**Experiment 3**

**Student Name:** Armaan Siag **UID:** 22BET10322

**Branch:** Information Technology **Section/Group:** 22BET\_IOT-703/A

**Semester:** 6th **Subject Code:** 22ITP-351

**Problem 1**

**Aim:**

Print linked list.

**Code:**

class Solution {

public:

// Function to display the elements of a linked list in same line

void printList(Node \*head) {

// iterating through the linked list

while (head != nullptr) {

// printing the data of current node

cout << head->data << " ";

// moving to the next node

head = head->next;

}

}

};

**Output:**

A screenshot of a computer

Description automatically generated

**Problem 2**

**Aim:**

Remove duplicates from a sorted list.

**Code:**

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* res = head;

while (head && head->next) {

if (head->val == head->next->val) {

head->next = head->next->next;

} else {

head = head->next;

}

}

return res;

}

};

**Output:**

A screenshot of a phone

Description automatically generated A screenshot of a phone

Description automatically generated

Test Case 1 Test Case 2

**Problem 3**

**Aim:**

Reverse a linked list.

**Code:**

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode\* node = nullptr;

while (head != nullptr) {

ListNode\* temp = head->next;

head->next = node;

node = head;

head = temp;

}

return node;

}

};

**Output:**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Case 1 Case 2 Case 3

**Problem 4**

**Aim:**

Delete middle node of a list.

**Code:**

class Solution {

public:

ListNode\* deleteMiddle(ListNode\* head) {

if(!head || !head->next) return nullptr;

ListNode\*slow = head;

ListNode\*fast = head->next->next;

while(fast !=nullptr && fast->next != nullptr){

fast = fast->next->next;

slow = slow->next;

}

ListNode\*delnode = slow->next;

slow->next =slow->next->next;

delete delnode;

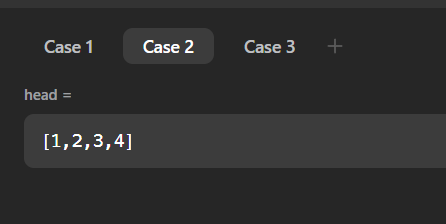
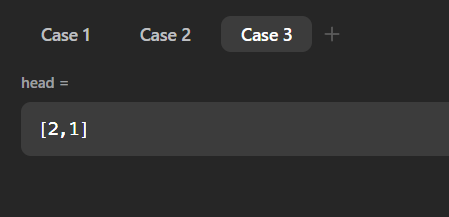
return head;

}

};

**Output:**

**A screenshot of a computer

Description automatically generated  **

Case 1 Case 2 Case 3

**Problem 5**

**Aim:**

Merge two sorted linked lists.

**Code:**

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

if (l1 == NULL) return l2;

if (l2 == NULL) return l1;

if (l1->val > l2->val) swap(l1, l2);

ListNode\* ans = l1;

while (l1 != NULL && l2 != NULL) {

ListNode\* temp = NULL;

while (l1 != NULL && l1->val <= l2->val) {

temp = l1;

l1 = l1->next;

}

if (temp != NULL) temp->next = l2;

swap(l1, l2);

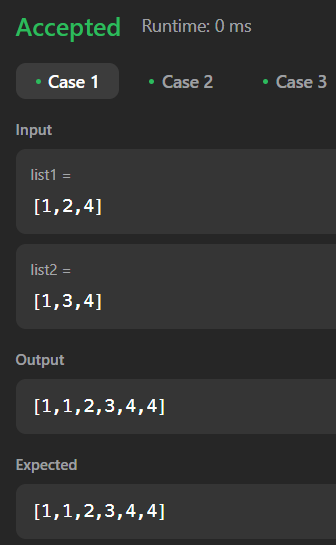
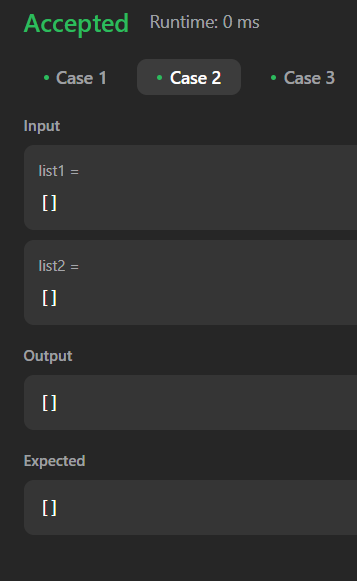
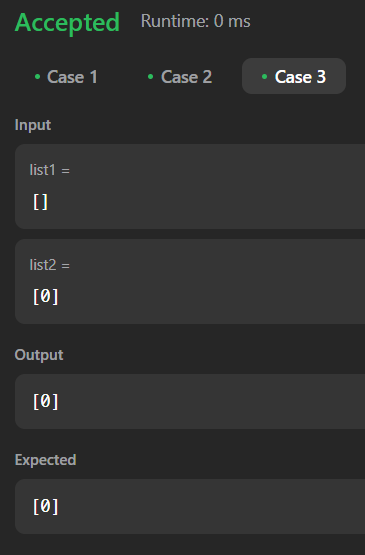
}

return ans;

}

};

**Output:**

Case 1 Case 2 Case 3

**Problem 6**

**Aim:**

Remove duplicates from sorted lists 2.

