# Assignment-3

**Student Name:** Sameer **UID:** 22BCS15631

**Branch:** Computer Science & Engineering **Section/Group:** IOT-614/B

**Semester:** 6th **Date of Performance:** 14/02/2025

**Subject Name:** Advanced Programming Lab-2 **Subject Code:** 22CSP-351

**Q.1. Print Linked List**

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

**Code:**

class Solution

{

public:

void printList(Node \*head)

{

while (head != NULL)

{

cout << head->data;

if (head->next) cout << " ";

head = head->next;

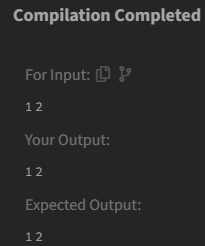
}

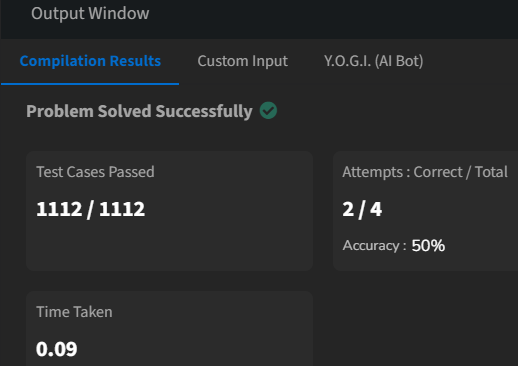
}

};

**\**

**Output:**

****

****

**Q.2. Remove Duplicates from Sorted List**

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 || head->next == NULL)

{

return head ;

}

ListNode\* temp = head ;

ListNode\* after = head->next ;

while(after != NULL)

{

if ((temp->val) == (after->val))

{

temp->next = after->next ;

after->next = NULL ;

delete after ;

if (temp->next == NULL)

{

return head ;

}

}

else

{

temp = temp->next ;

}

after = temp->next ;

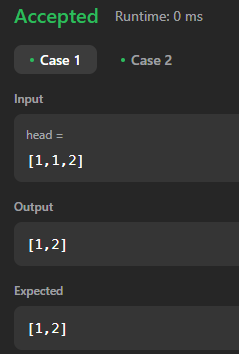
}

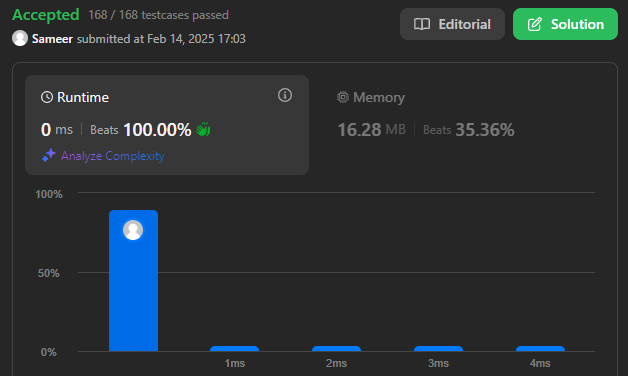
return head ;

}

};

**Output:**

****

****

**Q.3. Reverse Linked List**

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\* curr = head ;

ListNode\* prev = NULL ;

ListNode\* next = NULL ;

while(curr != NULL)

{

next = curr->next ;

curr->next = prev ;

prev = curr ;

curr = next ;

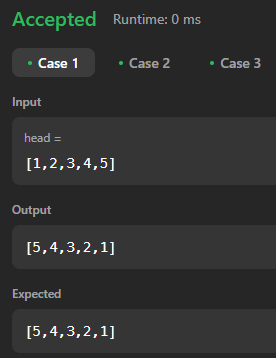
}

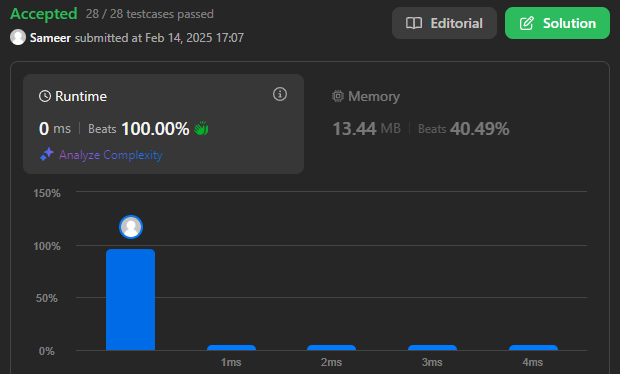
return prev ;

}

};

**Output:**





**Q.4. Delete the Middle Node of a Linked List**

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

**Code:**

class Solution

{

public:

ListNode\* deleteMiddle(ListNode\* head)

{

if (!head || !head->next)

{

return nullptr;

