**Waleed Afzal**

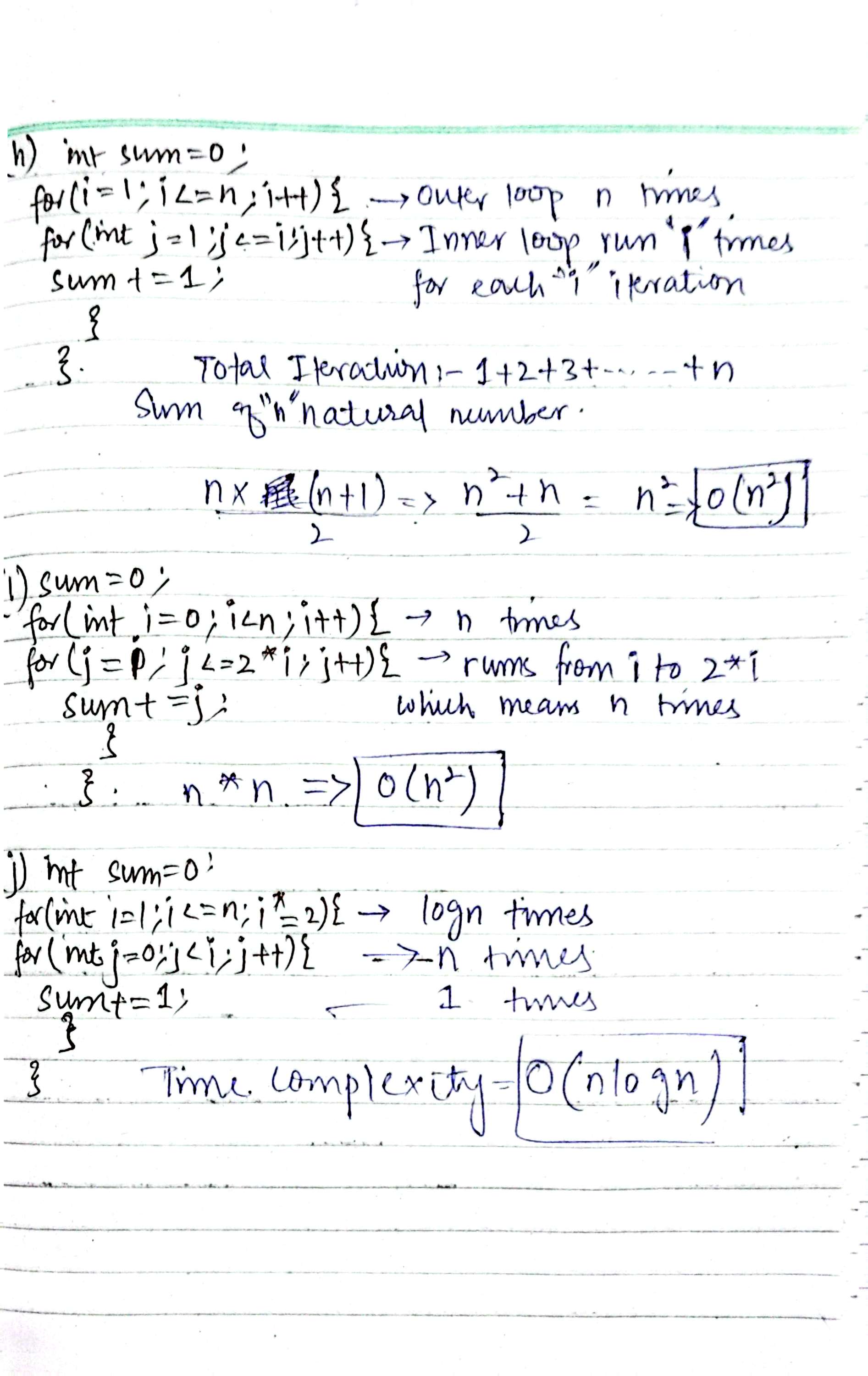
