

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

DATA STRUCTURES (23CS3PCDST)

Submitted by

SRIKRISHNA PEJATHAYA P S (1BM22CS290)

**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
Dec 2023- March 2024**

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 **SRIKRISHNA PEJATHAYA P S (1BM22CS290)**, who is a 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 2023-24. 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.

Prof. Lakshmi Neelima
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	<p>1) Write a program to simulate the working of stack using an array with the following:</p> <ul style="list-style-type: none"> a) Push b) Pop c) Display <p>The program should print appropriate messages for stack overflow, stack underflow.</p> <p>2) WAP to convert a given valid parenthesized infix arithmetic expression to a postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).</p>	5
2	<p>1) Write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate messages for overflow and underflow conditions.</p> <p>2) Write a program to simulate the working of a circular queue using an array. Provide the following operations: insert, delete & display. The program should print appropriate messages for queue empty and queue overflow conditions.</p>	9
3	<p>1) WAP to Implement Singly Linked List with following operations</p> <ul style="list-style-type: none"> a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list. d) Deletion of first element, specified element and last element in list. 	14
4	<p>1) WAP to Implement Single Linked List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.</p> <p>2) WAP to Implement Single Linked List to simulate Stack and Queue Operations.</p>	19

5	1) 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 2) LeetCode - Score Of Parentheses	30
6	1) Write a program a) To construct a binary Search tree. 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. 2) LeetCode - Delete the middle node of a Linked List 3) LeetCode - Odd Even Linked List	36
7	1) Write a program to traverse a graph using the BFS method. 2) Write a program to check whether a given graph is connected or not using the DFS method. 3) LeetCode - Delete Node in a BST 4) LeetCode - Find bottom left tree value	42

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 1:

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.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 3

int stack[MAX_SIZE], top=-1;

void push(int value){
    if (top==MAX_SIZE-1){
        printf("Stack Overflow \n");
    }
    else{
        top++;
        stack[top]=value;
        printf("Pushed %d onto stack \n", value);
    }
}

void pop(){
    if(top==-1){
        printf("Stack underflow \n");
    }
    else{
        printf("Popped %d from stack \n", stack[top]);
        top--;
    }
}

void display(){
    if(top==-1){
        printf("Stack is empty\n");
    }
    else{
        printf("Elements: \n");
        for(int i=0;i<=top;i++){
            printf("%d ", stack[i]);
        }
    }
}
```

```

        printf("\n");
    }
}

int main(){
    int choice, value;
    while (1){
        printf("\nStack Operation Menu: ");
        printf("1. Push 2. Pop 3. Display 4. Exit \n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1: printf("Enter value to push: ");
                    scanf("%d", &value);
                    push(value);
                    break;

            case 2: pop();
                    break;

            case 3: display();
                    break;

            case 4: printf("Exit");
                    exit(0);

            default: printf("Invalid choice!");
        }
    }
    return 0;
}

```

Output:

```
C:\Users\sathw\DS\report\lab2_1_1_stack.exe

Stack Operation Menu: 1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 2
Stack underflow

Stack Operation Menu: 1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 1
Enter value to push: 30
Pushed 30 onto stack

Stack Operation Menu: 1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 1
Enter value to push: 20
Pushed 20 onto stack

Stack Operation Menu: 1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10 onto stack

Stack Operation Menu: 1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 2
Popped 10 from stack

Stack Operation Menu: 1. Push 2. Pop 3. Display 4. Exit
Enter your choice: 3
Elements:
30 20
```

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

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int prec(char c) {
    if(c=='/' || c=='*')
        return 2;
    else if(c=='+' || c=='-')
        return 1;
    else
        return -1;
}

void infixtopostfix(char s[]) {
    char stack[100], result[100];
    int top=-1, result_i=0;
    int len=strlen(s);
```

