# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**SURYANSH JHA (1BF24CS308)**

**in partial fulfillment for the award of the degree of**
**BACHELOR OF ENGINEERING**
**in**
**COMPUTER SCIENCE AND ENGINEERING**

**B.M.S. COLLEGE OF ENGINEERING**
**(Autonomous Institution under VTU)**
**BENGALURU-560019**
**August-December 2025**

**B. M. S. College of Engineering,**
**Bull Temple Road, Bangalore 560019**
**(Affiliated To Visvesvaraya Technological University, Belgaum)**
**Department of Computer Science and Engineering**

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by **Suryansh Jha (1BF24CS308)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2025-2026. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.


**Anusha S**
Assistant Professor
Department of CSE
BMSCE, Bengaluru

**Dr. Kavitha Sooda**
Professor and Head
Department of CSE
BMSCE, Bengaluru

**Index Sheet**



**Course outcomes:**

| | |
|---|---|
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyze data structure operations for a given problem |
| CO3 | Design and develop solutions using the operations of linear and nonlinear data structure for a given specification. |
| CO4 | Conduct practical experiments for demonstrating the operations of different data structures. |

**Lab program 1:**

**Write a program to simulate the working of stack using an array with the following:**
**a) Push**
**b) Pop**
**c) Display**
**The program should print appropriate messages for stack overflow, stack underflow.**

**Code:**

```
#include <stdio.h>
#define n 5
int stack[n];
int top = -1;
void push(int x)
{
   if (top == (n - 1))
   {
      printf("Stack overflow"); return;
   }
   else
   {
      top++;
      stack[top] = x;
   }
}
void pop()
{
   if (top == -1)
   {
      printf("Underflow");
   }
   else
   {
      printf("The deleted element is %d", stack[top]);
      top--;
   }
}
void peek()
{
   if (top == -1)
   {
      printf("Underflow");
   }
   else
   {
      printf("The top element is %d", stack[top]);
   }
}
int main()
{
```

```
    int v, w;
    printf("1.PUSH\n2.POP\n3.PEEK\n4.EXIT\n");
    for (int i = 1; i > 0; i++)
    {
        printf("\nEnter the function to perfrom: ");
        scanf("%d", &w);
        switch (w)
        {
        case 1:
            printf("Enter the element to insert: ");
            scanf("%d", &v);
            push(v);
            break;
        case 2:
            pop();
            break;
        case 3:
            peek();
            break;
        case 4:
            return 0;
            break;
        default:
            printf("Invalid input");
        }
    }
    return 0;
}
```

**Output:**

```
1.PUSH
2.POP
3.PEEK
4.EXIT

Enter the function to perfrom: 1
Enter the element to insert: 23

Enter the function to perfrom: 1
Enter the element to insert: 34

Enter the function to perfrom: 1
Enter the element to insert: 45

Enter the function to perfrom: 1
Enter the element to insert: 56

Enter the function to perfrom: 1
Enter the element to insert: 67

Enter the function to perfrom: 1
Enter the element to insert: 78
Stack overflow
Enter the function to perfrom: 2
The deleted element is 67
Enter the function to perfrom: 2
The deleted element is 56
Enter the function to perfrom: 2
The deleted element is 45
Enter the function to perfrom: 2
The deleted element is 34
Enter the function to perfrom: 2
The deleted element is 23
Enter the function to perfrom: 2
Underflow
Enter the function to perfrom: 2
Underflow
Enter the function to perfrom: 1
Enter the element to insert: 23

Enter the function to perfrom: 1
Enter the element to insert: 234
```

**Lab Program 2:**

**WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. Theexpression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and /(divide)**

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define n 11
char stack[n];
int top = -1;
void push(char c)
{
   if (top == (n - 1))
   {
      printf("Stack overflow");
      return;
   }
   top++;
   stack[top] = c;
}
char pop()
{
   if (top == -1)
   {
      printf("Stack underflow");
      return -1;
   }
   return stack[top--];
}
char peek()
```

```
{
    if (top == -1)
    {
        printf("Stack underflow");
        return -1;
    }
    return stack[top];
}
int associativity(char c)
{
    if (c == '^')
    {
        return 1;
    }
    return 0;
}
int precedence(char c)
{
    switch (c)
    {
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
        return 2;
    case '^':
        return 3;
    case '(':
```

```
        return 0;
    }
    return -1;
}
void postfixtoprefix(char prefix[n], char postfix[n])
{
    int i;
    int k = 0;
    char c;

    for (i = 0; i < strlen(prefix); i++)
    {
        c = prefix[i];
        if (isalnum(c))
        {
            postfix[k++] = c;
        }
        else if (c == '(')
        {
            push(c);
        }
        else if (c == ')')
        {
            while (peek() != '(')
            {
                postfix[k++] = pop();
            }
            pop();
        }
```

```
        else
        {
            while (top != -1 && ((precedence(peek()) > precedence(c)) || ((precedence(peek()) == precedence(c)) && associativity(c) == 0)))
            {
                postfix[k++] = pop();
            }
            push(c); }}
    while (top != -1)
    {
        postfix[k++] = pop();
    }
    postfix[k] = '\0';
}
int main()
{
    char infix[n];
    printf("Enter the infix equation: ");
    gets(infix);
    char postfix[n];
    postfixtoprefix(infix, postfix);
    printf("The postfix expression is: ");
    puts(postfix);
    return 0;
}
```

**Output:**

```
PS D:\1BF24CS308 DS> cd 'd:\1BF24CS308 DS\output'
PS D:\1BF24CS308 DS\output> & .\'practice.exe'
Enter the infix equation: (a+(b*c-(d/e^f)*g)*h)
The postfix expression is: abc*def^/g*-h*+
```

**Lab Program 3:**

**a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display .The programshould print appropriate messagesfor queue empty and queue overflow conditions**

**Code:**

```
#include <stdio.h>

#define n 5

int queue[n];

int front = -1;

int rear = -1;

void enueue(int x)

{

   if (rear == n - 1)

   {

      printf("Queue is full\n");

   }

   else if (rear == -1)

   {

      front++;

      rear++;

      queue[rear] = x;

   }

   else

   {

      rear++;

      queue[rear] = x;

   }

}
```

```
void dequeue()
{
   if (front == -1)
   {
      printf("Empty queue\n");
   }
   else if (front == rear)
   {
      int w = queue[front];
      front = rear = -1;
      printf("The deleted eleemnt is %d\n", w);
   }
   else
   {
      int q = queue[front];
      front++;
      printf("The deleted element is: %d\n", q);
   }
}
void display()
{
   printf("The elements of aueue are: ");
   for (int i = front; i <= rear; i++)
   {
      printf("%d\n", queue[i]);
   }
}
int main()
{
```