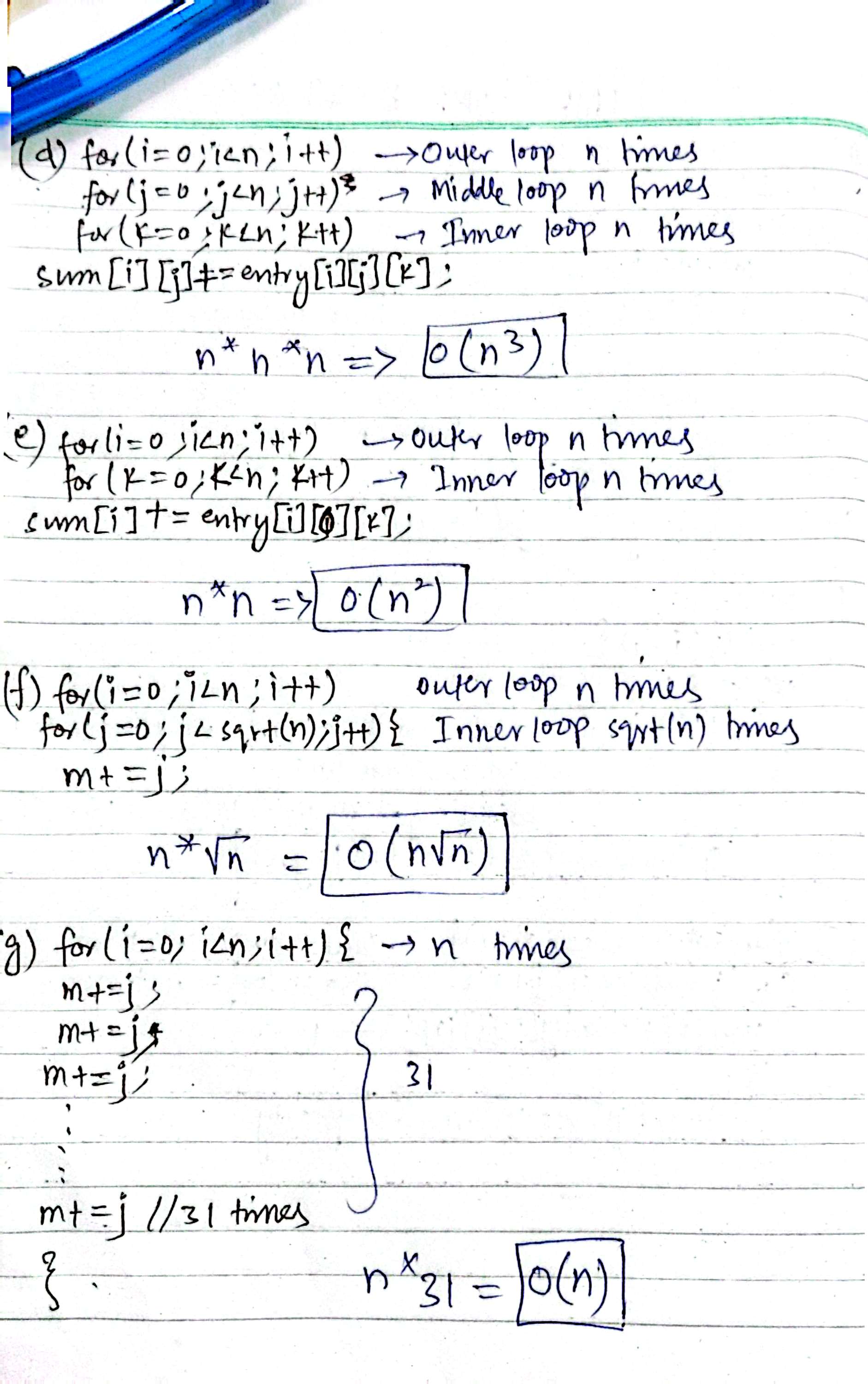
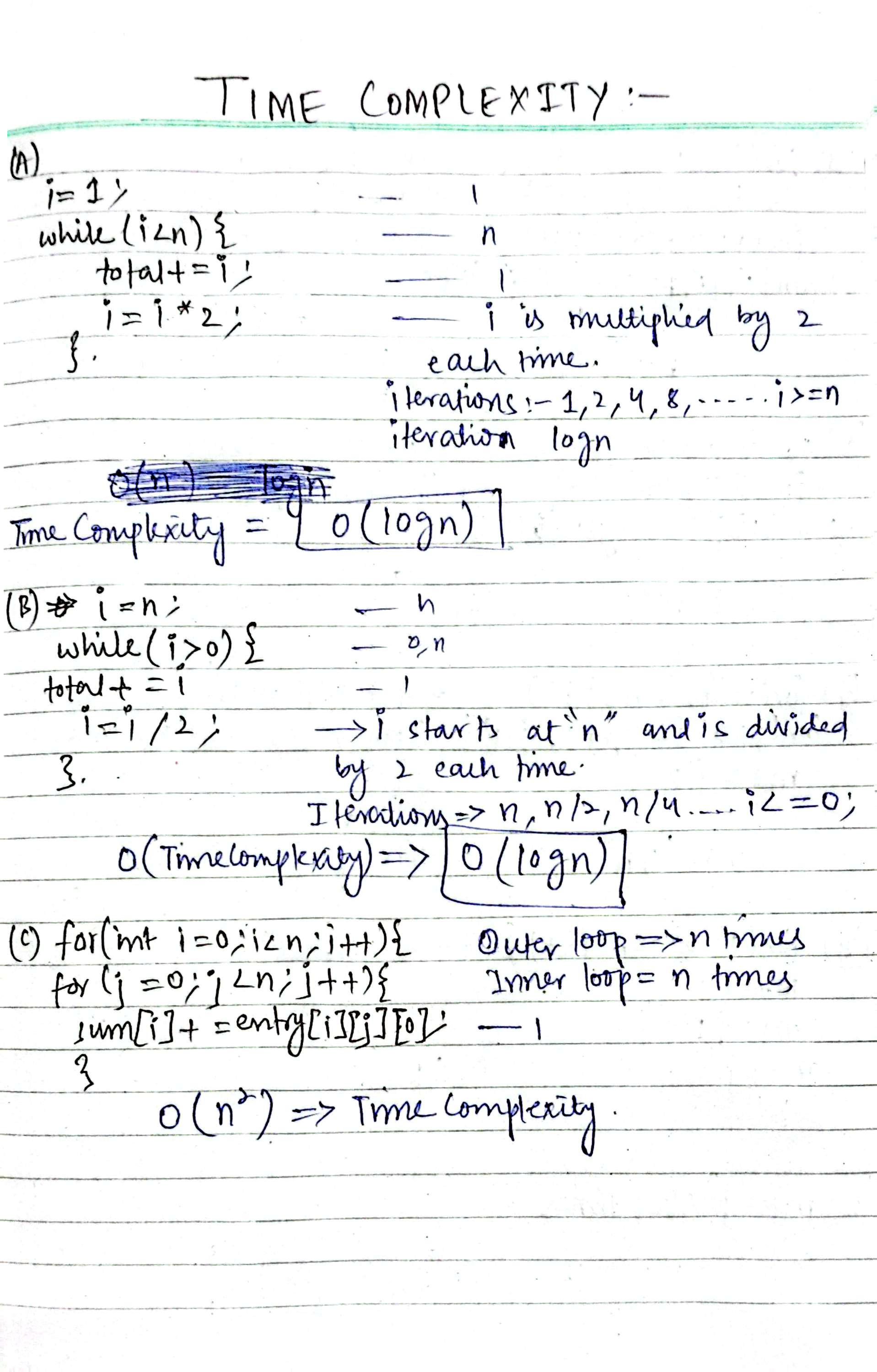
**23p-0566**