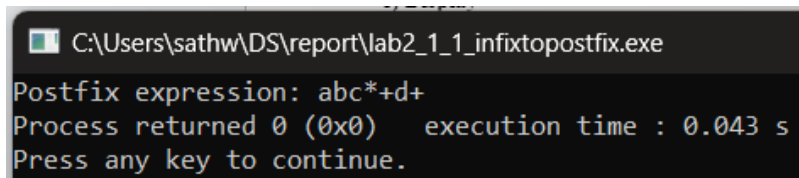
```

for(int i=0; i<len; i++) {
    char c=s[i];
    if(c>='0' && c<='9' || c>='a' && c<='z' || c>='A' && c<='Z') {
        result[result_i++]=c;
    }
    else if(c=='(') {
        stack[++top]=c;
    }
    else if(c==')') {
        while(top>=0 && stack[top]!='(') {
            result[result_i++]=stack[top--];
        }
        top--;
    }
    else {
        while(top>=0 && (prec(s[i])<prec(stack[top])) || prec(s[i])==prec(stack[top])) {
            result[result_i++]=stack[top--];
        }
        stack[++top]=c;
    }
}
while(top>=0) {
    result[result_i++]=stack[top--];
}
result[result_i]='\0';
printf("Postfix expression: %s", result);
}

int main() {
    char exp[]="a+b*c+d";
    infixtopostfix(exp);
    return 0;
}

```

Output:



```

C:\Users\sathw\DS\report\lab2_1_1_infixtopostfix.exe
Postfix expression: abc*+d+
Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.

```


Lab 2:

1) Write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate messages for overflow and underflow conditions.

```
#include<stdio.h>
#define MAX 3

int queue[MAX];
int front=-1, rear=-1;

void insert() {
    int num;
    printf("Enter a number: ");
    scanf("%d",&num);
    if(rear==MAX-1){
        printf("\n Overflow");
    }
    else if(front== -1 && rear== -1){
        front=0;
        rear=0;
    }
    else{
        rear++;
    }
    queue[rear]=num;
}

int delete(){
    int val;
    if(front== -1){
        printf("\n Underflow");
        return -1;
    }
    else if(front==rear){
        val=queue[front];
        front=rear=-1;
    }
    else{
        val=queue[front];
        front++;
    }
    return val;
}
```

```

}

void display(){
    int i;
    if(front==-1 || front>rear){
        printf("\n Queue Empty");
    }
    else{
        for(i=front;i<=rear;i++){
            printf("\n %d", queue[i]);
        }
    }
}

int main(){
    int choice, val;
    while (1){
        printf("\n Linear Queue Operation Menu");
        printf("\n 1. Insert\n 2. Delete\n 3. Display\n 4. Exit");
        printf("\n Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: insert();
                    break;
            case 2: val=delete();
                    printf("\n Deleted value: %d", val);
                    break;
            case 3: display();
                    break;
            case 4: printf("Exit!");
                    exit(0);
            default: printf("Input Invalid!");
        }
    }
    return 0;
}

```

Output:

```
C:\Users\sathw\DS\report\lab3_linearqueue.exe

Linear Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 2

Underflow
Deleted value: -1
Linear Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 30

Linear Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 20

Linear Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 10

Linear Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 2

Deleted value: 30
Linear Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 3

20
10
```

2) Write a program to simulate the working of a circular queue using an array. Provide the following operations: insert, delete & display. The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 3

int queue[MAX];
int front=-1, rear=-1;

void insert(){
    int num;
    printf("Enter a number: ");
    scanf("%d",&num);
    if((rear+1)%MAX==front){
        printf("\n Overflow");
    }
    else if(front== -1 && rear== -1){
        front=rear=0;
        queue[rear]=num;
    }
}
```

```

    else{
        rear=(rear+1)%MAX;
        queue[rear]=num;
    }
}

void delete(){
    if(front==-1){
        printf("\n Underflow");
    }
    else if(front==rear){
        printf("\n Deleted value: %d", queue[front]);
        front=rear=-1;
    }
    else{
        printf("\n Deleted value: %d", queue[front]);
        front=(front+1)%MAX;
    }
}

void display(){
    int i;
    if(front==-1){
        printf("\n Queue Empty");
    }
    else{
        printf("Elements are: \n");
        for(i=front;i!=rear;i=(i+1)%MAX){
            printf("%d\n", queue[i]);
        }
        printf("%d\n", queue[i]);
    }
}

int main(){
    int choice, value;
    while (1){
        printf("\nCircular Queue Operation Menu: ");
        printf("1. Insert 2. Delete 3. Display 4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);

        switch(choice)
        {

```

```

        case 1: insert();
            break;
        case 2: delete();
            break;
        case 3: display();
            break;
        case 4: printf("Exit!");
            exit(0);
        default: printf("Input Invalid!");
    }
}
return 0;
}

```

Output:

```

C:\Users\sathw\DS\report\lab3_8_1_circularqueue.exe
Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 2

Underflow
Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 30

Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 20

Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 10

Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 2

Deleted value: 30
Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter a number: 40

Circular Queue Operation Menu: 1. Insert 2. Delete 3. Display 4. Exit
Enter your choice: 3
Elements are:
20
10
40

```

Lab 3:

1) WAP to Implement Singly Linked List with following operations

- a) Create a linked list.
- b) Insertion of a node at first position, at any position and at end of list.
- c) Display the contents of the linked list.
- d) Deletion of first element, specified element and last element in the list.