```
    int s, c;
    printf("1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.EXIT\n");
    for (int i = 1; i > 0; i++)
    {
        printf("Enter the function to perfrom: ");
        scanf("%d", &c);
        switch (c)
        {
        case 1:
            printf("Enter the element to insert: ");
            scanf("%d", &s);
            enueue(s);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            return 0;
        }
    }
}
```

**Output:**

```
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
Enter the function to perfrom: 1
Enter the element to insert: 23
Enter the function to perfrom: 1
Enter the element to insert: 34
Enter the function to perfrom: 1
Enter the element to insert: 56
Enter the function to perfrom: 1
Enter the element to insert: 45
Enter the function to perfrom: 1
Enter the element to insert: 67
Enter the function to perfrom: 1
Enter the element to insert: 678
Queue is full
Enter the function to perfrom: 2
The deleted element is: 23
Enter the function to perfrom: 2
The deleted element is: 34
Enter the function to perfrom: 2
The deleted element is: 56
Enter the function to perfrom: 2
The deleted element is: 45
Enter the function to perfrom: 2
The deleted eleemnt is 67
Enter the function to perfrom: 2
Empty queue
Enter the function to perfrom: 1
Enter the element to insert: 23
Enter the function to perfrom: 1
Enter the element to insert: 34
Enter the function to perfrom: 3
The elements of aueue are: 23
34
Enter the function to perfrom: 4
```

**Lab Program 3b:**

**b ) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display .The programshould print appropriate messagesfor queue empty and queue overflow conditions**

```
#include<stdio.h>
#define n 5
int queue[n];
int front=-1;
int rear=-1;
void enqueue(int x){
   if(((rear+1)%n)==front){
      printf("Queue is full\n");
   }
   else if(front==-1&&rear==-1){
      front++;rear++;
      queue[rear]=x;
   }
   else{
      rear=(rear+1)%n;
      queue[rear]=x;
   }
}
void dequeue(){
   if(rear==-1&&front==-1){
      printf("Queue is empty\n");
   }
   else if(front==rear){
      int y=queue[front];
      printf("The deleted element is: %d\n",y);
      front=rear=-1;
```

```
    }
    else{
        int y=queue[front];
        front=(front+1)%n;
        printf("The deleted element is: %d",y);

    }
}
void display()
{
    printf("The elements of queue are: ");
    for(int i = front; i != (rear+1)%n;i=(i+1)%n)
    {
        printf("%d\n", queue[i]);
    }
}
int main()
{
    int s, c;
    printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
    for (int i = 1; i > 0; i++)
    {
        printf("\nEnter the function to perfrom: ");
        scanf("%d", &c);
        switch (c)
        {
        case 1:
            printf("Enter the element to insert: ");
            scanf("%d", &s);
```

```
        enqueue(s);
        break;
    case 2:
        dequeue();
        break;
    case 3:
        display();
        break;
    case 4:
        return 0;}}}
```

**Output:**

```
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter the function to perfrom: 1
Enter the element to insert: 23

Enter the function to perfrom: 1
Enter the element to insert: 34

Enter the function to perfrom: 1
Enter the element to insert: 45

Enter the function to perfrom: 1
Enter the element to insert: 56

Enter the function to perfrom: 1
Enter the element to insert: 67

Enter the function to perfrom: 1
Enter the element to insert: 678
Queue is full

Enter the function to perfrom: 2
The deleted element is: 23
Enter the function to perfrom: 2
The deleted element is: 34
Enter the function to perfrom: 2
The deleted element is: 45
Enter the function to perfrom: 2
The deleted element is: 56
Enter the function to perfrom: 2
The deleted element is: 67

Enter the function to perfrom: 2
Queue is empty

Enter the function to perfrom: 2
Queue is empty
```

**Lab Program 4:**

**WAP to Implement Singly Linked List with followingoperations**

**a) Createalinkedlist.**

**b) Insertion of a node at first position, at any position and at end of list.**

**Display the contents of the linked list.**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head = NULL;
void creation(int n)
{
   int a;
   if (n < 0)
   {
      printf("Number shoulb be greater than 1\n");
   }
   else
   {
      struct node *newnode, *temp;

      for (int i = 0; i < n; i++)
      {
         newnode = (struct node *)malloc(sizeof(struct node));
```

```
            printf("Enter the data: ");
            scanf("%d", &a);
            newnode->data = a;
            newnode->next = NULL;
            if (head == NULL)
            {
               head = newnode;
            }
            else
            {
               temp->next = newnode;
            }
            temp = newnode;
         }
      }
   }
void insert_at_begining(int a)
{
   struct node *newnode;
   newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = a;
   newnode->next = NULL;
   if (head == NULL)
   {
      head = newnode;
   }
   else
```

```
    {
        newnode->next = head;
        head = newnode;
    }
}
void insert_at_end(int a)
{
    struct node *newnode, *temp;
    temp = head;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = a;
    newnode->next = NULL;
    if (head == NULL)
    {
        head = newnode;
    }

    else
    {
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newnode;
    }
}
void insert_at_position(int a, int p)
```

```
{
    struct node *newnode, *temp;
    temp = head;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = a;
    for (int i = 1; i < p - 1; i++)
    {
        temp = temp->next;
    }
    newnode->next = temp->next;
    temp->next = newnode;
}
void display()
{
    printf("DISPLAY FUNCTION CALLED\n");
    struct node *temp;
    temp = head;
    printf("The elements of linked list are: \n");
    while (temp != NULL)
    {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}
int main()
{
    int a, p, c;
```

```
for (int i = 1; i > 0; i++)
{
    printf("\nEnter the operation: ");
    scanf("%d", &c);
    switch (c)
    {
    case 1:
        printf("CREATION CALLED\n");

        printf("Enter the number of elements to insert: ");
        scanf("%d", &a);
        creation(a);
        break;
    case 2:
        printf("INSERT AT BEGINING CALLED\n");

        printf("Enter the element to insert: ");
        scanf("%d", &a);
        insert_at_begining(a);
        break;
    case 3:
        printf("INSERT AT END CALLED\n");

        printf("Enter the element to insert: ");
        scanf("%d", &a);
        insert_at_end(a);
        break;
```

```
        case 4:
            printf("INSERT AT ANY POSITION CALLED\n");

            printf("Enter the position to insert: ");
            scanf("%d", &p);
            printf("Enter the element to insert: ");
            scanf("%d", &a);
            insert_at_position(a, p);
            break;
        case 5:
            display();
            break;
        case 6:
            return 0;
        default:
            printf("Invalid input\n");
            break;
        }
    }

    return 0;
}
```