**3D – Data Structure**

**Assignment #01:**

**14-9-2024**

**Time Complexity Question (A-J):**

****

**Task2:**#include<iostream>

#include<string>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class LinkedList {

private:

Node\* head;

public:

void insertAtBeginning(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

}

void insertInMiddle(int data, int key) {

Node\* newNode = new Node(data);

Node\* curr = head;

int index = 1;

while (curr->next != NULL) {

if (curr->data == key)

{

break;

}

index = index + 1;

curr = curr->next;

}

newNode->next = curr->next;

curr->next = newNode;

}

void insertNodeAtEnd(int val) {

Node\* newNode = new Node(val);

Node\* curr = head;

if (head == NULL) {

head = newNode;

return;

}

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = newNode;

}

bool deleteFirstNode() {

Node\* curr = head;

if (head == NULL) {

//cout << "There is No Node" << endl;

return false;

}

else {

head = head->next;

return true;

}

}

bool deleteNode(int key) {

Node\* curr = head;

int index = 0;

while (curr->next != NULL) {

if (curr->data == key) {

break;

}

index++;

curr = curr->next;

}

curr = head;

for (int i = 1;i < index;i++) {

curr->next = curr->next->next;

}

curr = curr->next->next;

// curr=NULL;

return true;

}

bool deleteLastNode() {

Node\* curr = head;

if (head == NULL) {

cout << "List is Empty" << endl;

return false;

}

while (curr->next->next != NULL) {

curr = curr->next;

}

curr->next = NULL;

// curr->next->next=NULL;

return true;

}

bool search(int val) {

Node\* curr = head;

int counter = 1;

while (curr != NULL) {

if (curr->data == val) {

break;

}

curr = curr->next;

counter++;

}

cout << "Value " << val << " Found at Position " << counter << endl;

return true;

}

void display() {

Node\* curr = head;

while (curr != NULL) {

cout << curr->data << " ";

curr = curr->next;

}

cout << endl;

}

};

int main() {

LinkedList L1;

L1.insertNodeAtEnd(0);

L1.insertNodeAtEnd(1);

L1.insertNodeAtEnd(2);

L1.insertNodeAtEnd(3);

L1.insertInMiddle(4, 3);

L1.insertNodeAtEnd(5);

L1.deleteNode(2);

L1.deleteLastNode();

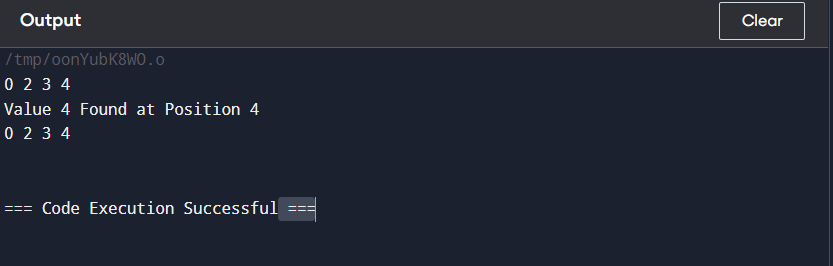
L1.display();

L1.search(4);

L1.display();

}

**OUTPUT:**

****

**Task3:**

#include<iostream>

#include<string>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node\* prev;

Node(int val)

{

data = val;

next = NULL;

prev = NULL;

}

};

class DoublyList {

private:

Node\* head;

public:

DoublyList() {

head = NULL;

}

void insertAtHead(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

newNode->prev = NULL;

if (head != NULL) {

head->prev = newNode;

}

head = newNode;

}

void insertAtTail(int val) {

Node\* newNode = new Node(val);

if (head == NULL) {

head = newNode;

return;

}

Node\* curr = head;

while (curr->next != NULL) {

curr = curr->next;

}

curr->next = newNode;

newNode->prev = curr;

//newNode->next = NULL;

}

void insertNodeinMiddle(int val, int key) {

Node\* newNode = new Node(val);

Node\* curr = head;

Node\* temp = head;

while (curr != NULL) {

if (curr->data == key) {

newNode->prev = curr;

newNode->next = curr->next;

curr->next = newNode;

newNode->next->prev = newNode;

}

curr = curr->next;

}

}

void removeFirst() {

if (head == NULL) {

cout << "List is empty!" << endl;

return;

}

head = head->next;

if (head != NULL) {

head->prev = NULL;

}

head->prev = NULL;

}

void removeLast() {

Node\* curr = head;

if (head == NULL) {

cout << "List is Empty" << endl;

