

1. Print linked list

The screenshot shows a coding platform interface for the problem "Print linked list". The "Output Window" on the left displays "Compilation Results" and "Problem Solved Successfully". It shows that 1112 out of 1112 test cases passed, with 1 correct attempt out of 1 total, 100% accuracy, and a time taken of 0.09 seconds. The "Solve Next" section lists other problems: "Node at a given index in linked list", "Delete Alternate Nodes", and "Insert in Middle of Linked List". The code editor on the right shows a C++ solution for printing a linked list.

```
1 // Driver Code Ends
19
20
21 class Solution {
22 public:
23     void printList(Node* head) {
24         while (head) {
25             cout << head->data << " ";
26             head = head->next;
27         }
28     }
29 };
30 // Driver Code Ends
```

2. Remove Duplicates from Sorted List

The screenshot shows a coding platform interface for the problem "83. Remove Duplicates from Sorted List". The problem description states: "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." Example 1 shows an input list [1, 1, 2] and an output list [1, 2]. Example 2 shows an input list [1, 2, 3, 4, 5] and an output list [1, 2, 3, 4, 5]. The code editor on the right shows a C++ solution for removing duplicates from a sorted linked list.

```
1 class Solution {
2 public:
3     ListNode* deleteDuplicates(ListNode* head) {
4         ListNode* curr = head;
5         while (curr && curr->next) {
6             if (curr->val == curr->next->val) {
7                 curr->next = curr->next->next;
8             } else {
9                 curr = curr->next;
10            }
11        }
12        return head;
13    };
14 };
15
16 void printList(ListNode* head) {
17     while (head) {
18         cout << head->val << " ";
19         head = head->next;
20     }
21 }
```

3. Reverse a linked list

The screenshot shows a coding platform interface for the problem "206. Reverse Linked List". The problem description states: "Given the head of a singly linked list, reverse the list, and return the reversed list." Example 1 shows an input list [1, 2, 3, 4, 5] and an output list [5, 4, 3, 2, 1]. Example 2 shows an input list [1, 2] and an output list [2, 1]. The code editor on the right shows a C++ solution for reversing a linked list.

```
1 class Solution {
2 public:
3     ListNode* reverseList(ListNode* head) {
4         ListNode* prev = NULL;
5         ListNode* curr = head;
6         while (curr) {
7             ListNode* next = curr->next;
8             curr->next = prev;
9             prev = curr;
10            curr = next;
11        }
12        return prev;
13    };
14 };
15
16 void printList(ListNode* head) {
17     while (head) {
18         cout << head->val << " ";
19         head = head->next;
20     }
21 }
```

4. Delete the middle node of a linked list

2095. Delete the Middle Node of a Linked List


Medium

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 $\lfloor n / 2 \rfloor^{th}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

- For $n = 1, 2, 3, 4$, and 5 , the middle nodes are $0, 1, 1, 2$, and 2 , respectively.

Example 1:



Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes are written below.
Since $n = 7$, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.

4.4K 71 50 Online

```
1 class Solution {
2 public:
3     ListNode* deleteMiddle(ListNode* head) {
4         if (!head || !head->next) return nullptr;
5         ListNode* slow = head, *fast = head, *prev = nullptr;
6         while (fast && fast->next) {
7             prev = slow;
8             slow = slow->next;
9             fast = fast->next->next;
10        }
11        prev->next = slow->next;
12        delete slow;
13        return head;
14    }
15
16    void printList(ListNode* head) {
17        while (head) {
18            cout << head->val << " ";
19            head = head->next;
20        }
21    }
22 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head =

5. Merge two sorted lists

21. Merge Two Sorted Lists

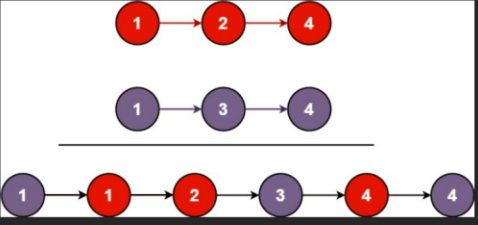
Easy

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.

Example 1:



Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

22.9K 411 429 Online

```
1 class Solution {
2 public:
3     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
4         if (!list1) return list2;
5         if (!list2) return list1;
6
7         if (list1->val < list2->val) {
8             list1->next = mergeTwoLists(list1->next, list2);
9             return list1;
10        } else {
11            list2->next = mergeTwoLists(list1, list2->next);
12            return list2;
13        }
14    }
15 };
16
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

list1 =

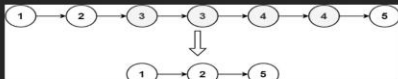
6. Remove duplicates from sorted list II

82. Remove Duplicates from Sorted List II

Medium

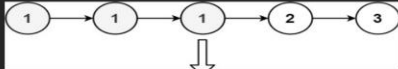
Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.