**Output:**

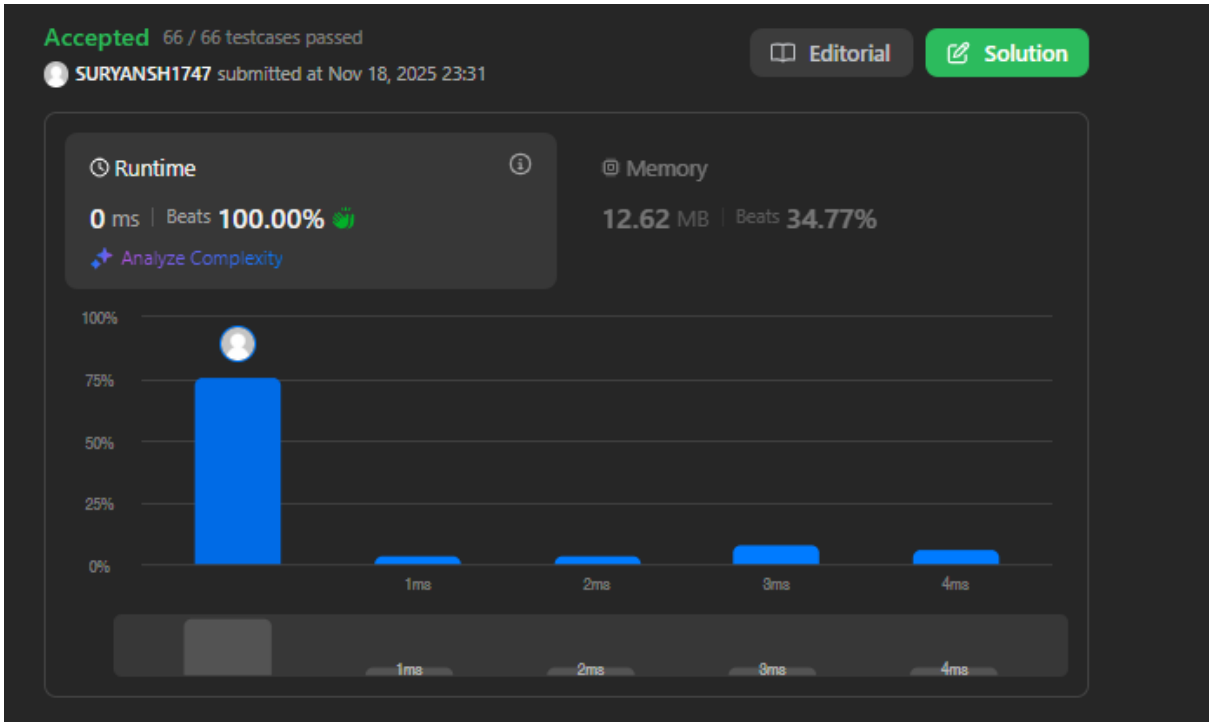
```
Enter the operation: 1
CREATION CALLED
Enter the number of elements to insert: 5
Enter the data: 34
Enter the data: 45
Enter the data: 67
Enter the data: 78
Enter the data: 89

Enter the operation: 2
INSERT AT BEGINING CALLED
Enter the element to insert: 356

Enter the operation: 3
INSERT AT END CALLED
Enter the element to insert: 499

Enter the operation: 4
INSERT AT ANY POSITION CALLED
Enter the position to insert: 3
Enter the element to insert: 699

Enter the operation: 5
DISPLAY FUNCTION CALLED
The elements of linked list are:
356
34
699
45
67
78
89
499

Enter the operation: 6
```

**Lab Program 4b LEETCODE:**

**Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.**

```
struct ListNode* removeElements(struct ListNode* head, int val) {
    struct ListNode* temp=head;

    struct ListNode* prev;struct ListNode* o=temp;
    while(temp!=NULL){
        if(head->val==val){
        head=head->next;
        free(temp);
        temp=head;
        continue;
    }
        if(temp->val==val){
            prev->next=temp->next;
            o=temp;temp=temp->next;
            free(o);
            continue;
        }
        prev=temp;
        temp=temp->next;

    }
    return head;
}
```

**OUTPUT:**


Accepted 66 / 66 testcases passed
SURYANSH1747 submitted at Nov 18, 2025 23:31
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
12.62 MB | Beats 34.77%
100%
75%
50%
25%
0%
1ms
2ms
3ms
4ms
1ms
2ms
3ms
4ms

**Lab Program 5:**

**WAP to Implement Singly Linked List with following operations**

**a) Create a linked list.**

**b) Deletion of first element, specified element and last element in the list.**

**c) Display the contents of the linked list.**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;struct node* next;
};
struct node* head=NULL;
void creation(int m){
   if(m<=0){
      printf("Value cannot be less than zero equal to zero");
      return;
   }int l;
      struct node* temp=head;

   for(int i=0;i<m;i++){
      struct node* newnode=(struct node*)malloc(sizeof(struct node));
      printf("Enter the data: ");
      scanf("%d",&l);
      newnode->data=l;newnode->next=NULL;
      if(head==NULL){
         head=newnode;
      }
      else{
```

```
            temp->next=newnode;
        }
        temp=newnode;
    }

}
void deletion_at_begining(){
    if(head==NULL){
        printf("EMPTY LIST");
        return;
    }
    struct node* temp=head;
    head=head->next;
    printf("The deleted element is %d",temp->data);
    free(temp);
}
void deletion_at_end(){

    if (head == NULL) {
        printf("EMPTY LIST\n");
        return;
    }
    struct node* temp = head;
    struct node* prev = NULL;
    if (head->next == NULL) {
        printf("The deleted element is: %d\n", temp->data);
        free(head);
```

```
        head = NULL;
        return;
    }
    while(temp->next != NULL){
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    printf("The deleted element is: %d\n", temp->data);
    free(temp);
}
void delete_by_position(int p){
    if(p<=0){
        printf("POSITION CANNOT BE ZERO OR LESS THAN ZERO");
        return;
    }
    struct node* temp=head;struct node* prev=NULL;
    for(int i=1;i<p;i++){
        prev=temp;
        temp=temp->next;

    }
    prev->next=temp->next;
    printf("The deleted element is %d",temp->data);
    free(temp);
}
void display(){
```

```
    struct node* temp=head;
    while(temp!=NULL){
        printf("%d\n",temp->data);
        temp=temp->next;
    }
}

int main(){
    int n,ch;int i=9;
    printf("1.Creation\n2.Delete at Begining\n3.Delete at end\n4.Delete by
position\n5.Display\n6.Exit\n");
    while(i>0){
        printf("\nEnter the operation: ");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                printf("Enter the number of nodes to insert: ");
                scanf("%d",&n);
                creation(n);
                break;
            case 2:
                deletion_at_begining();
                break;
            case 3:
                deletion_at_end();
                break;
            case 4:
```