}

if (head->next == NULL) {

delete head;

head = NULL;

return;

}

while (curr->next != NULL) {

curr = curr->next;

}

curr->prev->next = NULL; // Remove the last node from the list

delete curr; // Free the memory of the last node

}

bool search(int data) {

Node\* curr = head;

while (curr != NULL) {

if (curr->data == data) {

return true;

}

curr = curr->next;

}

return false;

}

void printList() {

Node\* curr = head;

while (curr != NULL) {

cout << curr->data << " ";

curr = curr->next;

}

cout << endl;

}

void removeAtPosition(int key) {

Node\* curr = head;

int counter = 1;

while (curr != NULL) {

if (curr->data == key) {

curr->prev->next = curr->next;

curr->next->prev = curr->prev;

}

curr = curr->next;

counter++;

}

}

};

class CircularList {

private:

Node\* head;

public:

CircularList() {

head = NULL;

}

void insertNodeAtBeginning(int data) {

Node\* newNode = new Node(data);

Node\* curr = head;

if (head == NULL) {

head = newNode;

newNode->next = head;

}

else {

while (curr->next != head) {

curr = curr->next;

}

newNode->next = head;

curr->next = newNode;

head = newNode;

}

}

void insertNodeAtTail(int val) {

Node\* newNode = new Node(val);

Node\* curr = head;

if (head == NULL) {

head = newNode;

}

else {

do {

curr = curr->next;

} while (curr->next != head);

curr->next = newNode;

newNode->next = head;

}

}

void display() {

Node\* curr = head;

do {

cout << curr->data << " ";

curr = curr->next;

} while (curr != head);

}

bool deleteFirstNode() {

Node\* curr = head;

Node\* temp = head;

if (head == NULL) { //if no node

return false;

} //for 1 node

else if (head->next == NULL) {

delete head;

head = NULL;

}

do {

curr = curr->next;

} while (curr->next != head);

head = head->next;

if (curr == head) {

curr->next = NULL;

delete temp;

}

else {

curr->next = head;

delete temp;

}

return true;

}

bool deleteLastNode() {

Node\* curr = head;

Node\* temp = head;

if (head == NULL) { //if no node

return false;

}

if (head->next == NULL) { // if only 1 node

head = NULL;

return true;

}

//if 2 nodes

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

while (curr->next != head) {

curr = curr->next;

}

head->next = NULL;

delete curr;

return true;

}

else { //more than 2 nodes

do {

curr = curr->next;

} while (curr->next != head);

// Finding the second-to-last node

while (temp->next != curr) {

temp = temp->next;

}

// Updating the second-to-last node to point to head

temp->next = head;

delete curr; // Deleting the last node

return true;

}

}

bool deleteNode(int key) {

Node\* curr = head;

Node\* temp = head;

if (head == NULL) { //list empty

return false;

}

if (curr->data == key && curr->next == NULL) {

delete head;

}

if (curr->data == key) {

if (deleteFirstNode()) {

return true;

}

return false;

}

while (curr->next != head) {

curr = curr->next;

}

if (curr->data == key) {

if (deleteLastNode()) {

return true;

}

return false;

}

curr = head;

while (curr->next != head) {

if (curr->data == key) {

if (curr->next != head) {

temp->next = curr->next;

delete curr;

return true;

}

else {

if (curr->next == head) {

temp = head;

while (temp->next != curr) {

temp = temp->next;

}

temp->next = head;

delete curr;

return true;

}

// return true;

}

}

else {

temp = curr;

curr = curr->next;

}

}

return true;

}

bool search(int key) {

if (head == NULL) {

return false; // List is empty

}

Node\* curr = head;

do {

if (curr->data == key) { // Correct comparison

return true; // Value found

}

curr = curr->next;

} while (curr != head); // Loop until we return to the head

return false;

}

};

int main() {

DoublyList L1;

L1.insertAtTail(10);

L1.insertAtTail(20);

L1.insertAtTail(30);

L1.insertAtTail(40);

L1.printList();

L1.insertNodeinMiddle(25, 30);

L1.printList();

L1.removeAtPosition(25);

L1.printList();

if (L1.search(20)) {

cout << "Data available in List\n";

};

L1.printList();

cout << "\n===> CIRCULAR LINKEDLIST <========\n\n";

CircularList CL1;

CL1.insertNodeAtBeginning(10);

CL1.insertNodeAtBeginning(20);

CL1.insertNodeAtBeginning(30);

CL1.display();

cout << endl;

CL1.insertNodeAtTail(700);

CL1.insertNodeAtTail(800);

CL1.insertNodeAtTail(900);

CL1.display();

cout << endl;

CL1.deleteFirstNode();

CL1.display();

cout << endl;

CL1.deleteLastNode();

CL1.insertNodeAtTail(900);

CL1.display();

CL1.deleteNode(800);

cout << endl;

CL1.display();

if (CL1.search(7000) == true) {

cout << "\nValue Found\n";