}

ListNode\* slow = head;

ListNode\* fast = head;

ListNode\* prev = nullptr;

while (fast && fast->next)

{

fast = fast->next->next;

prev = slow;

slow = slow->next;

}

prev->next = slow->next;

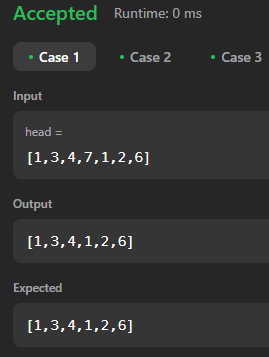
delete slow;

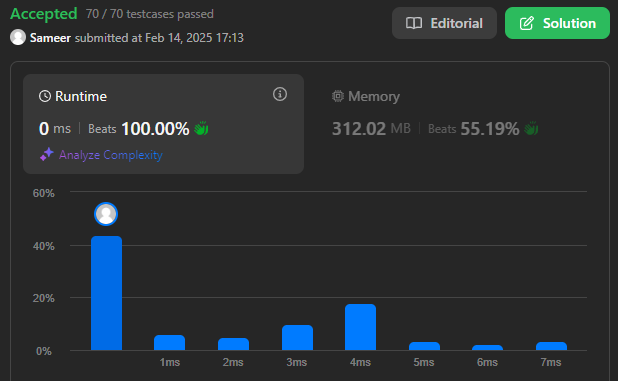
return head;

}

};

**Output:**

****

****

**Q.5 Merge Two Sorted Lists**

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\* solve(ListNode\* list1, ListNode\* list2)

{

if (list1->next == NULL)

{

list1->next = list2 ;

return list1 ;

}

ListNode\* curr1 = list1 ;

ListNode\* next1 = curr1->next ;

ListNode\* curr2 = list2 ;

ListNode\* next2 = curr2->next ;

while (curr2 != NULL && next1 != NULL)

{

if (curr2->val >= curr1->val && curr2->val <= next1->val)

{

curr1->next = curr2 ;

next2 = curr2->next ;

curr2->next = next1 ;

curr1 = curr2 ;

curr2 = next2 ;

}

else

{

curr1 = next1 ;

next1 = next1->next ;

if (next1 == NULL)

{

curr1->next = curr2 ;

return list1 ;

}

}

}

return list1 ;

}

ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2)

{

if (list1 == NULL)

{

return list2 ;

}

if (list2 == NULL)

{

return list1 ;

}

if (list1->val <= list2->val)

{

return solve(list1, list2) ;

}

else

{

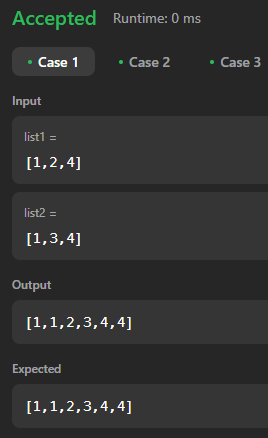
return solve(list2, list1) ;

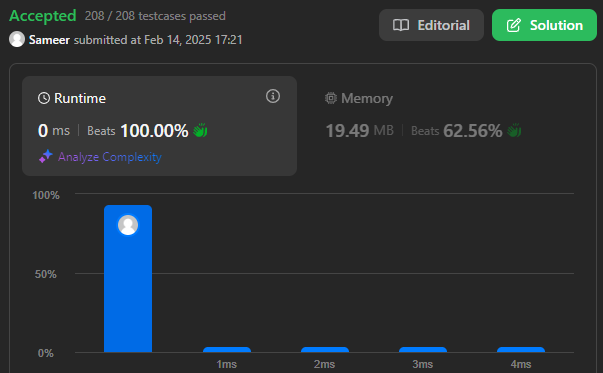
}

}

};

**Output:**





**Q.6. Remove Duplicates from Sorted List II**

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.

**Code:**

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* dummy = new ListNode(0);

dummy->next = head;

ListNode\* prev = dummy;

ListNode\* current = head;

while (current) {

bool hasDuplicates = false;

while (current->next && current->val == current->next->val) {

current = current->next;

hasDuplicates = true;

}

if (hasDuplicates) {

prev->next = current->next;

} else {

prev = prev->next;

}

current = current->next;