```
            printf("Enter the positon to delete: ");
            scanf("%d",&n);
            delete_by_position(n);
            break;
        case 5:
            printf("ELEMENTS OF LINKED LIST ARE: \n");
            display();
            break;
        case 6:
            exit(0);
            break;
        default:
            printf("Invalid input");

      }
   }
   return 0;
}
```

**Ouput:**

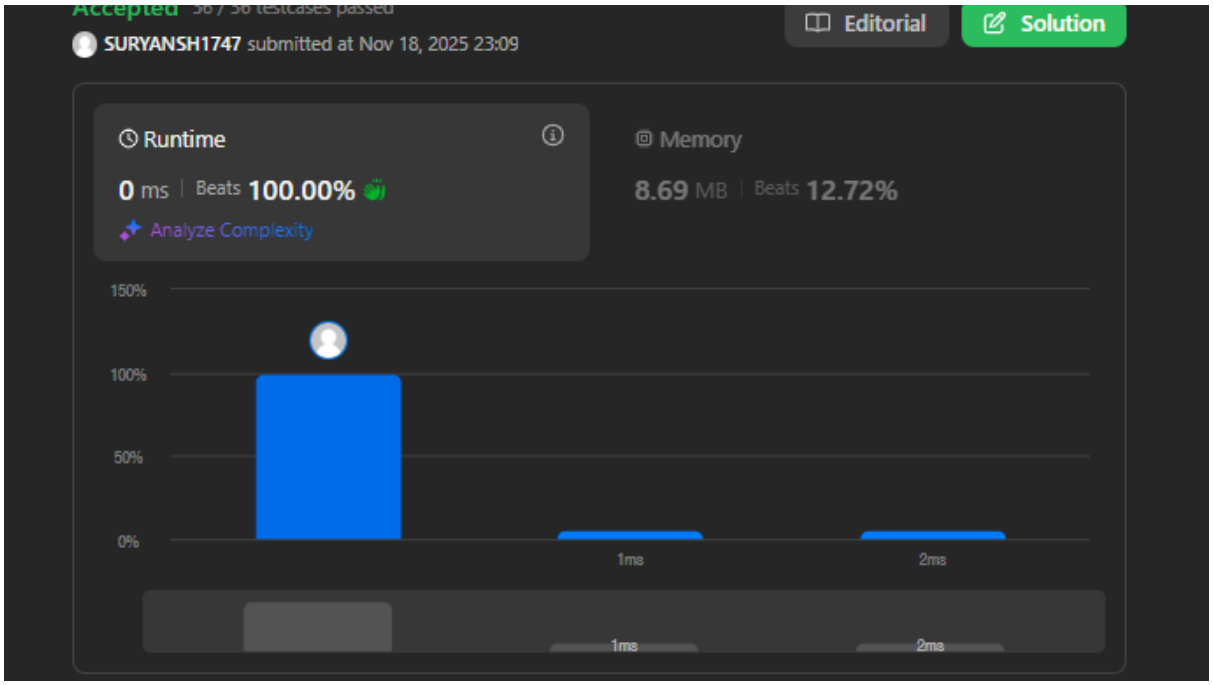
```
1.Creation
2.Delete at Begining
3.Delete at end
4.Delete by position
5.Display
6.Exit

Enter the operation: 1
Enter the number of nodes to insert: 7
Enter the data: 23
Enter the data: 34
Enter the data: 45
Enter the data: 56
Enter the data: 67
Enter the data: 78
Enter the data: 89

Enter the operation: 2
The deleted element is 23
Enter the operation: 3
The deleted element is: 89

Enter the operation: 4
Enter the positon to delete: 3
The deleted element is 56
Enter the operation: 5
ELEMENTS OF LINKED LIST ARE:
34
45
67
78

Enter the operation: 6
```

**Lab Program 5b LEETCODE:**

**Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.**

```
struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode* temp=head;int count=0;int a;int b=0;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    if(count%2==1){
        a=(count+1)/2;
        for(int i=1;i<a;i++){
            head=head->next;
        }
        return head;}
    else{
        a=(count/2)+1;
        for(int i=1;i<a;i++){
            head=head->next;
        }
        return head;
    }
    return head;

}
```

**Output:**



**Lab Program 6a:**

**a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list,Concatenation of two linked lists.**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head1 = NULL;
struct node *head2 = NULL;
void creation1(int n)
{
   int a;
   if (n < 0)
   {
      printf("Number should be greater than 1\n");
   }
   else
   {
      struct node *newnode, *temp;

      for (int i = 0; i < n; i++)
      {
         newnode = (struct node *)malloc(sizeof(struct node));
         printf("Enter the data: ");
```

```
            scanf("%d", &a);
            newnode->data = a;
            newnode->next = NULL;
            if (head1 == NULL)
            {
               head1 = newnode;
            }
            else
            {
               temp->next = newnode;
            }
            temp = newnode;
         }
      }
}
void creation2(int n)
{
   int a;
   if (n < 0)
   {
      printf("Number shoulb be greater than 1\n");
   }
   else
   {
      struct node *newnode, *temp;

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

```
        {
            newnode = (struct node *)malloc(sizeof(struct node));
            printf("Enter the data: ");
            scanf("%d", &a);
            newnode->data = a;
            newnode->next = NULL;
            if (head2 == NULL)
            {
                head2 = newnode;
            }
            else
            {
                temp->next = newnode;
            }
            temp = newnode;
        }
    }
}
void display()
{
    printf("DISPLAY FUNCTION CALLED\n");
    struct node *temp;
    temp = head1;
    printf("The elements of linked list are: \n");
    while (temp != NULL)
    {
        printf("%d\n", temp->data);
```

```
        temp = temp->next;
    }
}
void concatenation(){
    if(head1==NULL){
        head1=head2;return;
    }
    struct node* temp=head1;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=head2;
}
void sort(){
    if(head1==NULL){
        printf("LIST IS EMPTY\n");return;
    }
    struct node *i,*j;
    int temp;
    for(i=head1;i->next!=NULL;i=i->next){
        for(j=i->next;j!=NULL;j=j->next){
            if(i->data>j->data){
                temp=i->data;
                i->data=j->data;
                j->data=temp;
            }
        }
```

```
    }


}
void reverse(){
    struct node* prev=NULL;
    struct node* current=head1;
    struct node* next;
    while(current!=NULL){
        next=current->next;
        current->next=prev;prev=current;
        current=next;
    }
    head1=prev;
}
int main()
{
    int a, p, c;
    printf("1.Creation of list 1\n2.Cfreation of list
2\n3.Concatenation\n4.Sort\n5.Reverse\n6.Display\n7.Exit");
    for (int i = 1; i > 0; i++)
    {
        printf("\nEnter the operation: ");
        scanf("%d", &c);
        switch (c)
        {
        case 1:
            printf("CREATION 1 CALLED\n");
```