}

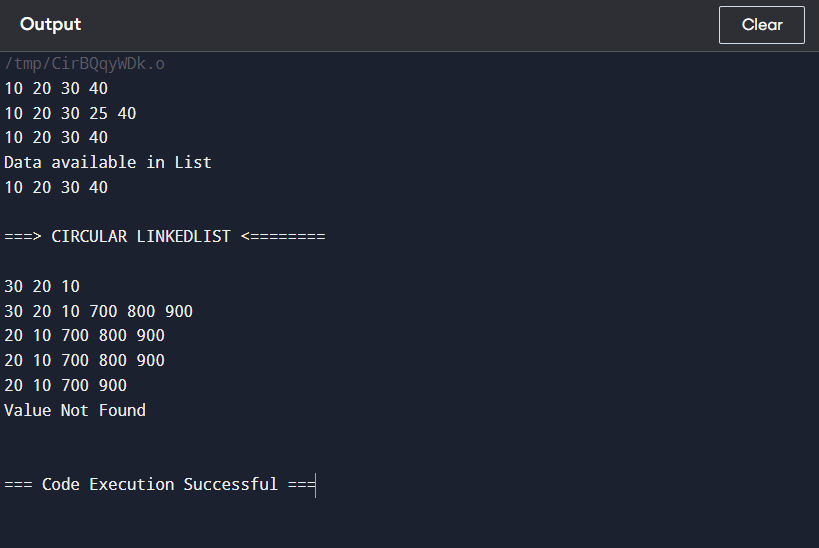
else {

cout << "\nValue Not Found\n";

}

}

**OUTPUT:**



**Task4:**

#include<iostream>

#include<string>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class LinkedList {

private:

Node\* head;

public:

void insertAtBeginning(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

}

void insertInMiddle(int data, int key) {

Node\* newNode = new Node(data);

Node\* curr = head;

int index = 1;

while (curr->next != NULL) {

if (curr->data == key)

{

break;

}

index = index + 1;

curr = curr->next;

}

newNode->next = curr->next;

curr->next = newNode;

}

void insertNodeAtEnd(int val) {

Node\* newNode = new Node(val);

Node\* curr = head;

if (head == NULL) {

head = newNode;

return;

}

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = newNode;

}

bool deleteFirstNode() {

Node\* curr = head;

if (head == NULL) {

//cout << "There is No Node" << endl;

return false;

}

else {

head = head->next;

return true;

}

}

bool deleteNode(int key) {

Node\* curr = head;

int index = 0;

while (curr->next != NULL) {

if (curr->data == key) {

break;

}

index++;

curr = curr->next;

}

curr = head;

for (int i = 1;i < index;i++) {

curr->next = curr->next->next;

}

curr = curr->next->next;

// curr=NULL;

return true;

}

bool deleteLastNode() {

Node\* curr = head;

if (head == NULL) {

cout << "List is Empty" << endl;

return false;

}

while (curr->next->next != NULL) {

curr = curr->next;

}

curr->next = NULL;

// curr->next->next=NULL;

return true;

}

bool search(int val) {

Node\* curr = head;

int counter = 1;

while (curr != NULL) {

if (curr->data == val) {

break;

}

curr = curr->next;

counter++;

}

cout << "Value " << val << " Found at Position " << counter << endl;

return true;

}

void reverseList() {

Node\* curr = NULL;

Node\* temp = head;

Node\* temp1 = NULL;

while (temp != NULL) {

temp1 = temp->next;

temp->next = curr;

curr=temp;

temp=temp1;

// curr = curr->next;

}

head=curr;

cout << "List Reversed: " << endl;

}

void display() {

Node\* curr = head;

while (curr != NULL) {

cout << curr->data << " ";

curr = curr->next;

}

cout << endl;

}

};

int main() {

LinkedList L1;

L1.insertNodeAtEnd(0);

L1.insertNodeAtEnd(1);

L1.insertNodeAtEnd(2);

L1.insertNodeAtEnd(3);

L1.display();

L1.insertInMiddle(4, 3);

L1.display();

L1.insertNodeAtEnd(5);

L1.display();

L1.deleteNode(2);

L1.display();

L1.deleteLastNode();

L1.display();

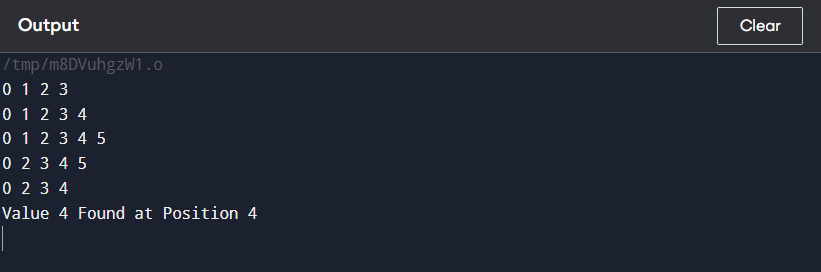
L1.search(4);

L1.reverseList();

L1.display();

}

**OUTPUT**

****

**Task5:**

#include<iostream>

#include<string>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node\* prev;

Node(int val)

{

data = val;

next = NULL;

prev = NULL;

}

};

class DoublyList {

private:

Node\* head;

public:

DoublyList() {

head = NULL;

}

void insertAtHead(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

newNode->prev = NULL;

if (head != NULL) {

head->prev = newNode;

}

head = newNode;

}

void insertAtTail(int val) {

Node\* newNode = new Node(val);

if (head == NULL) {

head = newNode;

return;

}

Node\* curr = head;

while (curr->next != NULL) {

curr = curr->next;

}

curr->next = newNode;

newNode->prev = curr;

//newNode->next = NULL;

}

void insertNodeinMiddle(int val, int key) {

Node\* newNode = new Node(val);

Node\* curr = head;

