

Assignment:3

Name:Himanshu

UID:22BCS12456

Subject:AP

Section:IOT-607/B

1. Print Linked List:

Code:

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

Output:

Output Window

Compilation Results

Custom Input

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

Problem Solved Successfully

[Suggest Feedback](#)

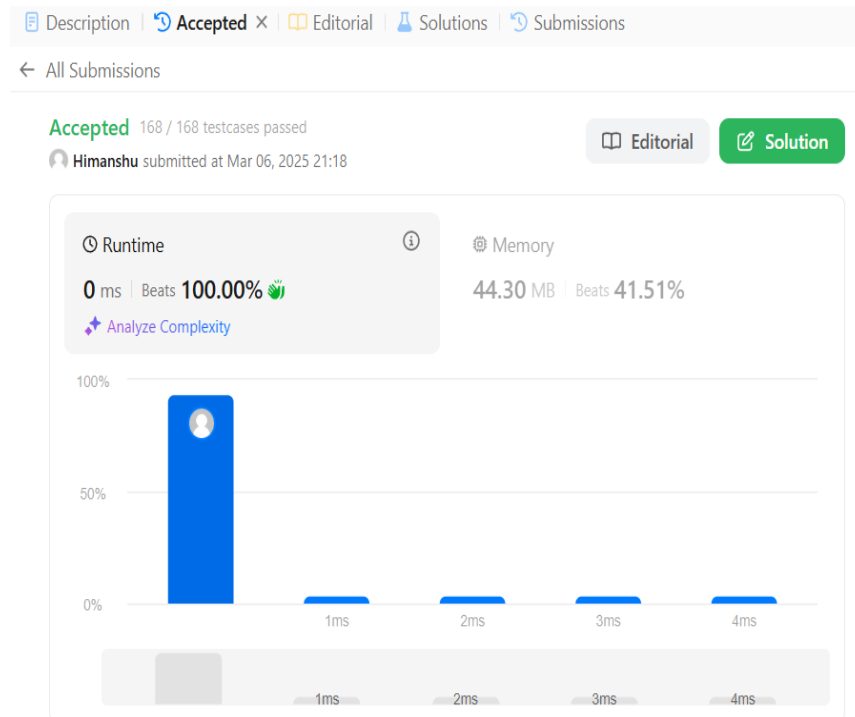
| | |
|--|--|
| <div>Test Cases Passed</div> <div>1112 / 1112</div> | <div>Attempts : Correct / Total</div> <div>1 / 1</div> <div>Accuracy : 100%</div> |
| <div>Points Scored </div> <div>1 / 1</div> <div>Your Total Score: 13 </div> | <div>Time Taken</div> <div>1.95</div> |

2. Remove duplicates from a sorted list:

Code:

```
class Solution {  
    public ListNode deleteDuplicates(ListNode head) {  
        ListNode curr = head;  
  
        while (curr != null) {  
            while (curr.next != null && curr.val == curr.next.val)  
                curr.next = curr.next.next;  
            curr = curr.next;  
        }  
  
        return head;  
    }  
}
```

Output:



3. Reverse a linked list:

Code:

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

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =
[1,2,3,4,5]

Output

[5,4,3,2,1]

Expected

[5,4,3,2,1]

4. Delete middle node of a list:

Code:

```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        ListNode dummy = new ListNode(0, head);
        ListNode slow = dummy;
        ListNode fast = dummy;

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

        // Delete the middle node.
        slow.next = slow.next.next;
        return dummy.next;
    }
}
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =
[1,3,4,7,1,2,6]

Output

[1,3,4,1,2,6]

Expected

[1,3,4,1,2,6]

5. Merge two sorted linked lists:

Code:

```
class Solution {  
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {  
        if (list1 == null || list2 == null)  
            return list1 == null ? list2 : list1;  
        if (list1.val > list2.val) {  
            ListNode temp = list1;  
            list1 = list2;  
            list2 = temp;  
        }  
        list1.next = mergeTwoLists(list1.next, list2);  
        return list1;  
    }  
}
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

list1 =
[1,2,4]

list2 =
[1,3,4]

Output

[1,1,2,3,4,4]

Expected

[1,1,2,3,4,4]

6. Detect a cycle in a linked list:

Code:

```
class Solution {  
    public boolean hasCycle(ListNode head) {  
        ListNode slow = head;  
        ListNode fast = head;  
  
        while (fast != null && fast.next != null) {  
            slow = slow.next;  
            fast = fast.next.next;  
            if (slow == fast)  
                return true;  
        }  
  
        return false;  
    }  
}
```

Output:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =
[3,2,0,-4]

pos =
1

Output

true

Expected

true

7. Rotate a list:

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;
        for (; tail.next != null; tail = tail.next)
            ++length;
        tail.next = head;

        final int t = length - k % length;
        for (int i = 0; i < t; ++i)
            tail = tail.next;
        ListNode newHead = tail.next;
        tail.next = null;

        return newHead;
    }
}
```

Output:

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =
[1,2,3,4,5]

k =
2

Output

[4,5,1,2,3]

Expected

[4,5,1,2,3]

8. Sort List:

Code:

```
class Solution {
    public ListNode sortList(ListNode head) {
        final int length = getLength(head);
        ListNode dummy = new ListNode(0, head);

        for (int k = 1; k < length; k *= 2) {
            ListNode curr = dummy.next;
            ListNode tail = dummy;
            while (curr != null) {
                ListNode l = curr;
                ListNode r = split(l, k);
                curr = split(r, k);
                ListNode[] merged = merge(l, r);
                tail.next = merged[0];
                tail = merged[1];
            }
        }

        return dummy.next;
    }

    private int getLength(ListNode head) {
        int length = 0;
        for (ListNode curr = head; curr != null; curr = curr.next)
            ++length;
        return length;
    }

    private ListNode split(ListNode head, int k) {
        while (--k > 0 && head != null)
            head = head.next;
        ListNode rest = head == null ? null : head.next;
        if (head != null)
```



```

        head.next = null;
    return rest;
}

private ListNode[] merge(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0);
    ListNode tail = dummy;

    while (l1 != null && l2 != null) {
        if (l1.val > l2.val) {
            ListNode temp = l1;
            l1 = l2;
            l2 = temp;
        }
        tail.next = l1;
        l1 = l1.next;
        tail = tail.next;
    }
    tail.next = l1 == null ? l2 : l1;
    while (tail.next != null)
        tail = tail.next;

    return new ListNode[] {dummy.next, tail};
}

```

Output:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =
[4,2,1,3]

Output

[1,2,3,4]

Expected

[1,2,3,4]

9. Merge k sorted lists:

Code:

```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        ListNode dummy = new ListNode(0);
        ListNode curr = dummy;
        Queue<ListNode> minHeap = new PriorityQueue<>((a, b) ->
            Integer.compare(a.val, b.val));

        for (final ListNode list : lists)
            if (list != null)
                minHeap.offer(list);

        while (!minHeap.isEmpty()) {
            ListNode minNode = minHeap.poll();
            if (minNode.next != null)
                minHeap.offer(minNode.next);
            curr.next = minNode;
            curr = curr.next;
        }

        return dummy.next;
    }
}
```

Output:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
lists =
[[1,4,5], [1,3,4], [2,6]]
```

Output

```
[1,1,2,3,4,4,5,6]
```

Expected

```
[1,1,2,3,4,4,5,6]
```