```
    printf("Enter the number of elements to insert: ");
    scanf("%d", &a);
    creation1(a);
    break;
 case 2:
    printf("CREATION 2 CALLED\n");

    printf("Enter the number of elements to insert: ");
    scanf("%d", &a);
    creation2(a);
    break;
case 3:
    printf("CONVATENATION CALLED");
    concatenation();
    break;
case 4:
    printf("SORT CALLED");
    sort();
    break;
case 5:
    printf("REVERSE CALLED");
    reverse();
    break;
case 6:
    display();
    break;
```

```
        case 7:
            return 0;break;
        default:
            printf("INVALID INPUT\n");
            break;
        }}
    return 0;
}
```

**Output:**

```
PS D:\1BF24CS308 DS> cd 'd:\1BF24CS308 DS\output'
PS D:\1BF24CS308 DS\output> & .\'lab6a.exe'
1.Creation of list 1
2.Cfreation of list 2
3.Concatenation
4.Sort
5.Reverse
6.Display
7.Exit
Enter the operation: 1
CREATION 1 CALLED
Enter the number of elements to insert: 4
Enter the data: 23
Enter the data: 67
Enter the data: 2
Enter the data: 789

Enter the operation: 2
CREATION 2 CALLED
Enter the number of elements to insert: 3
Enter the data: 78
Enter the data: 34
Enter the data: 56

Enter the operation: 3
CONVATENATION CALLED
Enter the operation: 6
DISPLAY FUNCTION CALLED
The elements of linked list are:
23
67
2
789
78
34
56
```

```
Enter the operation: 4
SORT CALLED
Enter the operation: 6
DISPLAY FUNCTION CALLED
The elements of linked list are:
2
23
34
56
67
78
789

Enter the operation: 5
REVERSE CALLED
Enter the operation: 6
DISPLAY FUNCTION CALLED
The elements of linked list are:
789
78
67
56
34
23
2

Enter the operation: 7
PS D:\1BF24CS308 DS\output>
```

**Lab Program 6b:**

**b) WAP to Implement Single Link List to simulate Stack & Queue Operations.**

**For stack**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *top = NULL;
void push(int a)
{
   struct node *newnode;
   newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = a;
   newnode->next = NULL;
   if (top == NULL)
   {
      top = newnode;
   }
   else
   {
      newnode->next = top;
      top = newnode;
   }
}
```

```
void pop()
{
    if (top == NULL)
    {
        printf("Stack underflow\n");
        return;
    }
    else if (top->next == NULL)
    {
        printf("The deleted data is: %d\n", top->data);
        struct node *temp = top;
        top = NULL;
        free(temp);
    }
    else
    {
        struct node *temp = top;
        printf("The deleted data is: %d\n", top->data);

        top = top->next;
        free(temp);
    }
}
void display()
{
    struct node *temp = top;
    printf("The elements of stack are: \n");
```

```
    while (temp != NULL)
    {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}
int main()
{
    int a, p;
    printf("1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
    while (1)
    {
        printf("Enter the operation: ");
        scanf("%d", &a);
        switch (a)
        {
        case 1:
            printf("Enter the element ot push: ");
            scanf("%d", &p);
            push(p);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
```

```
        case 4:
            return 0;
        }
    }
    return 0;
}
```

**Output:**

```
PS D:\1BF24CS308 DS> cd 'd:\1BF24CS308 DS\output'
PS D:\1BF24CS308 DS\output> & .\'stacklinklist.exe'
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter the operation: 1
Enter the element ot push: 23
Enter the operation: 1
Enter the element ot push: 34
Enter the operation: 1
Enter the element ot push: 45
Enter the operation: 1
Enter the element ot push: 67
Enter the operation: 2
The deleted data is: 67
Enter the operation: 2
The deleted data is: 45
Enter the operation: 1
Enter the element ot push: 567
Enter the operation: 3
The elements of stack are:
567
34
23
Enter the operation: 4
PS D:\1BF24CS308 DS\output>
```

**For QUEUE**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;
void enqueue(int a)
{
   struct node *newnode;
   newnode = (struct node *)malloc(sizeof(struct node));
   if(newnode==NULL){
      printf("MEMORT ALLOCATION FAILED");
      return;
   }
   newnode->data = a;
   newnode->next = NULL;
   if (front == NULL)
   {
      front = newnode;rear=newnode;
   }
   else
   {
```

```
        rear->next = newnode;
        rear = newnode;
    }
}
void dequeue()
{
    if (front == NULL)
    {
        printf("Queue is empty\n");
        return;
    }
    else if (front->next == NULL)
    {
        printf("The deleted data is: %d \n", front->data);
        struct node *temp = front;
        front = rear = NULL;
        free(temp);
    }
    else
    {
        printf("The deleted data is: %d \n", front->data);
        struct node *temp = front;
        front = front->next;
        free(temp);
    }
}
void display()
```

```
{
    struct node *temp = front;
    printf("The elements of queue are: \n");
    while (temp != NULL)
    {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}
int main()
{
    int a, p;
    printf("1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.EXIT\n");
    while (1)
    {
        printf("Enter the operation: ");
        scanf("%d", &a);
        switch (a)
        {
        case 1:
            printf("Enter the element to enqueue: ");
            scanf("%d", &p);
            enqueue(p);
            break;
        case 2:
            dequeue();
            break;
```

```
        case 3:
            display();
            break;
        case 4:
            return 0;
        }
    }
    return 0;
}
```

**OUPUT:**