```
#include<stdio.h>
#include<stdlib.h>

struct node{
    int data;
    struct node* next;
};

struct node* head= NULL;
void insertatbeginning(int value);
void insertatend(int value);
void insertatmiddle(int value, int pos);
void deletefrombeginning();
void deletefromend();
void deletefrommiddle(int pos);
void displaylist();

int main(){
    int choice, value, pos;
    while(1){
        printf("\nSingle Linked List Operation Menu \n");
        printf("1. Insert at beginning ");
        printf("2. Insert at end ");
        printf("3. Insert at position ");
        printf("4. Delete from beginning ");
        printf("5. Delete from end ");
        printf("6. Delete from position ");
        printf("7. Exit ");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice){
            case 1: printf("\n Enter the value to insert: ");
                    scanf("%d", &value);
                    insertatbeginning(value);
                    break;
            case 2: printf("\n Enter the value to insert: ");
```

```

        scanf("%d", &value);
        insertatend(value);
        break;
case 3: printf("\n Enter the value to insert: ");
        scanf("%d", &value);
        printf("\n Enter the position to insert: ");
        scanf("%d", &pos);
        insertatmiddle(value, pos);
        break;
case 4: deletefrombeginning();
        break;
case 5: deletefromend();
        break;
case 6: printf("\n Enter the position to delete: ");
        scanf("%d", &pos);
        deletefrommiddle(pos);
        break;
case 7: printf("\n Exit!");
        exit(0);
default: printf("Invalid choice!");
}
displaylist();
}
return 0;
}

void insertatbeginning(int value){
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=head;
    head=newnode;
    printf("Insertion at beginning successful! \n");
}

void insertatend(int value){
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=NULL;
    if(head==NULL){
        head=newnode;
    }
    else{
        struct node* temp=head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
    }
}

```

```

        temp->next=newnode;
    }
    printf("Insertion at end successful! \n");
}
void insertatmiddle(int value, int pos){
    if(pos<=0){
        printf("Invalid position");
        return;
    }
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
    newnode->data=value;
    if(pos==1){
        newnode->next=head;
        head=newnode;
    }
    else{
        struct node* temp=head;
        for(int i=1; i<pos-1 && temp!=NULL; i++){
            temp=temp->next;
        }
        if(temp==NULL){
            printf("Invalid position \n");
            free(newnode);
            return;
        }
        newnode->next=temp->next;
        temp->next=newnode;
    }
    printf("Insertion at middle successful! \n");
}
void deletefrombeginning(){
    if(head==NULL){
        printf("List is empty! \n");
        return;
    }
    struct node* temp=head;
    head=head->next;
    free(temp);
    printf("Deletion from beginning successful! \n");
}
void deletefromend(){
    if(head==NULL){
        printf("List is empty! \n");
        return;
    }

```



```

    }
    if(head->next==NULL){
        free(head);
        head=NULL;
        printf("Deletion from end successful! \n");
        return;
    }
    struct node* temp=head;
    while(temp->next->next!=NULL){
        temp=temp->next;
    }
    free(temp->next);
    temp->next=NULL;
    printf("Deletion from end successful! \n");
}

void deletefrommiddle(int pos){
    if(pos<=0 || head==NULL){
        printf("List is empty \n");
        return;
    }
    if(pos==1){
        struct node* temp=head;
        head=head->next;
        free(temp);
        printf("Deletion from middle successful! \n");
        return;
    }
    struct node* temp=head;
    for(int i=1; i<pos-1 && temp!=NULL; i++){
        temp=temp->next;
    }
    if(temp==NULL || temp->next==NULL){
        printf("Invalid Position \n");
        return;
    }
    struct node* temp1=temp->next;
    temp->next=temp->next->next;
    free(temp1);
    printf("Deletion from middle successful! \n");
}

void displaylist(){
    if(head==NULL){
        printf("List is empty \n");
        return;
    }

```

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

Output:

```
C:\Users\sathw\DS\report\lab4_singlelinkedlist.exe

Single Linked List Operation Menu
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete from beginning
5. Delete from end
6. Delete from position
7. Exit
Enter your choice: 4
List is empty!
Cannot display an empty list

Enter your choice: 1

Enter the value to insert: 30
Insertion at beginning successful!
30 NULL

Enter your choice: 2

Enter the value to insert: 10
Insertion at end successful!
30 10 NULL

Enter your choice: 3

Enter the value to insert: 20

Enter the position to insert: 2
Insertion at middle successful!
30 20 10 NULL

Enter your choice: 4
Deletion from beginning successful!
20 10 NULL

Enter your choice: 5
Deletion from end successful!
20 NULL

Enter your choice: 6

Enter the position to delete: 2
Invalid Position
20 NULL
```

Lab 4:

1) WAP to Implement Single Linked 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* head = NULL;

void insertAtBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
}

void insertAtEnd(struct Node** list, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void insertAtMiddle(int data, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        for (int i = 1; i < position - 1; i++) {
            if (temp != NULL) {
                temp = temp->next;
            }
        }
    }
}

```

```

        if (temp != NULL) {
            newNode->next = temp->next;
            temp->next = newNode;
        } else {
            printf("Invalid position for insertion.\n");
        }
    }
}

```

```

void printList() {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

```

```

void sortList() {
    if (head == NULL) return;
    struct Node *i, *j;
    int temp;
    for (i = head; 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 reverseList() {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* nextNode = NULL;
    while (current != NULL) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    head = prev;
}

```

```

}

void concatenateLists(struct Node* second) {
    if (head == NULL) {
        head = second;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = second;
    }
}

int main() {
    int choice, data, position;
    struct Node* list2= NULL;
    printf("\nSingle Linked List Operation Menu");
    printf("\n1.Insert at Beginning\n");
    printf("\n2.Insert at End\n");
    printf("\n3.Insert at Middle\n");
    printf("\n4.Sort List\n");
    printf("\n5.Reverse List \n");
    printf("\n6.Concatenate Lists \n");
    printf("\n0.Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtBeginning(data);
                break;
            case 2:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&list2, data);
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position for insertion: ");

```

```

        scanf("%d", &position);
        insertAtMiddle(data, position);
        break;
case 4:
    sortList();
    printf("List sorted successfully.\n");
    break;
case 5:
    reverseList();
    printf("List reversed successfully.\n");
    break;
case 6:
    insertAtEnd(&list2, 2);
    insertAtEnd(&list2, 4);
    concatenateLists(list2);
    printf("Lists concatenated successfully.\n");
    break;
case 0:
    printf("Exiting program.\n");
    exit(0);
default:
    printf("Invalid choice. Please enter a valid option.\n");
}
printf("Current List: ");
printList();
}

return 0;
}

```

Output:

```
C:\Users\sathw\DS\report\lab5_sortrevcon.exe
Single Linked List Operation Menu
1.Insert at Beginning
2.Insert at End
3.Insert at Middle
4.Sort List
5.Reverse List
6.Concatenate Lists
0.Exit
Enter your choice: 1
Enter data to insert at the beginning: 30
Current List: 30
Enter your choice: 1
Enter data to insert at the beginning: 20
Current List: 20 30
Enter your choice: 1
Enter data to insert at the beginning: 10
Current List: 10 20 30
Enter your choice: 3
Enter data to insert: 40
Enter position for insertion: 2
Current List: 10 40 20 30
Enter your choice: 4
List sorted successfully.
Current List: 10 20 30 40
Enter your choice: 5
List reversed successfully.
Current List: 40 30 20 10
Enter your choice: 6
Lists concatenated successfully.
Current List: 40 30 20 10 2 4
```

2) WAP to Implement Single Linked List to simulate Stack and Queue Operations.

