode current = dummy;
12
13 while (!minHeap.isEmpty()) {
14     ListNode node = minHeap.poll();
15     current.next = node;
16     current = current.next;
17
18     if (node.next != null) {
19         minHeap.add(node.next);
20     }
21 }
22
23 return dummy.next;
```

Run

Submit

22:53

06-03-2025