```
PS D:\1BF24CS308 DS> cd 'd:\1BF24CS308 DS\output'
PS D:\1BF24CS308 DS\output> & .\'queuelinklist.exe'
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
Enter the operation: 1
Enter the element to enqueue: 23
Enter the operation: 1
Enter the element to enqueue: 45
Enter the operation: 1
Enter the element to enqueue: 67
Enter the operation: 1
Enter the element to enqueue: 89
Enter the operation: 2
The deleted data is: 23
Enter the operation: 2
The deleted data is: 45
Enter the operation: 1
Enter the element to enqueue: 345
Enter the operation: 1
Enter the element to enqueue: 67
Enter the operation: 3
The elements of queue are:
67
89
345
67
Enter the operation: 4
PS D:\1BF24CS308 DS\output>
```

**Lab Program 7:**

**WAP to Implement doubly link list with primitive operations**

**a) Create a doubly linked list.**

**b) Insert a new node to the left of the node.**

**c) Delete the node based on a specific value**

**d) Display the contents of the list**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;struct node* next;struct node* prev;
};
struct node* head=NULL;struct node* tail=NULL;
void creation(int m){
   if(m<=0){
      printf("Value cannot be less than zero equal to zero");
      return;
   }int l;
   for(int i=0;i<m;i++){
      struct node* newnode=(struct node*)malloc(sizeof(struct node));
      printf("Enter the data: ");
      scanf("%d",&l);
      newnode->data=l;newnode->prev=NULL;newnode->next=NULL;
      if(head==NULL){
         head=newnode;
      }
      else{
```

```
            tail->next=newnode;
            newnode->prev=tail;
        }
        tail=newnode;
    }

}
void insert_at_beegining(int val){
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=val;newnode->prev=NULL;
    newnode->next=NULL;
    if(head==NULL){head=tail=newnode;return;}
    newnode->next=head;
    head->prev=newnode;
    head=newnode;

}
void delete_by_value(int val){
    struct node* temp=head;
    while(temp!=NULL&&temp->data!=val){
        temp=temp->next;
    }
    if(temp==NULL){
        printf("VALUE NOT FOUND");
        return;
    }
    temp->prev->next=temp->next;
```

```
    temp->next->prev=temp->prev;
    free(temp);
}
void display(){
    struct node* temp=head;
    while(temp!=NULL){
        printf("%d\n",temp->data);
        temp=temp->next;
    }
}

int main(){
    int n,ch;int i=9;
    printf("1.Creation\n2.Insert at Begining\n3.Delete by value\n4.Display\n5.Exit\n");
    while(i>0){
        printf("\nEnter the operation: ");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                printf("Enter the number of nodes to insert: ");
                scanf("%d",&n);
                creation(n);
                break;
            case 2:
                printf("Enter the value to insert: ");
                scanf("%d",&n);
                insert_at_beegining(n);
```

```
            break;
        case 3:
            printf("Enter th value to delete: ");
            scanf("%d",&n);
            delete_by_value(n);
            break;
        case 4:
            printf("The elements of list are: \n");
            display();
            break;
        case 5: exit(0);
            break;
        default:
            printf("Invalid input");

      }
   }
   return 0;
}
```

**Output:**

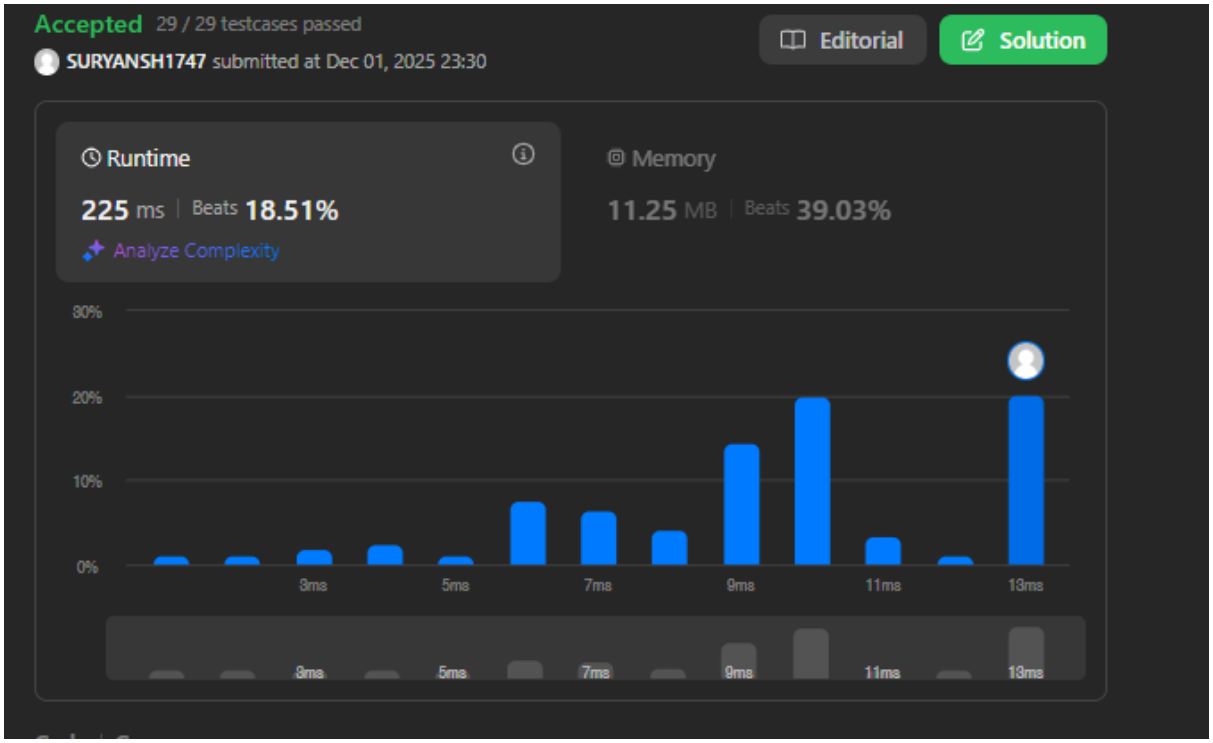
```
1.Creation
2.Insert at Begining
3.Delete by value
4.Display
5.Exit

Enter the operation: 1
Enter the number of nodes to insert: 5
Enter the data: 23
Enter the data: 34
Enter the data: 45
Enter the data: 56
Enter the data: 67

Enter the operation: 2
Enter the value to insert: 345

Enter the operation: 3
Enter th value to delete: 45

Enter the operation: 4
The elements of list are:
345
23
34
56
67

Enter the operation: 5
```

**Lab Program 7b LEETCODE:**

**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.**

```
bool check(struct ListNode* t,struct ListNode* h,int o){
    struct ListNode* u=h;
    for(int i=0;i<o;i++){
        if(t->next==u){
            return true;
        }
        u=u->next;
    }
    return false;
 }
bool hasCycle(struct ListNode *head) {
    if (head == NULL || head->next == NULL) {
        return false;
    }
    struct ListNode* temp=head;int p=1;
    while(head!=NULL){
        temp=temp->next;p++;
        if(temp==NULL){return false;}
        if(check(temp,head,p)){
            return true;
        }
    }
    return false;
}
```

**OUTPUT:**


Accepted 29 / 29 testcases passed
SURYANSH1747 submitted at Dec 01, 2025 23:30
Editorial
Solution
Runtime
225 ms | Beats 18.51%
Analyze Complexity
Memory
11.25 MB | Beats 39.03%
30%
20%
10%
0%
3ms
5ms
7ms
9ms
11ms
13ms