Example 1:



Input: head = [1,2,3,3,4,4,5]
Output: [1,2,5]

Example 2:



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

9.1K 82 62 Online

```
1 class Solution {
2 public:
3     ListNode* deleteDuplicates(ListNode* head) {
4         if (!head || !head->next) return head;
5
6         ListNode dummy(0);
7         dummy->next = head;
8         ListNode* prev = &dummy;
9
10        while (head) {
11            bool duplicate = false;
12            while (head->next && head->val == head->next->val) {
13                duplicate = true;
14                head = head->next;
15            }
16            if (duplicate) {
17                prev->next = head->next;
18                head = head->next;
19            } else {
20                prev->next = head;
21                prev = head;
22                head = head->next;
23            }
24        }
25        return dummy->next;
26    }
27 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

head =

7. Linked list cycle

141. Linked List Cycle Solved

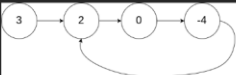
Easy Topics Companies

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`.

Example 1:



Input: `head = [3,2,0,-4]`, `pos = 1`
Output: `true`
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:



Input: `head = [1,2]`, `pos = 0`
Output: `false`

16.2K 352 186 Online

```
1 class Solution {
2 public:
3     bool hasCycle(ListNode *head) {
4         if (!head || !head->next) return false;
5         ListNode *slow = head, *fast = head;
6         while (fast && fast->next) {
7             slow = slow->next;
8             fast = fast->next->next;
9             if (slow == fast) return true;
10        }
11        return false;
12    }
13};
```

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3

Input: head =

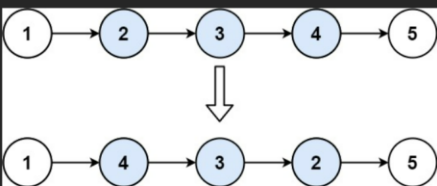
8. Reverse linked list II

92. Reverse Linked List II

Medium Topics Companies

Given the `head` of a singly linked list and two integers `left` and `right` where `left <= right`, reverse the nodes of the list from position `left` to position `right`, and return the reversed list.

Example 1:



Input: `head = [1,2,3,4,5]`, `left = 2`, `right = 4`
Output: `[1,4,3,2,5]`

Example 2:

Input: `head = [5]`, `left = 1`, `right = 1`
Output: `[5]`

12K 151 101 Online

```
1 prev = prev->next;
2 }
3
4 ListNode* curr = prev->next;
5 ListNode* nextNode;
6
7 for (int i = 0; i < right - left; ++i) {
8     nextNode = curr->next;
9     curr->next = nextNode->next;
10    nextNode->next = prev->next;
11    prev->next = nextNode;
12 }
13
14 return dummy->next;
15 }
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input: head =

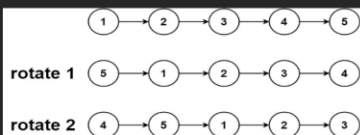
9. Rotate list

61. Rotate List Solved

Medium Topics Companies

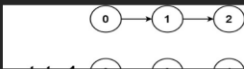
Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:



Input: `head = [1,2,3,4,5]`, `k = 2`
Output: `[4,5,1,2,3]`

Example 2:



Input: `head = [0,1,2]`, `k = 0`
Output: `[0,1,2]`

10.2K 101 81 Online

```
1 class Solution {
2 public:
3     ListNode* rotateRight(ListNode* head, int k) {
4         if (!head || !head->next || k == 0) return head;
5
6         ListNode* temp = head;
7         int length = 1;
8         while (temp->next) {
9             temp = temp->next;
10            length++;
11        }
12
13        temp->next = head;
14        k = k % length;
15        int steps = length - k;
16
17        while (steps-- > 0) temp = temp->next;
18        head = temp->next;
19        temp->next = null;
20    }
21};
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input: head =

10. Sort list

148. Sort List

Medium

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]

Example 2:

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

```
class Solution {
public:
    ListNode* merge(ListNode* left, ListNode* right) {
        if (!left) return right;
        if (!right) return left;

        if (left->val < right->val) {
            left->next = merge(left->next, right);
            return left;
        } else {
            right->next = merge(left, right->next);
            return right;
        }
    }

    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;

        ListNode *slow = head, *fast = head, *prev = nullptr;
        while (fast && fast->next) {
            // ... (rest of the code)
        }
    }
}
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input: head =

11. Linked list cycle II

142. Linked List Cycle II

Medium

Given the **head** of a linked list, return the node where the cycle begins. If there is no cycle, return **null**.

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 (**0-indexed**). It is **-1** if there is no cycle. **Note that pos is not passed as a parameter.**

Do not modify the linked list.

Example 1:

Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.

```
class Solution {
public:
    ListNode* detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;

            if (slow == fast) {
                slow = head;
                while (slow != fast) {
                    slow = slow->next;
                    fast = fast->next;
                }
                return slow;
            }
        }

        return nullptr;
    }
}
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input: head =