

# Assignment

Name – JIGYA JAIN


UID – 22BCS50101

Section – 609-A

## 1. Print Linked ListCode –

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

## Submission –

My Submissions						All Submissions
						 Refresh
Time (IST)	Status	Marks	Lang	Test Cases	Code	
2025-02-13 23:31:41	Correct	0 ?	java	1112 / 1112	View	
2025-02-13 23:30:26	Correct	1	java	1112 / 1112	View	
2025-02-13 23:28:39	Wrong	0	java	0 / 1112	View	

```
1  
51  
52  
53 /* Node is defined as  
54 class Node {  
55     int data;  
56     Node next;  
57     Node(int x) {  
58         data = x;  
59         next = null;  
60     }  
61 }*/  
62 /*  
63 Print elements of a Linked List on console  
64 Head pointer input could be NULL as well for empty list  
65 */  
66  
67 class Solution {  
68     void printList(Node head) {  
69         while(head != null)  
70         {  
71             System.out.print(head.data + " ");  
72             head = head.next;  
73         }  
74     }  
75 }  
76
```

## 2. Remove duplicates from a sorted list –

### Code –

```
class Solution {  
    public ListNode deleteDuplicates(ListNode head) {  
        if(head == null || head.next == null) return head;  
        ListNode beg = head;  
        ListNode itr = head.next;  
        while(itr != null)  
        {  
            if(beg.val == itr.val)  
            {
```

```

        beg.next = null;
        itr = itr.next;
    }
    else{
        beg.next = itr;
        beg = itr;
        itr = itr.next;
    }
}
return head;
}
}

```

Submission –

Description   Editorial   Solutions   Submissions						Code	
Status	Language	Runtime	Memory	Notes		Java	Auto
5 Accepted Feb 16, 2025	Java	0 ms	43.8 MB			<pre> 10  */ 11  class Solution { 12      public ListNode deleteDuplicates(ListNode head) { 13          if(head == null    head.next == null) return head; 14 15          ListNode beg = head; 16          ListNode itr = head.next; 17          while(itr != null) 18          { 19              if(beg.val == itr.val) 20              { 21                  beg.next = null; 22                  itr = itr.next; 23              } 24              else{ </pre>	
4 Accepted Feb 14, 2025	Java	0 ms	44.2 MB				
3 Accepted Feb 14, 2025	Java	30 ms	44.3 MB				
2 Runtime Error Feb 14, 2025	Java	N/A	N/A				
1 Runtime Error Feb 14, 2025	Java	N/A	N/A				

### 3. Reverse a linked list –

Code –

```

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

```

Submission –

Status ▾	Language ▾	Runtime	Memory	Notes	Java ▾ 🔒 Auto
3 <b>Accepted</b> Feb 14, 2025	Java	0 ms	42.4 MB		9 * } 10 */ 11 12 class Solution { 13     public ListNode reverseList(ListNode head) { 14         if(head == null    head.next == null) return head; 15         ListNode ans = reverseList(head.next); 16         head.next.next = head; 17         head.next = null; 18 19         return ans; 20     } 21 }
2 <b>Accepted</b> Feb 14, 2025	Java	0 ms	42.6 MB		
1 <b>Compile Error</b> Feb 14, 2025	Java	N/A	N/A	+ Notes	

#### 4. Delete middle node of a list –

Code –

```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if(head.next == null) return null;
        ListNode mid = head;
        ListNode temp = head;
        ListNode ans = null;
        while(temp != null && temp.next != null)
        {
            ans = mid;
            mid = mid.next;
            temp = temp.next.next;
        }
        ans.next = mid.next;
        return head;
    }
}
```