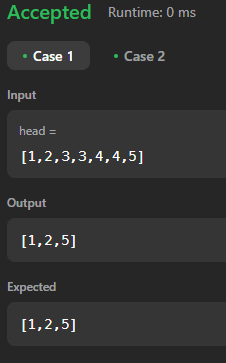
}

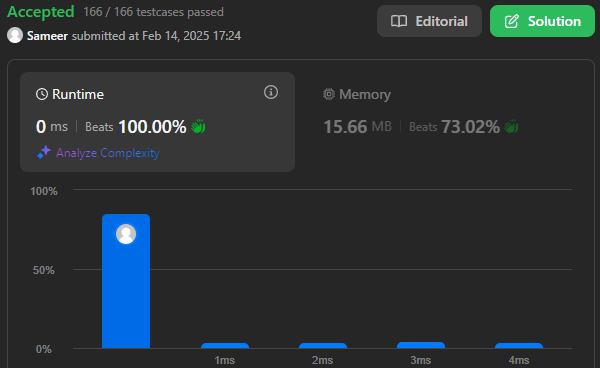
return dummy->next;

}

};

**Output:**

****

****

**Q.7 Linked List Cycle**

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

**Code:**

class Solution

{

public:

bool hasCycle(ListNode \*head)

{

if (head == NULL || head->next == NULL)

{

return false ;

}

ListNode\* slow = head ;

ListNode\* fast = head ;

while (slow != NULL && fast != NULL && fast->next != NULL)

{

slow = slow->next ;

fast = fast->next->next ;

if (slow == fast)

{

return true ;

}

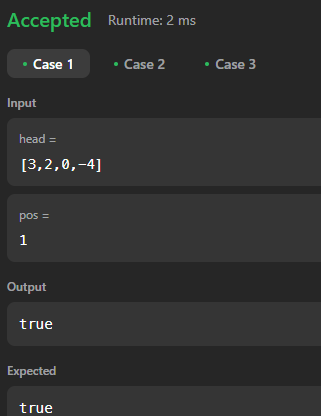
}

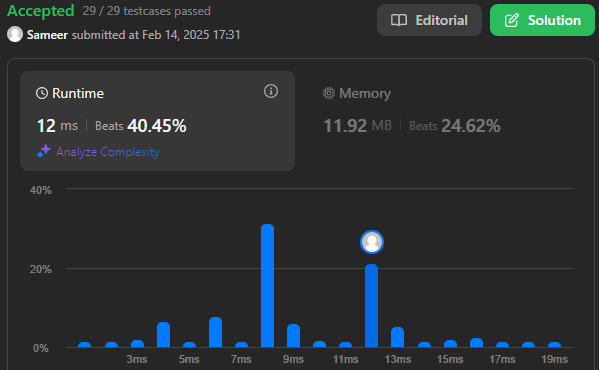
return false ;

}

};

**Output:**

****

****

**Q.8 Reverse Linked List II**

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.

**Code:**

class Solution {

public:

ListNode\* reverseBetween(ListNode\* head, int left, int right)

{

if (!head || left == right) return head;

ListNode\* dummy = new ListNode(0);

dummy->next = head;

ListNode\* prev = dummy;

for (int i = 1; i < left; ++i)

{

prev = prev->next;

}

ListNode\* start = prev->next;

ListNode\* then = start->next;

for (int i = left; i < right; ++i)

{

start->next = then->next;

then->next = prev->next;

prev->next = then;

then = start->next;

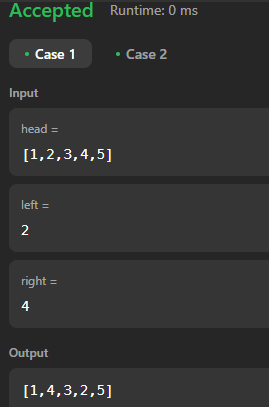
}

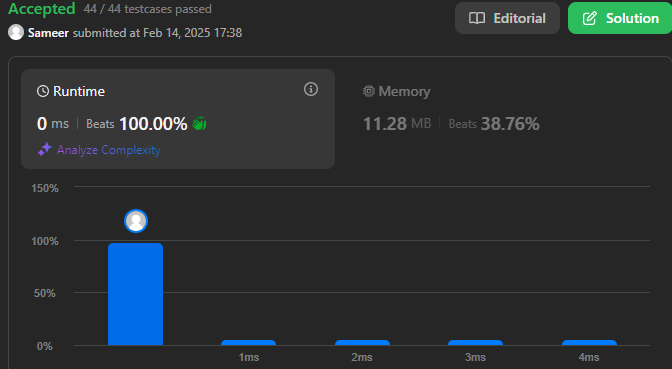
return dummy->next;

}

};

**Output:**

****

****

**5**

**Q.9. Rotate List**

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 || !head->next || k == 0) return head;

ListNode\* curr = head;

int length = 1;

while (curr->next)

{

curr = curr->next;

length++;

}

k = k % length;

if (k == 0) return head;

curr->next = head; // Form a cycle

for (int i = 0; i < length - k; ++i)

{

curr = curr->next;

}

ListNode\* newHead = curr->next;