**Lab Program 8:**

**8a) Write a program**

**a) ToconstructabinarySearchtree.**

**b) To traverse the tree using all the methods i.e., in-order, preorder and post order**

**c) To display the elements in the tree.**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *left;
    struct node *right;
};
struct node* create_node(int value){
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->left = NULL;
    newnode->right = NULL;
    return newnode;
}
struct node* insert(struct node* root, int value){
    if(root == NULL){
        return create_node(value);
    }
    if(value < root->data){
        root->left = insert(root->left, value);
    } else if(value > root->data){
        root->right = insert(root->right, value);
```

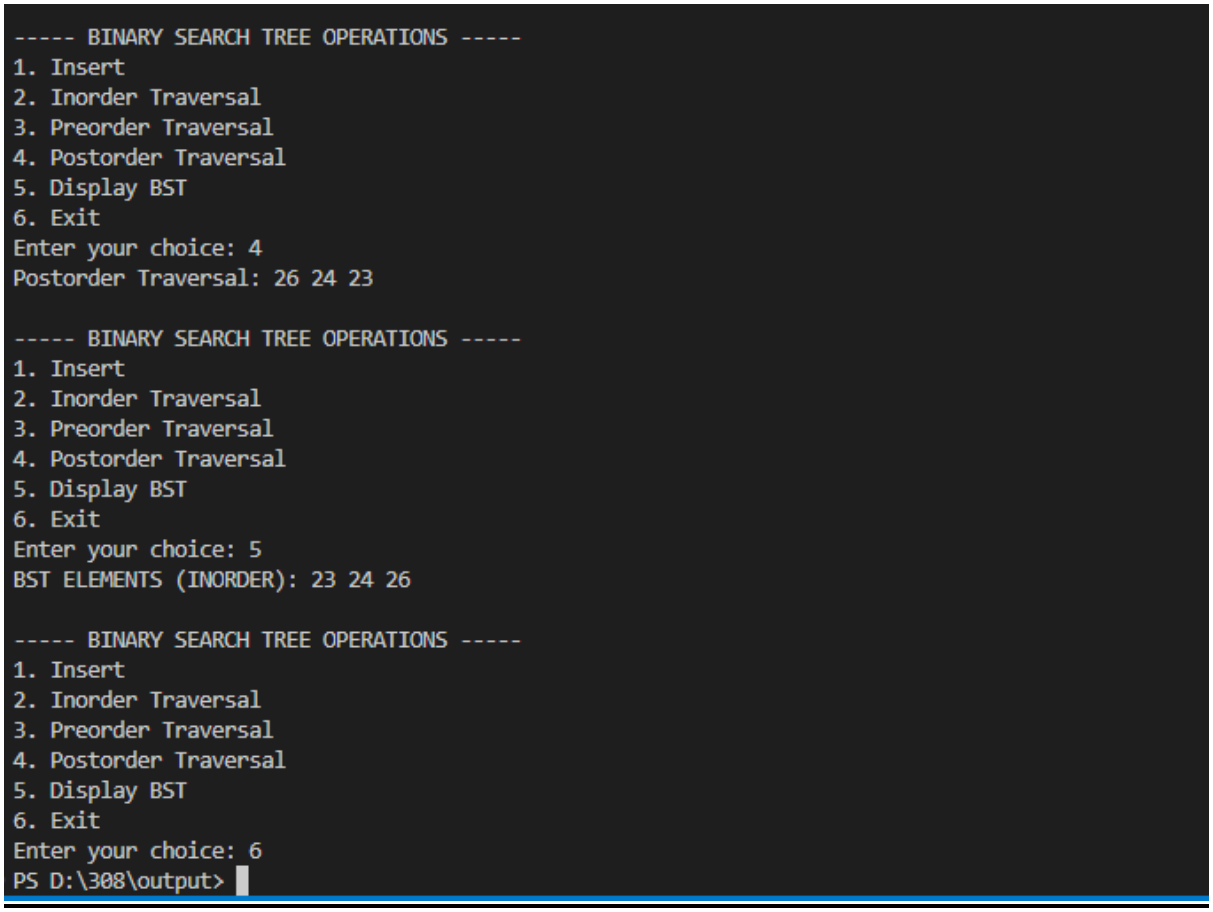
```
    }
    return root;
}
void inorder(struct node* root){
    if(root==NULL){
        return;
    }
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
void postorder(struct node* root){
    if(root==NULL){
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
void preorder(struct node* root){
    if(root==NULL){
        return;
    }
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}
```

```
void display(struct node* root ){
    printf("BST ELEMENTS (INORDER): ");
    inorder(root);
    printf("\n");
}
int main(){
    struct node* root = NULL;
    int choice, value;
    while(1){
        printf("\n----- BINARY SEARCH TREE OPERATIONS -----\n");
        printf("1. Insert\n");
        printf("2. Inorder Traversal\n");
        printf("3. Preorder Traversal\n");
        printf("4. Postorder Traversal\n");

        printf("5. Display BST\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
          case 1:
             printf("Enter value to insert: ");
             scanf("%d", &value);
             root = insert(root, value);
             break;
            case 2:
             printf("Inorder Traversal: ");
```

```
            inorder(root);
            printf("\n");
            break;
           case 3:
            printf("Preorder Traversal: ");
            preorder(root);
            printf("\n");
            break;
           case 4:
            printf("Postorder Traversal: ");
           postorder(root);
            printf("\n");
            break;
           case 5:
            display(root);
            break;
             case 6:
             exit(0);
         default:
             printf("INVALID INPUT\n");
      }
   }
   return 0;
}
```

**Output:**

```
----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter value to insert: 23

----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter value to insert: 24

----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter value to insert: 26

----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 2
Inorder Traversal: 23 24 26