Submission –

Status ▾	Language ▾	Runtime	Memory	Notes	Java ▾ 🔒 Auto
6 <b>Accepted</b> Feb 20, 2025	Java	3 ms	63.7 MB		11 class Solution { 12     public ListNode deleteMiddle(ListNode head) { 13         if(head.next == null) return null; 14 15         ListNode mid = head; 16         ListNode temp = head; 17         ListNode ans = null; 18 19         while(temp != null && temp.next != null) 20         { 21             ans = mid; 22             mid = mid.next; 23             temp = temp.next.next; 24         } 25         ans.next = mid.next; 26         return head; 27     } 28 }
5 <b>Runtime Error</b> Feb 20, 2025	Java	N/A	N/A		
4 <b>Accepted</b> Feb 20, 2025	Java	3 ms	65.5 MB		
3 <b>Accepted</b> Feb 20, 2025	Java	3 ms	64.3 MB		
2 <b>Accepted</b> Feb 20, 2025	Java	3 ms	64.4 MB		
1 <b>Accepted</b> Feb 18, 2025	Java	3 ms	63 MB		

## 5. Merge two sorted linked lists –

Code –

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode l1 = list1;
        ListNode l2 = list2;
        ListNode res = new ListNode(0);
        ListNode temp = res;

        while(l1 != null && l2 != null )
        {
            if(l1.val < l2.val)
            {
                temp.next = l1;
                l1 = l1.next;
            }
            else
            {
                temp.next = l2;
                l2 = l2.next;
            }
            temp = temp.next;
        }
        temp.next = (l1 != null ) ? l1 : l2;
        return res.next;
    }
}
```

Submission –



The screenshot shows a submission interface with a dark background. On the left, a green 'Accepted' status is displayed with the date 'Mar 05, 2025'. Below this, a 'Java' language tag is shown. To the right of the language tag, the execution time '0 ms' and memory usage '42.8 MB' are indicated. The main part of the screenshot displays the Java code for the 'mergeTwoLists' method, which is identical to the code provided in the previous block. The code is highlighted with a light blue background, and the line numbers 11 through 27 are visible on the left side of the code editor.

```
11 class Solution {
12     public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
13         ListNode l1 = list1;
14         ListNode l2 = list2;
15         ListNode res = new ListNode(0);
16         ListNode temp = res;
17         while(l1 != null && l2 != null )
18         {
19             if(l1.val < l2.val)
20             {
21                 temp.next = l1;
22                 l1 = l1.next;
23             }
24             else
25             {
26                 temp.next = l2;
27                 l2 = l2.next;
28             }
29             temp = temp.next;
30         }
31         temp.next = (l1 != null ) ? l1 : l2;
32         return res.next;
33     }
34 }
```

## 6. Detect a cycle in a linked list –

Code –

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

Submission –

The screenshot displays a submission interface with two main panels. The left panel shows a table of submissions, and the right panel shows the code for the selected submission.

	Description	Editorial	Solutions	Submissions	
	Status	Language	Runtime	Memory	Notes
2	Accepted a minute ago	Java	0 ms	44.5 MB	
1	Wrong Answer Feb 24, 2025	Java	N/A	N/A	+ Notes

The right panel shows the code for the selected submission (Submission 2):

```
1 class Solution {
2     public boolean hasCycle(ListNode head) {
3         if (head == null) {
4             return false;
5         }
6
7         ListNode slow = head;
8         ListNode fast = head;
9
10        while (fast != null && fast.next != null) {
11            slow = slow.next;
12            fast = fast.next.next;
13
14            if (slow == fast) {
15                return true;
16            }
17        }
18    }
19 }
```

## 7. Rotate a list –

Code –

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

```

    }

    int nodes = 1;
    ListNode beg = head;
    ListNode end = head;

    while (end.next != null) {
        end = end.next;
        nodes++;
    }

    int rotate = k % nodes;
    if (rotate == 0) {
        return head;
    }
    end.next = head;

    ListNode newTail = head;
    for (int i = 0; i < nodes - rotate - 1; i++) {
        newTail = newTail.next;
    }
    ListNode newHead = newTail.next;
    newTail.next = null;
    return newHead;
}
}

```

Submission –

The screenshot displays a submission interface for a programming problem. On the left, the submission status is 'Accepted' with 232/232 testcases passed. The user 'Jigya\_jain' submitted it on Mar 06, 2025 at 22:27. The runtime is 0 ms, beating 100.00% of other submissions. The memory usage is 42.76 MB, beating 36.07%. A blue bar chart shows the user's performance relative to others. On the right, the Java code is displayed, which implements a function to rotate a linked list to the right by k places.

**Submission Details:**

- Status: Accepted (232 / 232 testcases passed)
- User: Jigya\_jain (submitted at Mar 06, 2025 22:27)
- Runtime: 0 ms | Beats 100.00%
- Memory: 42.76 MB | Beats 36.07%

**Code:**