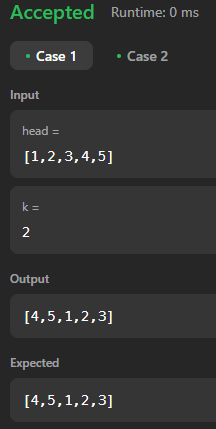
curr->next = nullptr;

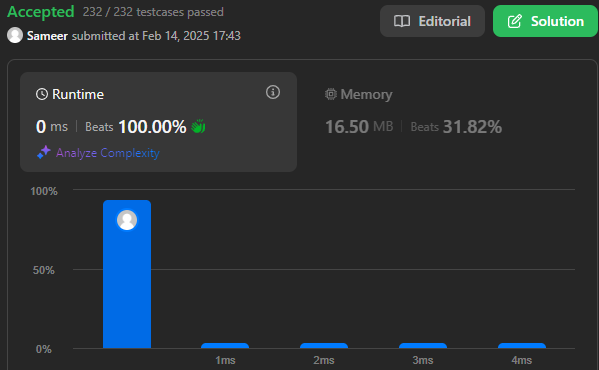
return newHead;

}

};

**Output:**

****

****

**Q.10. Sort List**

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 || !head->next)

return head;

ListNode\* slow = head;

ListNode\* fast = head;

ListNode\* prev = nullptr;

while (fast && fast->next)

{

prev = slow;

slow = slow->next;

fast = fast->next->next;

}

prev->next = nullptr;

ListNode\* l1 = sortList(head);

ListNode\* l2 = sortList(slow);

return merge(l1, l2);

}

ListNode\* merge(ListNode\* l1, ListNode\* l2)

{

ListNode dummy(0);

ListNode\* curr = &dummy;

while (l1 && l2)

{

if (l1->val < l2->val)

{

curr->next = l1;

l1 = l1->next;

}

else

{

curr->next = l2;

l2 = l2->next;

}

curr = curr->next;

}

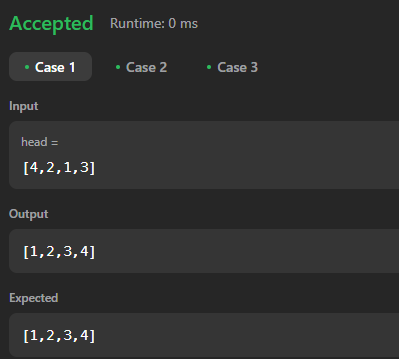
curr->next = l1 ? l1 : l2;

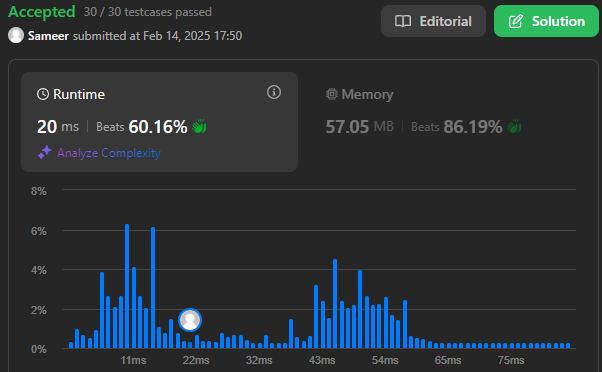
return dummy.next;

}

};

**Output:**

****

****

**Q.11 Linked List Cycle II**

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

**Code:**

class Solution {

public:

ListNode \*detectCycle(ListNode \*head)

{

if (!head || !head->next)

return nullptr;

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;

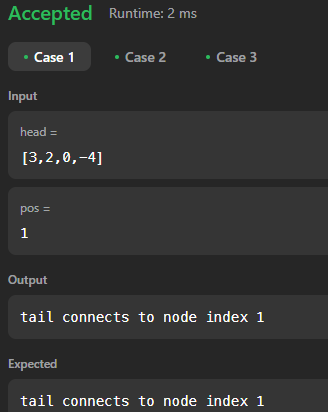
}

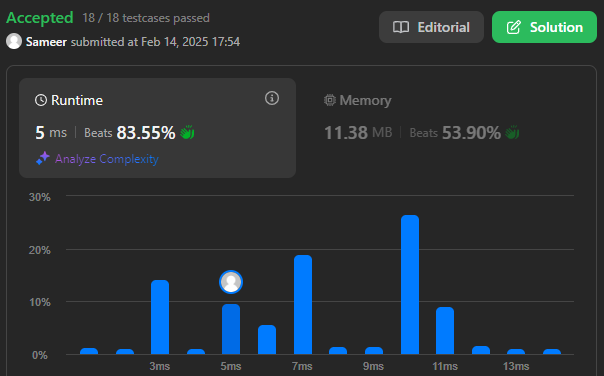
return slow;

}

}

**Output:**

****

****