Stack:

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Node {
    int data;
    struct Node* next;
};
```

```
struct Node* top = NULL;
```

```
void push(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Overflow\n");
```



```

        return;
    }

    newNode->data = data;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (top == NULL) {
        printf("Underflow: Stack is empty.\n");
        return -1;
    }

    struct Node* temp = top;
    int poppedData = temp->data;
    top = temp->next;
    free(temp);

    return poppedData;
}

void display() {
    if (top == NULL) {
        printf("Empty.\n");
        return;
    }

    struct Node* current = top;
    printf("Stack: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    int choice, data;

    while (1) {
        printf("\n Stack Operation Menu\n");
        printf("1. Push\n");
        printf("2. Pop\n");

```

```

printf("3. Display\n");
printf("0. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter data to push onto the stack: ");
        scanf("%d", &data);
        push(data);
        break;
    case 2:
        if (top == NULL) {
            printf("Underflow: Stack is empty, cannot pop.\n");
        } else {
            printf("Popped element: %d\n", pop());
        }
        break;
    case 3:
        display();
        break;
    case 0:
        printf("Exiting program.\n");
        exit(0);
    default:
        printf("Invalid choice. Please enter a valid option.\n");
}
}

return 0;
}

```

Output:

```
C:\Users\sathw\DS\report\lab5_stack_sll.exe
Stack Operation Menu
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 2
Underflow: Stack is empty, cannot pop.

Stack Operation Menu
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 1
Enter data to push onto the stack: 30

Stack Operation Menu
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 1
Enter data to push onto the stack: 10

Stack Operation Menu
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 1
Enter data to push onto the stack: 20

Stack Operation Menu
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 2
Popped element: 20

Stack Operation Menu
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 3
Stack: 10 30
```

Queue:

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Node {
    int data;
    struct Node* next;
};
```

```
struct Queue {
    struct Node* front;
    struct Node* rear;
};
```

```
void initializeQueue(struct Queue* queue) {
    queue->front = NULL;
    queue->rear = NULL;
```

```

}

int isEmpty(struct Queue* queue) {
    return (queue->front == NULL);
}

void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Queue Overflow: Unable to enqueue element.\n");
        return;
    }
    newNode->data = data;
    newNode->next = NULL;

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }

    printf("Enqueued: %d\n", data);
}

int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue Underflow: Unable to dequeue element.\n");
        return -1;
    }

    struct Node* temp = queue->front;
    int data = temp->data;

    if (queue->front == queue->rear) {
        queue->front = NULL;
        queue->rear = NULL;
    } else {
        queue->front = queue->front->next;
    }

    free(temp);
    printf("Dequeued: %d\n", data);
}

```

```

    return data;
}

void displayQueue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    struct Node* current = queue->front;
    printf("Queue: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Queue myQueue;
    initializeQueue(&myQueue);
    int choice, data;
    printf("\n Queue Operation Menu\n");
    printf("1. Enqueue (Push)\n");
    printf("2. Dequeue (Pop)\n");
    printf("3. Display Queue\n");
    printf("0. Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&myQueue, data);
                break;
            case 2:
                dequeue(&myQueue);
                break;
            case 3:
                displayQueue(&myQueue);
                break;
            case 0:

```

```

        printf("Exiting program.\n");
        exit(0);
    default:
        printf("Invalid choice. Please enter a valid option.\n");
    }
}

return 0;
}

```

Output:

```

C:\Users\sathw\DS\report\lab5_queue_sll.exe

Queue Operation Menu
1. Enqueue (Push)
2. Dequeue (Pop)
3. Display Queue
0. Exit
Enter your choice: 2
Queue Underflow: Unable to dequeue element.
Enter your choice: 1
Enter data to enqueue: 30
Enqueued: 30
Enter your choice: 1
Enter data to enqueue: 10
Enqueued: 10
Enter your choice: 1
Enter data to enqueue: 20
Enqueued: 20
Enter your choice: 2
Dequeued: 30
Enter your choice: 3
Queue: 10 20

```

Lab 5:

1) 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* prev;
    struct Node* next;
};

struct Node* head=NULL;

struct Node* create(int value) {
    struct Node* newnode=(struct Node*)malloc(sizeof(struct Node));
    newnode->data=value;
    newnode->prev=NULL;
    newnode->next=NULL;
    struct newnode;
};

void display() {
    struct Node* curr=head;
    while(curr!=NULL) {
        printf("%d ->",curr->data);
        curr=curr->next;
    }
    printf("NULL \n");
}

void insertleft(int value, int target) {
    struct Node* newnode=create(value);
    struct Node* curr = head;
    if(curr==NULL || curr->data==target) {
        newnode->next=curr;
        if(curr!=NULL) {
            curr->prev=newnode;
        }
        head=newnode;
        return;
    }

    while(curr!=NULL && curr->data!=target) {
        curr=curr->next;
    }

    if(curr!=NULL) {
        newnode->next=curr;
        newnode->prev=curr->prev;
    }
}

```