Node\* temp = head;

while (curr != NULL) {

if (curr->data == key) {

newNode->prev = curr;

newNode->next = curr->next;

curr->next = newNode;

newNode->next->prev = newNode;

}

curr = curr->next;

}

}

void removeFirst() {

if (head == NULL) {

cout << "List is empty!" << endl;

return;

}

head = head->next;

if (head != NULL) {

head->prev = NULL;

}

head->prev = NULL;

}

void removeLast() {

Node\* curr = head;

if (head == NULL) {

cout << "List is Empty" << endl;

}

if (head->next == NULL) {

delete head;

head = NULL;

return;

}

while (curr->next != NULL) {

curr = curr->next;

}

curr->prev->next = NULL; // Remove the last node from the list

delete curr; // Free the memory of the last node

}

bool search(int data) {

Node\* curr = head;

while (curr != NULL) {

if (curr->data == data) {

return true;

}

curr = curr->next;

}

return false;

}

void printList() {

Node\* curr = head;

while (curr != NULL) {

cout << curr->data << " ";

curr = curr->next;

}

cout << endl;

}

void deleteNode(int key) {

Node\* curr = head;

int counter = 1;

while (curr != NULL) {

if (curr->data == key) {

curr->prev->next = curr->next;

curr->next->prev = curr->prev;

}

curr = curr->next;

counter++;

}

}

void sorting() {

cout << "After Sorting:\n";

if (head == NULL) { // No node

cout << "No Data to Sort\n";

return;

}

if (head->next == NULL) { // Only one node

cout << "Cannot Sort only 1 Data\n";

return;

}

Node\* curr = head;

if (head->next->next == NULL) { // Two nodes

curr = curr->next;

if (head->data > curr->data) {

// Swap the two nodes

curr->next = head;

head->prev = curr;

head->next = NULL;

curr->prev = NULL;

head = curr;

}

return;

}

// More than 2 nodes case

int count = 0;

curr = head;

// Counting total nodes in the list

while (curr != NULL) {

count++;

curr = curr->next;

}

// Bubble sort logic with node swaps

for (int i = 0; i < count - 1; i++) {

curr = head;

for (int j = 0; j < count - 1 - i; j++) {

Node\* temp = curr->next;

if (curr->data > temp->data) {

// Adjust the previous node of `urr if curr is not head

if (curr->prev != NULL) {

curr->prev->next = temp;

}

else {

head = temp; // Update head if curr is head node

}

// Adjust the next node of temp if temp is not the last node

if (temp->next != NULL) {

temp->next->prev = curr;

}

// Swap the nodes

curr->next = temp->next;

temp->prev = curr->prev;

temp->next = curr;

curr->prev = temp;

}

curr = temp;

}

}

}

};

int main() {

cout << "\n===> DOUBLY LINKEDLIST <========\n\n";

DoublyList L1;

L1.insertAtTail(5);

L1.insertAtTail(20);

L1.insertAtTail(10);

L1.insertAtTail(40);

L1.printList(); //(5,20,10,40)

L1.sorting(); //calling sorting

L1.printList(); //(5, 10, 20, 40)

L1.insertAtTail(9); //inserting 9 at the tail

L1.printList(); //(5,10,20,40,9)

L1.sorting(); //calling sorting function

L1.printList(); //after sorting (5,9,10,20,40)

L1.insertAtHead(2);

L1.insertAtHead(1);

L1.insertAtHead(50);

L1.printList();

L1.removeFirst();

L1.printList();

L1.removeLast();

L1.printList();

L1.insertAtTail(33);

L1.printList();

L1.deleteNode(9);

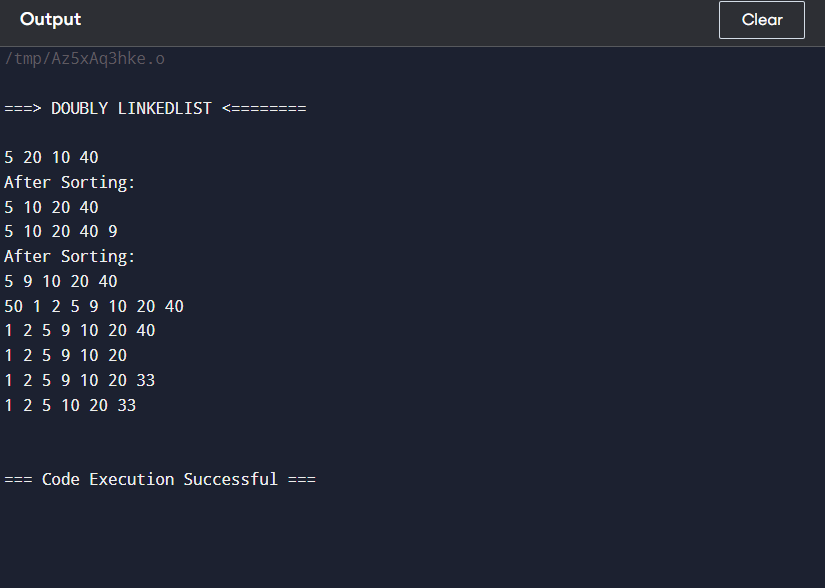
L1.printList();

system("pause");

return 0;

}

**OUTPUT:**

****

**Task6:**#include<iostream>

#include<string>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class LinkedList {

private:

Node\* head;

public:

void insertAtBeginning(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

}

void insertInMiddle(int data, int key) {

Node\* newNode = new Node(data);