```

10  /*
11  class Solution {
12      public ListNode rotateRight(ListNode head, int k) {
13          if (head == null || head.next == null) {
14              return head;
15          }
16
17          int nodes = 1;
18          ListNode beg = head;
19          ListNode end = head;
20
21          while (end.next != null) {
22              end = end.next;
23              nodes++;
24          }
25
26          int rotate = k % nodes;
27          if (rotate == 0) {
28              return head;
29          }
30
31          end.next = head;

```

## 8. Sort List –

Code –

```
class Solution {
```

```
    public ListNode sortList(ListNode head) {  
        if (head == null) {  
            return null;  
        }  
    }
```

```
        PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) ->  
Integer.compare(a.val, b.val));
```

```
        while (head != null) {  
            minHeap.add(head);  
            head = head.next;  
        }
```

```
        ListNode dummy = new ListNode(0);  
        ListNode temp = dummy;
```

```
        while (!minHeap.isEmpty()) {  
            temp.next = minHeap.poll();  
            temp = temp.next;  
        }
```

```
        temp.next = null; // Ensure the last node points to null  
        return dummy.next;
```

```
    }  
}
```

Submission –

The screenshot displays the LeetCode submission interface. On the left, the 'Submissions' tab is active, showing a submission with status 'Accepted', language 'Java', runtime '44 ms', and memory '55.6 MB'. On the right, the 'Code' editor shows the following Java code:

```
1  /**  
2   * Definition for singly-linked list.  
3   * public class ListNode {  
4   *     int val;  
5   *     ListNode next;  
6   *     ListNode() {}  
7   *     ListNode(int val) { this.val = val; }  
8   *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }  
9   * }  
10 */  
11 class Solution {  
12     public ListNode sortList(ListNode head) {  
13         if (head == null) {  
14             return null;  
15         }  
16  
17         PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) -> Integer.compare(a.val, b.val));
```

9. Merge k sorted lists –

Code –

```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        PriorityQueue<int[]> minHeap = new PriorityQueue<>((a, b) ->
Integer.compare(a[0], b[0]));
        ListNode head = null, temp = null;

        for (int i = 0; i < lists.length; i++) {
            if (lists[i] != null) {
                minHeap.add(new int[] {lists[i].val, i});
            }
        }

        while (!minHeap.isEmpty()) {
            int[] smallest = minHeap.poll();
            int i = smallest[1];

            if (head == null) {
                head = lists[i];
                lists[i] = lists[i].next;
                temp = head;
            } else {
                temp.next = lists[i];
                temp = temp.next;
                lists[i] = lists[i].next;
            }

            if (lists[i] != null) {
                minHeap.add(new int[] {lists[i].val, i});
            }
        }

        return head;
    }
}
```

Submission –



DescriptionAccepted ×EditorialSolutionsSubmissions

Status

Language

Runtime

Memory

Notes

1

Accepted  
a few seconds ago

Java

5 ms

44.6 MB

Code

JavaAuto

```
8      *      ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9      *  }
10     */
11     class Solution {
12     public ListNode mergeKLists(ListNode[] lists) {
13         PriorityQueue<int[]> minHeap = new PriorityQueue<>((a, b) -> a[0] - b[0]);
14         ListNode head = null, temp = null;
15
16         for (int i = 0; i < lists.length; i++) {
17             if (lists[i] != null) {
18                 minHeap.add(new int[]{lists[i].val, i});
19             }
20         }
21
22         while (!minHeap.isEmpty()) {
23             int[] smallest = minHeap.poll();
24             int i = smallest[1];
25             temp = new ListNode(smallest[0], lists[i].next);
26             lists[i] = lists[i].next;
27             if (lists[i] != null) {
28                 minHeap.add(new int[]{lists[i].val, i});
29             }
30             head = temp;
31         }
32         return head;
33     }
34 }
```