```

        if(curr->prev!=NULL) {
            curr->prev->next=newnode;
        }
        else {
            printf("Not found! \n");
            free(newnode);
        }
    }
}

void deletenode(int value) {
    struct Node* curr=head;
    while(curr!=NULL && curr->data!=value) {
        curr=curr->next;
    }

    if(curr!=NULL) {
        if(curr->prev!=NULL) {
            curr->prev->next=curr->next;
        }
        else {
            head=curr->next;
        }

        if(curr->next!=NULL) {
            curr->next->prev=curr->prev;
        }
        free(curr);
        printf("Deleted\n");
    }
    else {
        printf("Not found! \n");
    }
}

int main() {
    int choice, value, target;

    printf("\n-----Double Linked List Operation Menu-----\n");
    printf("1)Create 2)Insert a new node to left 3)Delete a node 4)Exit\n");

    while(1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

```



```

switch(choice) {
case 1:
    printf("Enter first value: ");
    scanf("%d", &value);
    head=create(value);
    display();
    break;
case 2:
    printf("Enter value to be inserted: ");
    scanf("%d", &value);
    printf("Enter target node: ");
    scanf("%d", &target);
    insertleft(value, target);
    display();
    break;
case 3:
    printf("Enter value to be deleted: ");
    scanf("%d", &value);
    deletenode(value);
    display();
    break;
case 4:
    printf("Exiting...\n");
    exit(0);
default:
    printf("Invalid input\n");
}
}
return 0;
}

```

Output:

```
C:\Users\sathw\DS\report\lab6_dll.exe

-----Double Linked List Operation Menu-----
1)Create 2)Insert a new node to left 3)Delete a node 4)Exit
Enter your choice: 1
Enter first value: 30
30 ->NULL
Enter your choice: 2
Enter value to be inserted: 10
Enter target node: 30
10 ->30 ->NULL
Enter your choice: 2
Enter value to be inserted: 20
Enter target node: 30
10 ->20 ->30 ->NULL
Enter your choice: 3
Enter value to be deleted: 10
Deleted
20 ->30 ->NULL
```

2) Leetcode - Score Of Parantheses

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct Stack {
    int* array;
    int top, max;
};

struct Stack* create(int max) {
    struct Stack* stack=(struct Stack*)malloc(sizeof(struct Stack));
    stack->max=max;
    stack->top=-1;
    stack->array=(int*)malloc(stack->max * sizeof(int));
    return stack;
}

void push(struct Stack* stack, int item) {
    stack->array[++stack->top] = item;
}

int pop(struct Stack* stack) {
    if(stack->top== -1) {
        return -1;
    }
}
```

```

    }
    return stack->array[stack->top--];
}

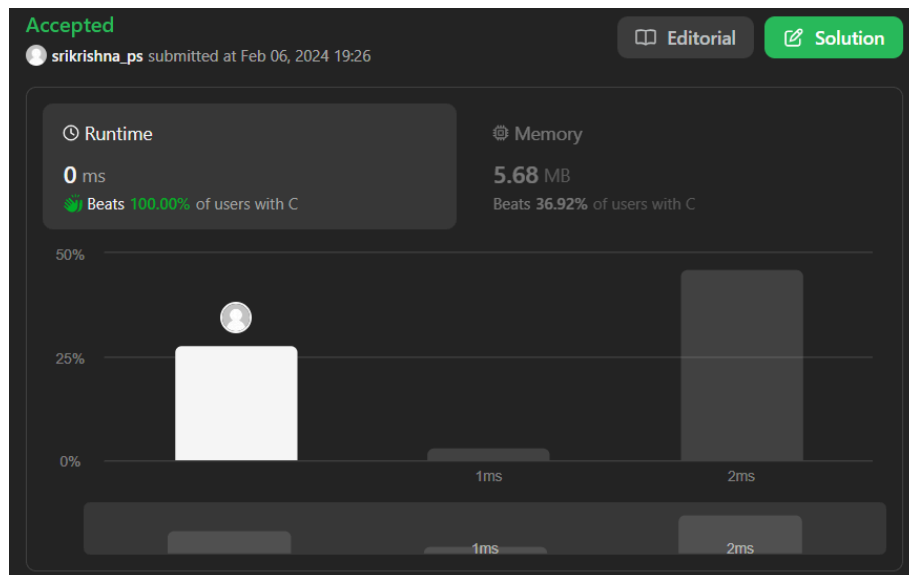
int scoreOfParentheses(char* s) {
    struct Stack* st=create(strlen(s));
    int score=0;

    for(int i=0;i<strlen(s);i++) {
        char ch=s[i];
        if(ch=='(') {
            push(st, score);
            score=0;
        }
        else {
            score=pop(st) + ((2*score)>1 ? (2*score) : 1);
        }
    }
    free(st->array);
    free(st);

    return score;
}