Node\* curr = head;

int index = 1;

while (curr->next != NULL) {

if (curr->data == key)

{

break;

}

index = index + 1;

curr = curr->next;

}

newNode->next = curr->next;

curr->next = newNode;

}

void insertNodeAtEnd(int val) {

Node\* newNode = new Node(val);

Node\* curr = head;

if (head == NULL) {

head = newNode;

return;

}

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = newNode;

}

bool deleteFirstNode() {

Node\* curr = head;

if (head == NULL) {

//cout << "There is No Node" << endl;

return false;

}

else {

head = head->next;

return true;

}

}

bool deleteNode(int key) {

Node\* curr = head;

Node\* prev = NULL;

if (head == NULL) {

cout << "No Nodes Present\n";

return false;

}

// Handle deletion when the list has only 1 node

if (head->next == NULL) {

if (head->data == key) {

delete head;

head = NULL;

return true;

}

return false;

}

// for 2 nodes

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

curr = head;

while (curr != NULL) {

if (curr->data == key && curr == head) {

head = head->next;

delete curr;

return true;

}

if (curr->data == key && curr->next != NULL) {

curr = curr->next;

delete curr;

head->next = NULL;

return true;

}

curr = curr->next;

}

return false;

}

// for more than 2 nodes

curr = head;

while (curr != NULL) {

if (curr->data == key) {

if (curr == head) {

head = curr->next;

delete curr;

curr = head;

}

else {

prev->next = curr->next;

delete curr;

curr = prev->next;

}

return true;

}

prev = curr;

curr = curr->next;

}

return false;

}

bool deleteLastNode() {

Node\* curr = head;

if (head == NULL) {

cout << "List is Empty" << endl;

return false;

}

while (curr->next->next != NULL) {

curr = curr->next;

}

curr->next = NULL;

// curr->next->next=NULL;

return true;

}

bool search(int val) {

Node\* curr = head;

int counter = 1;

while (curr != NULL) {

if (curr->data == val) {

return true;

break;

}

curr = curr->next;

counter++;

}

// cout << "Value " << val << " Found at Position " << counter << endl;

return false;

}

void display() {

Node\* curr = head;

while (curr != NULL) {

cout << curr->data << " ";

curr = curr->next;

}

cout << endl;

}

void removeDuplicate() {

cout << "After Removing Duplicates: \n";

Node\* curr = head;

Node\* temp = NULL;

if (head == NULL) {

cout << "No Nodes Present\n";

}

if (head->next == NULL) {

cout << "Only 1 node present\n";

}

if (head->next->next == NULL) { //for 2 nodes

temp = curr;

curr = curr->next;

if (temp->data == curr->data) {

delete temp;

head = curr;

// delete curr;

// curr==NULL;

}

else {

cout << "Both Nodes have different Data/Values\n";

}

}

// For more than 2 nodes

curr = head;

while (curr != NULL) {

Node\* temp = curr;

Node\* prevTemp = curr;

Node\* nextTemp = curr->next;

while (nextTemp != NULL) {

if (curr->data == nextTemp->data) {

// Remove duplicate node

prevTemp->next = nextTemp->next;

delete nextTemp;

nextTemp = prevTemp->next; // Update nextTemp

}

else {

prevTemp = nextTemp;

nextTemp = nextTemp->next;

}

}

curr = curr->next;

}

}

};

int main() {

LinkedList L1;

L1.insertNodeAtEnd(3);

L1.insertNodeAtEnd(3);

L1.insertNodeAtEnd(7);

L1.insertNodeAtEnd(2);

L1.insertInMiddle(9, 3);

L1.display();

L1.removeDuplicate();

L1.display();

L1.insertNodeAtEnd(3);

L1.display();

L1.removeDuplicate();

L1.display();

L1.insertNodeAtEnd(5);

L1.display();

L1.deleteNode(3); //deleting the node with value 1

L1.display();

L1.deleteLastNode();

L1.display();

if (L1.search(2)) {

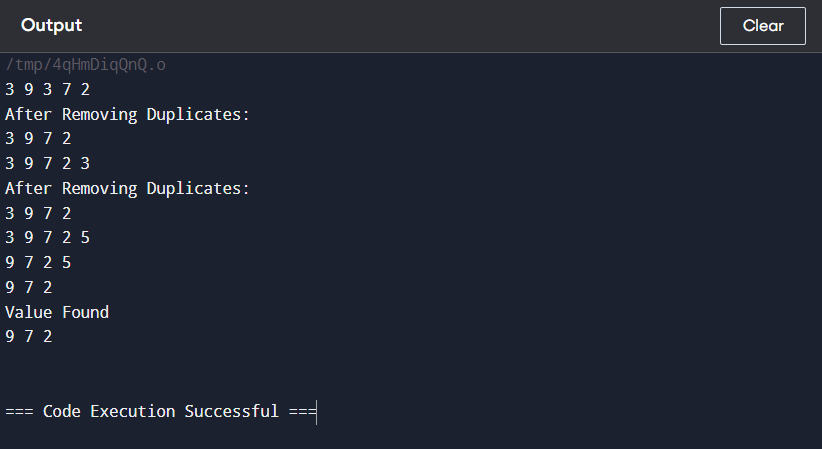
cout << "Value Found\n";

};

L1.display();

}

**OUTPUT:**

****

**Task7:**#include <iostream>

#include <string>

using namespace std;

class Node {

public:

Node\* prev, \* next;

char character;

Node() {

prev = NULL;

next = NULL;

}

Node(char d) {

character = d;

prev = NULL;

next = NULL;

}

};

class DoublyCircularList {

Node\* head;

public:

DoublyCircularList() {

head = NULL;

}

void insertAtTail(char input) {

Node\* newNode = new Node(input);

// When there is no node

if (head == NULL) {

head = newNode;

newNode->next = head;

head->prev = newNode;

return;

}

Node\* curr = head;

// When there is only one node

if (curr->next == head) {

curr->next = newNode;

curr->prev = newNode;

newNode->prev = curr;

newNode->next = curr;

}

// When there are n number of nodes

do {

curr = curr->next;

} while (curr->next != head);

curr->next = newNode;

newNode->prev = curr;

newNode->next = head;

head->prev = newNode;

}

bool checkIfNumbers() {

if (head == nullptr) {

cout << "\n-- List is empty. Cannot check for digits.\n";

return false;

}

Node\* curr = head;

bool allFoundDigits = true;

do {

if (curr->character < '0' || curr->character > '9') {

allFoundDigits = false;

break;

}

curr = curr->next;

} while (curr != head);

return allFoundDigits;

}

bool isVowelCheck(char ch) {

return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||

ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U');

//else return false if not vowel

return false;

}

void rotateString() { //function for rotating string

if (head == NULL) {

cout << "\nList is Empty\n";

return;

}

head = head->next;

}

void PigLatin() {

if (head == NULL) { //if there is no node in list

cout << "\nList is Empty\n";

return;

}

Node\* curr = head;

// if first letter is vowel

if (isVowelCheck(curr->character)) {

displayChar();

cout << "-way" << endl;

return;

}

else {

Node\* Dash = new Node('-'); //for using it as condition fulfils

insertAtTail('-');

}

if (checkIfNumbers()) {

displayChar();

cout << "way" << endl;

return;

}

Node\* endNode;

while (curr->next != head) {

curr = curr->next;

}

endNode = curr;

curr = head;

do {

if (isVowelCheck(curr->character)) {

displayChar();

cout << "ay";

return;

}

else {

rotateString();

}

curr = curr->next;

} while (curr->next != endNode);

displayChar();

cout << "ay" << endl;

}

void displayChar() {

if (head == NULL) {

cout << "\nList is Emptyn";

return;

}

cout << "\nString is: \n\n";

Node\* curr = head;

do {

cout << curr->character;

curr = curr->next;

} while (curr != head);

}

};

int main() {

DoublyCircularList l1;

string Userinput;

cout << "Enter a string: ";

getline(cin, Userinput);

for (int i = 0; i < Userinput.length(); i++) {

l1.insertAtTail(Userinput[i]);

}

// Display the original string

cout << "Original string: ";

l1.displayChar();

// Converting the string to Pig Latin

// l1.display();

cout << "\n\nPig Latin form ==> ";

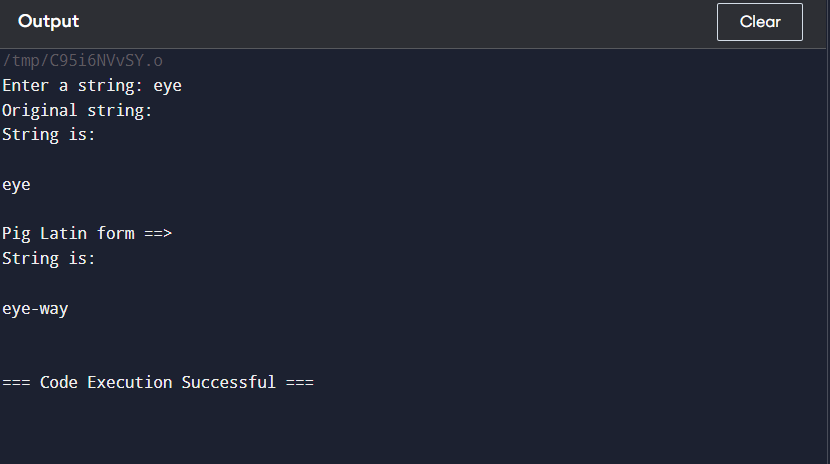
l1.PigLatin();

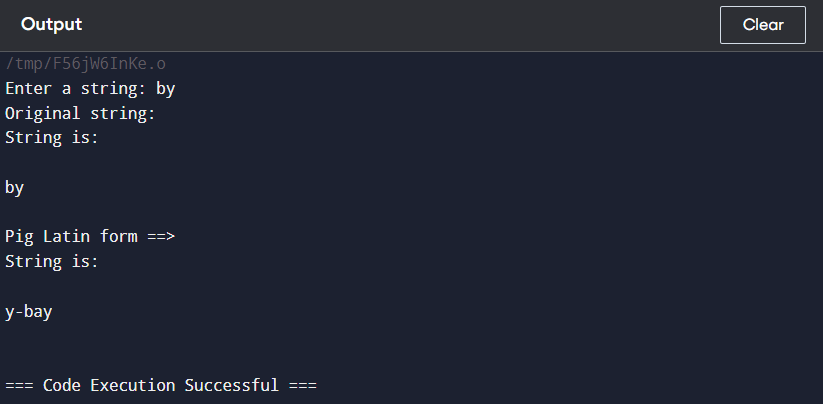
system("pause");

return 0;

}

**OUTPUT:**

****

****