----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 3
Preorder Traversal: 23 24 26
```


----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 4
Postorder Traversal: 26 24 23
----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 5
BST ELEMENTS (INORDER): 23 24 26
----- BINARY SEARCH TREE OPERATIONS -----
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 6
PS D:\308\output>

**Lab Program 8b LEETCODE:**

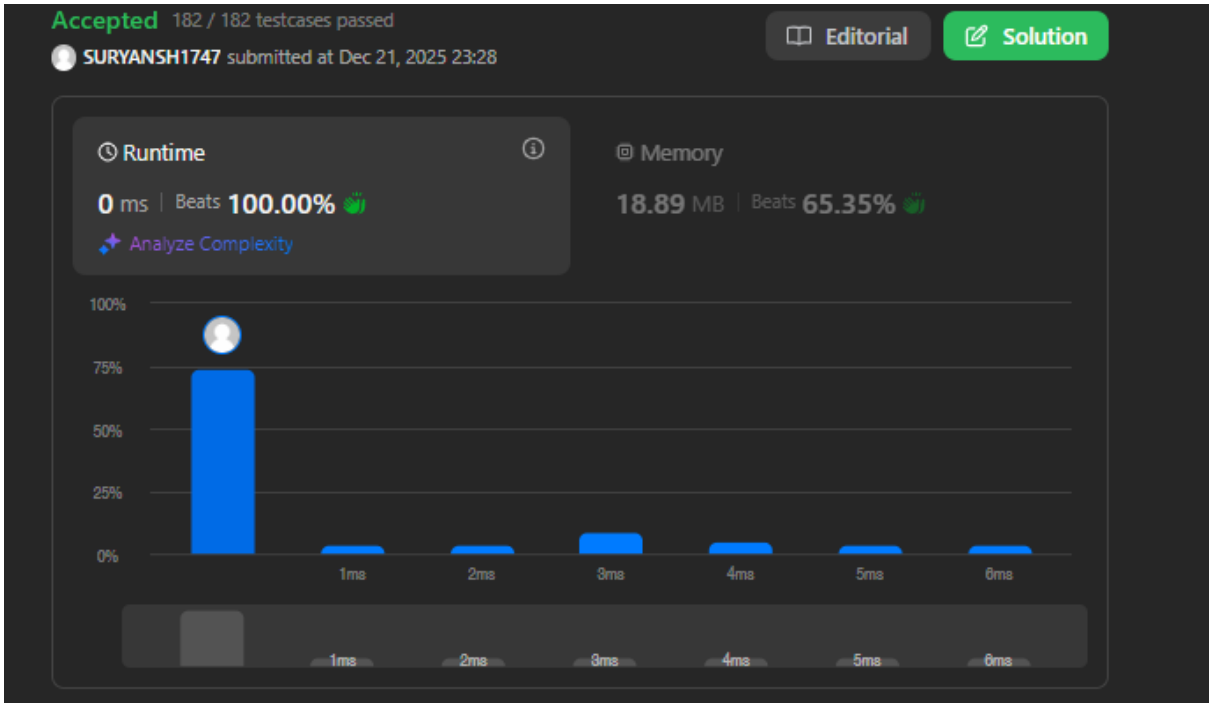
**You are given two binary trees root1 and root2.**

**Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree. Return the merged tree.**

**Note: The merging process must start from the root nodes of both trees.**

```
struct TreeNode* sumtree(struct TreeNode* r1,struct TreeNode* r2){
    if(r1==NULL){return r2;}
    else if(r2==NULL){return r1;}
    r2->val=r1->val+r2->val;
   r2->left = sumtree(r1->left, r2->left);
    r2->right = sumtree(r1->right, r2->right);
    return r2;
 }
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
   if(root1==NULL){return root2;}
   if(root2==NULL){return root1;}
   return sumtree(root1,root2);
}
```

**Ouput:**



**Lab Program 9:**

**a) Write a program to traverse a graph using BFS method.**

```
#include<stdio.h>
#include<stdlib.h>
int graph[20][20];
int n;int v[20];
int queue[200];
void bfs(int start){
    int rear=0;int front=0;
    queue[rear++]=start;
    v[start]=1;
    while(front<rear){
        int p=queue[front++];
        printf("%d ",p);
        for(int i=0;i<n;i++){
            if(graph[p][i]==1&&v[i]==0){
                queue[rear++]=i;
                v[i]=1;
            }
        }
    }
}
int main(){
    printf("Enter the number of vertices: ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix: ");
    for(int i=0;i<n;i++){
```

```
        for(int j=0;j<n;j++){
            scanf("%d",&graph[i][j]);
        }
    }
    for(int i=0;i<n;i++){
        v[i]=0;
    }
 printf("\nBFS TRAVERSAL IS: \n");
    bfs(0);
    return 0;
}
```

**OUTPUT:**

```
PS D:\1BF24CS308 DS\output> & .\'Lab9.exe'
Enter the number of vertices: 5
Enter the adjacency matrix: 0
1
0
1
0
1
0
1
0
1
0
1
0
0
0
1
0
0
0
1
0
1
0
1
0

BFS TRAVERSAL IS:
0 1 3 2 4
PS D:\1BF24CS308 DS\output>
```

**Lab Program 9b:**

**b) Write a program to check whether given graph is connected or not using DFS method.**

```
#include<stdio.h>
#include<stdlib.h>
int graph[20][20];
int n;int v[20];int c=0;
void dfs(int start){
    v[start]=1;c++;
    printf("%d ",start);
    for(int i=0;i<n;i++){
            if(graph[start][i]==1&&v[i]==0){
                v[i]=1;
                dfs(i);
            }
        }
}
int main(){
    printf("Enter the number of vertices: ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix: ");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&graph[i][j]);
        }
    }
    for(int i=0;i<n;i++){
```

```
        v[i]=0;
    }
printf("\nDFS TRAVERSAL\n ");
    dfs(0);
    if(c==n){
        printf("\nCONNECTED ");
    }
    return 0;
}
```

**Output:**

```
PS D:\1BF24CS308 DS> cd 'd:\1BF24CS308 DS\output'
PS D:\1BF24CS308 DS\output> & .\'Lab9b.exe'
Enter the number of vertices: 5
Enter the adjacency matrix: 0
1
0
1
0
1
0
1
0
1
0
1
0
0
0
1
0
0
0
1
0
1
0
1
0

DFS TRAVERSAL
 0 1 2 4 3
CONNECTED
PS D:\1BF24CS308 DS\output>
```

**Lab Program 10:**

**Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.**

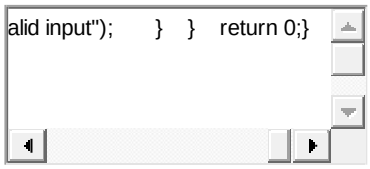
```
#include <stdio.h>

#define MAX 20

int hashTable[MAX];

int m;

void insert(int key)

{

   int index = key % m;

   if (hashTable[index] == -1)

   {

      hashTable[index] = key;

   }

   else

   {

      int i = 1;

      while (hashTable[(index + i) % m] != -1)

      {

         i++;

      }

      hashTable[(index + i) % m] = key;

   }

}
```

```
void display()
{
    printf("\nHash Table:\n");
    for (int i = 0; i < m; i++)
    {
        if (hashTable[i] != -1)
        {
            printf("Address %d: %d\n", i, hashTable[i]);
        }
        else
        {
            printf("Address %d: Empty\n", i);
        }
    }
}

int main()
{
    int n, key;

    printf("Enter size of hash table (m): ");
    scanf("%d", &m);

    printf("Enter number of employee records: ");
    scanf("%d", &n);

    for (int i = 0; i < m; i++)
```

```
    {
        hashTable[i] = -1;
    }

    printf("Enter %d employee keys (4-digit):\n", n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &key);
        insert(key);
    }

    display();
    return 0;
}
```

**Output:**

```
PS D:\1BF24CS308 DS> cd 'd:\1BF24CS308 DS\output'
PS D:\1BF24CS308 DS\output> & .\'Lab program 10.exe'
Enter size of hash table (m): 3
Enter number of employee records: 3
Enter 3 employee keys (4-digit):
1234
3456
6778

Hash Table:
Address 0: 3456
Address 1: 1234
Address 2: 6778
PS D:\1BF24CS308 DS\output>
```


alid input"); } } return 0;}