```

Output:



Lab 6:

1) Write a program

- a) To construct a binary Search tree.
- 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* newNode(int data) {
    struct Node* node=(struct Node*)malloc(sizeof(struct Node));
    node->data=data;
    node->left=NULL;
    node->right=NULL;
    return node;
};

struct Node* insert(struct Node* node, int data) {
    if(node==NULL) {
        return newNode(data);
    }
    else if(data < node->data) {
        node->left=insert(node->left,data);
    }
    else if(data > node->data) {
        node->right=insert(node->right,data);
    }
    return node;
};

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

```

void postorder(struct Node* root) {
    if(root!=NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->data);
    }
}

```

```

void preorder(struct Node* root) {
    if(root!=NULL) {
        printf("%d ",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

```

```

void display(struct Node* root) {
    if(root!=NULL) {
        display(root->left);
        printf("%d ",root->data);
        display(root->right);
    }
}

```

```

int main() {
    struct Node* root=NULL;
    int choice, data;

    printf("\n-----Binary Search Tree Traversal Menu-----\n");
    printf("1)Insert 2)Inorder 3)Preorder 4)Postorder 5)Display 0)Exit\n");

    while(1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter a data to insert (0 to exit): ");
                scanf("%d", &data);
                if(data==0) {
                    break;
                }
                root=insert(root,data);

```

```

        break;
    case 2:
        printf("Inorder: \n");
        inorder(root);
        printf("\n");
        break;
    case 3:
        printf("Preorder: \n");
        preorder(root);
        printf("\n");
        break;
    case 4:
        printf("Postorder: \n");
        postorder(root);
        printf("\n");
        break;
    case 5:
        display(root);
        printf("\n");
        break;
    case 0:
        printf("Exiting...\n");
        return 0;
    default:
        printf("Invalid input\n");
    }
}
}

```

Output:

```
C:\Users\sathw\DS\report\lab7_bst.exe
-----Binary Search Tree Traversal Menu-----
1)Insert 2)Inorder 3)Preorder 4)Postorder 5)Display 0)Exit
Enter your choice: 1
Enter a data to insert (0 to exit): 30
Enter your choice: 1
Enter a data to insert (0 to exit): 20
Enter your choice: 1
Enter a data to insert (0 to exit): 10
Enter your choice: 1
Enter a data to insert (0 to exit): 5
Enter your choice: 1
Enter a data to insert (0 to exit): 40
Enter your choice: 1
Enter a data to insert (0 to exit): 35
Enter your choice: 2
Inorder:
5 10 20 30 35 40
Enter your choice: 3
Preorder:
30 20 10 5 40 35
Enter your choice: 4
Postorder:
5 10 20 35 40 30
Enter your choice: 5
5 10 20 30 35 40
```

2) LeetCode - Delete the middle node of a Linked List

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */

struct ListNode* deleteMiddle(struct ListNode* head) {
    if (!head || !head->next) {
        return NULL;
    }
    int count = 0;
    struct ListNode* curr = head;
    while (curr) {
        count++;
        curr = curr->next;
    }

