# **AP Assignment 3**

Name: Dhruv Bhatia

UID: 22BCS12196

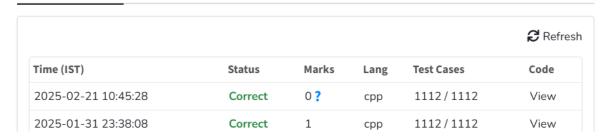
Section: 608-B

# 1) Print Linked List

```
class Solution {
  public:
    void printList(Node *head) {
      Node*temp=head;
      while(temp!=NULL){
        cout<<temp->data<<" ";
        temp=temp->next;
      }
  }
};
```

#### My Submissions

All Submissions



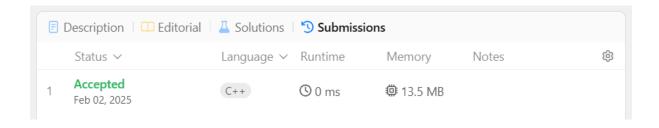
#### 2) Remove duplicates from a sorted list

```
while(temp && temp->next){
       if(temp->val==temp->next->val){
          temp->next=temp->next->next;
       }
       else{
          temp=temp->next;
       }
     }
     return head;
};
                                                                                  [] <
  🗉 Description | 🛄 Editorial | 🚣 Solutions | 🖰 Submissions
      Status ∨
                            Language ∨ Runtime
                                                    Memory
                                                                                    (6)
                                                                 Notes
      Accepted
                            C++
                                       (3) 0 ms
                                                    @ 16.2 MB
      Mar 05, 2025
```

#### 3) Reverse a linked list

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode*prev=NULL;
        ListNode*curr=head;
        while(curr!=NULL){
            ListNode*next=curr->next;
            curr->next=prev;
            prev=curr;
            curr=next;
        }
        return prev;
```

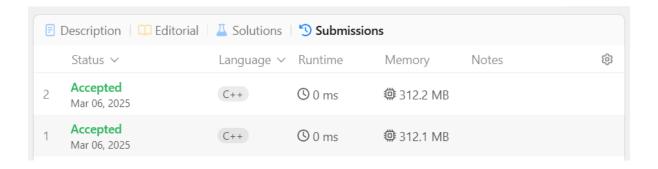
```
};
```



## 4) Delete middle node of a list

```
class Solution {
public:
  ListNode* deleteMiddle(ListNode* head) {
    if(!head->next){
       return NULL;
    }
    if(!head->next->next){
       head->next = NULL;
       return head;
    }
    ListNode* slow = head;
    ListNode* fast = head;
    ListNode* prev = NULL;
    while(fast && fast->next){
       prev=slow;
       slow = slow->next;
       fast = fast->next->next;
     }
    prev->next=slow->next;
    delete slow;
```

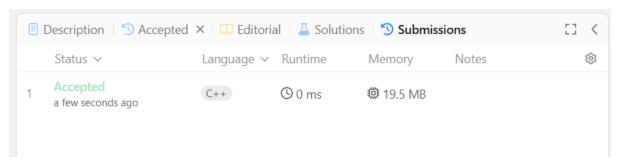
```
return head;
};
```



## 5) Merge two sorted linked lists

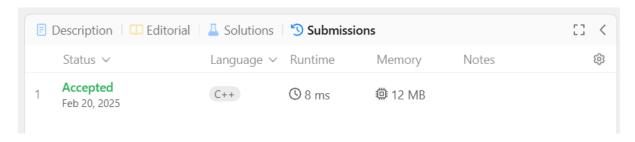
```
class Solution {
public:
  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
     ListNode dummy(0);
     ListNode* tail = &dummy;
     while (list1 && list2) {
       if (list1->val <= list2->val) {
          tail->next = list1;
          list1 = list1 -> next;
       } else {
          tail->next = list2;
          list2 = list2 - next;
       }
       tail = tail->next;
     tail->next = list1 ? list1 : list2;
     return dummy.next;
  }
```

**}**;



### 6) Detect a cycle in a linked list

```
class Solution {
public:
  bool hasCycle(ListNode *head) {
    ListNode*slow=head;
    ListNode*fast=head;
    while(fast && fast->next) {
        slow=slow->next;
        fast=fast->next->next;
        if(fast==slow) {
            return true;
        }
    }
    return false;
}
```



#### 7) Rotate a list

class Solution {

```
public:
  ListNode* rotateRight(ListNode* head, int k) {
     if (!head \parallel !head->next \parallel k == 0) return head;
     int length = 1;
     ListNode* tail = head;
     while (tail->next) {
        tail = tail->next;
       length++;
     }
     k = k \% length;
     if (k == 0) return head;
     ListNode* newTail = head;
     for (int i = 0; i < length - k - 1; i++) {
       newTail = newTail->next;
     }
     ListNode* newHead = newTail->next;
     newTail->next = nullptr;
     tail->next = head;
     return newHead;
  }
};

■ Description | ⑤ Accepted × | □ Editorial | △ Solutions | ⑤ Submissions
                                                                                        (ĝ)
      Status ∨
                             Language ∨ Runtime
                                                       Memory
                                                                    Notes
      Accepted
                             C++
                                         (3) 0 ms
                                                       @ 16.5 MB
      a minute ago
```

#### 8) Sort list

```
class Solution {
public:
  ListNode* findMiddle(ListNode* head) {
     ListNode* slow = head;
     ListNode* fast = head->next;
     while (fast && fast->next) {
       slow = slow->next;
       fast = fast->next->next;
     }
    return slow;
  }
  ListNode* merge(ListNode* left, ListNode* right) {
     ListNode dummy(0);
    ListNode* tail = &dummy;
    while (left && right) {
       if (left->val <= right->val) {
         tail->next = left;
         left = left->next;
       } else {
         tail->next = right;
         right = right->next;
       tail = tail->next;
     }
     tail->next = left ? left : right;
     return dummy.next;
  ListNode* sortList(ListNode* head) {
```

```
if (!head || !head->next) return head
     ListNode* mid = findMiddle(head);
     ListNode* rightHalf = mid->next;
     mid->next = nullptr;
     ListNode* leftSorted = sortList(head);
     ListNode* rightSorted = sortList(rightHalf);
     return merge(leftSorted, rightSorted);
  }
};
 ■ Description | ⑤ Accepted × | □ Editorial | △ Solutions ⑤ Submissions
                                                                                 Status ∨
                           Language ∨ Runtime
                                                   Memory
                                                                Notes
                                                                                   (ģ)
     Accepted
                          C++
                                      (3) 18 ms
                                                   a few seconds ago
```

#### 9) Merge k sorted lists

```
#include <queue>
class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val;
        }
    };
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;
        for (ListNode* list : lists) {
            if (list) minHeap.push(list);
        }
}
```

```
ListNode dummy(0);
     ListNode* tail = &dummy;
     while (!minHeap.empty()) {
       ListNode* smallest = minHeap.top();
       minHeap.pop();
       tail->next = smallest;
       tail = tail->next;
       if (smallest->next) {
          minHeap.push(smallest->next);
     }
     return dummy.next;
  }
};

■ Description | ⑤ Accepted × | □ Editorial | △ Solutions | ⑤ Submissions
      Status V
                           Language V Runtime
                                                                                    (ĝ)
                                                    Memory
                                                                 Notes
      Accepted
                           C++
                                                    @ 18.4 MB
                                       (3) 2 ms
      a few seconds ago
```