**Code:**

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* res = head;

while (head && head->next) {

if (head->val == head->next->val) {

head->next = head->next->next;

} else {

head = head->next;

}

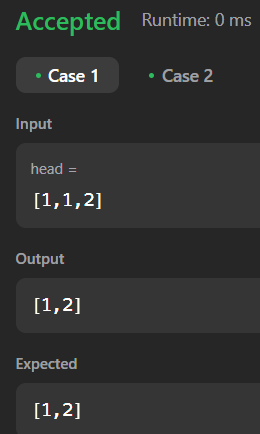
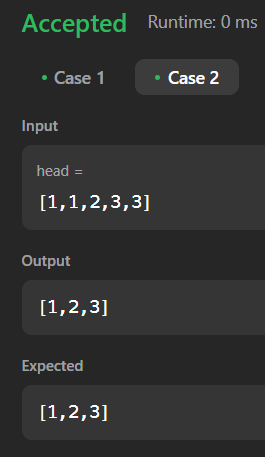
}

return res;

}

};

**Output:**

Case 1 Case 2

**Problem 7**

**Aim:**

Detect a cycle in a linked list.

**Code:**

class Solution {

public:

bool hasCycle(ListNode \*head) {

ListNode\* fast = head;

ListNode\* slow = head;

while (fast != nullptr && fast->next != nullptr) {

fast = fast->next->next;

slow = slow->next;

if (fast == slow) {

return true;

}

}

return false;

}

};

**Output:**

A screenshot of a computer

Description automatically generated A screenshot of a black screen

Description automatically generated A screenshot of a black screen

Description automatically generated

Case 1 Case 2 Case 3

**Problem 8**

**Aim:**

Reverse linked list 2

**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 = 0; i < left - 1; i++) {

prev = prev->next;

}

ListNode\* cur = prev->next;

for (int i = 0; i < right - left; i++) {

ListNode\* temp = cur->next;

cur->next = temp->next;

temp->next = prev->next;

prev->next = temp;

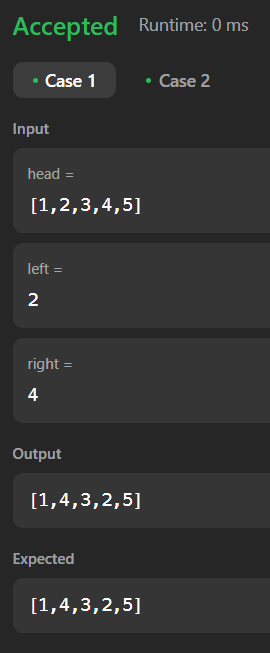
}

return dummy->next;

}

};

**Output:**

Case 1 Case 2

**Problem 9**

**Aim:**

Rotate a list.

**Code:**

class Solution {

public:

ListNode\* rotateRight(ListNode\* head, int k) {

if (!head || !head->next || k == 0)

return head;

ListNode\* tail;

int length = 1;

for (tail = head; tail->next; tail = tail->next)

++length;

tail->next = head;

const int t = length - k % length;

for (int i = 0; i < t; ++i)

tail = tail->next;

ListNode\* newHead = tail->next;

tail->next = nullptr;

return newHead;

}

};

**Output:**

A screenshot of a black screen

Description automatically generated A screenshot of a black screen

Description automatically generated

Case 1 Case 2

**Problem 10**

**Aim:**

Merge k sorted lists.

**Code:**

class Solution {

public:

ListNode\* mergeKLists(vector<ListNode\*>& lists) {

int n=lists.size();

priority\_queue<int,vector<int>,greater<int>>q;

for(int i=0;i<n;i++){

ListNode\* temp=lists[i];

while(temp!=NULL){

q.push(temp->val);

temp=temp->next;

}

}

if(!q.empty()){

ListNode\* ans=new ListNode(q.top());

q.pop();

ListNode\* tempp=ans;

while(!q.empty()){

int temp=q.top();

q.pop();

tempp->next=new ListNode(temp);

if(tempp->next!=NULL) tempp=tempp->next;

}

return ans;

}

else return NULL;

}

};

**Output:**

A screenshot of a black screen

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a black screen

Description automatically generated

Case 1 Case 2 Case 3

**Problem 11**

**Aim:**

Sort List.

**Code:**

#include <iostream>

using namespace std;

//takesoumen collection

class Solution {

public:

ListNode\* sortList(ListNode\* head) {

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

// Find the middle using slow and fast pointers

ListNode\* slow = head;

ListNode\* fast = head->next;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

}

ListNode\* mid = slow->next;

slow->next = nullptr;

// Recursively split and merge

ListNode\* left = sortList(head);

ListNode\* right = sortList(mid);

return merge(left, right);

}

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

ListNode dummy(0);

ListNode\* tail = &dummy;

while (l1 && l2) {

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

tail->next = l1;

l1 = l1->next;

} else {

tail->next = l2;

l2 = l2->next;

}

tail = tail->next;

}

tail->next = l1 ? l1 : l2;

return dummy.next;

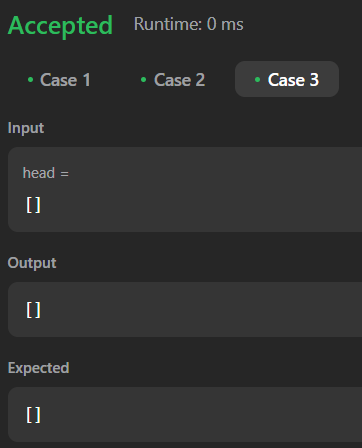
}

};

**Output:**

A screenshot of a black screen

Description automatically generated A screenshot of a computer

Description automatically generated 

Case 1 Case 2 Case 3