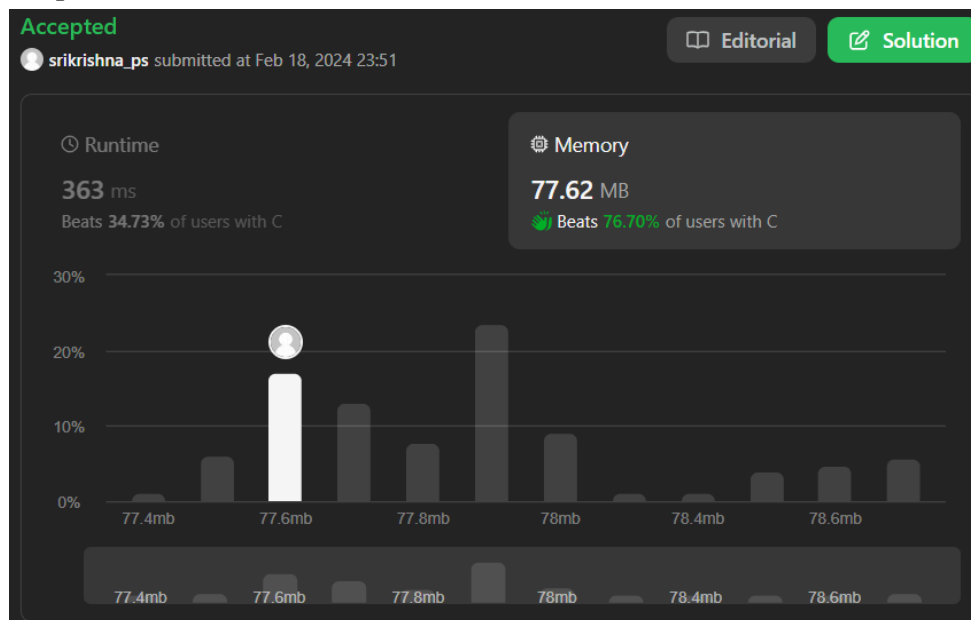
    int middle = count / 2;
```

```

curr = head;
if (middle == 0) {
    head = head->next;
    free(curr);
    return head;
}
for (int i = 0; i < middle - 1; i++) {
    curr = curr->next;
}
struct ListNode* temp = curr->next;
curr->next = curr->next->next;
free(temp);
return head;
}

```

Output:



3) LeetCode - Odd Even Linked List

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* oddEvenList(struct ListNode* head) {
    if (!head || !head->next) {

```



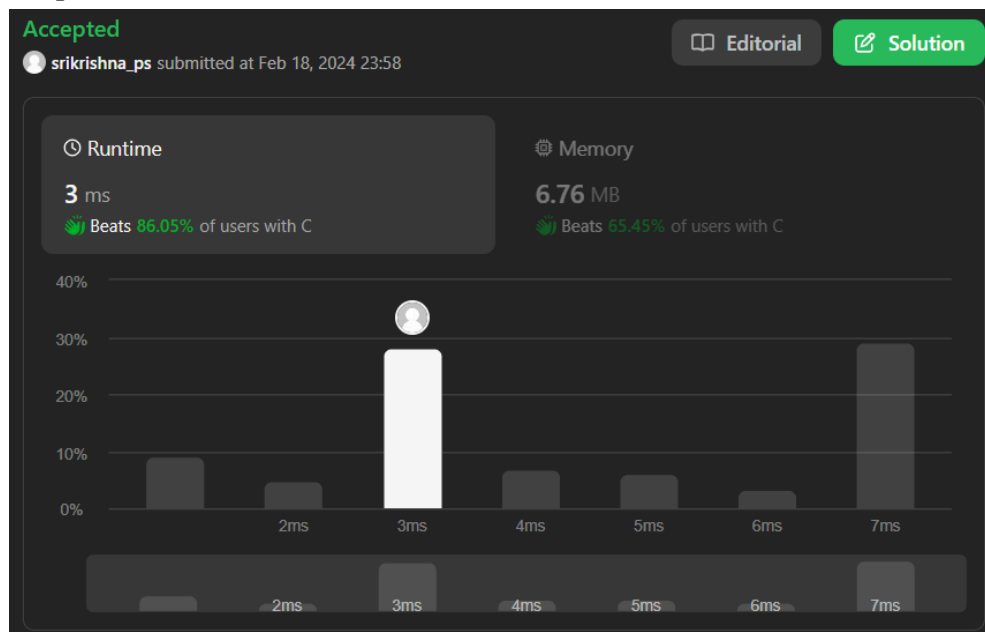
```

    return head;
}
struct ListNode* oddHead = (struct ListNode*)malloc(sizeof(struct ListNode));
struct ListNode* evenHead = (struct ListNode*)malloc(sizeof(struct ListNode));
struct ListNode* odd = oddHead;
struct ListNode* even = evenHead;
struct ListNode* curr = head;
while (curr) {
    odd->next = curr;
    odd = odd->next;
    curr = curr->next;
    if (curr) {
        even->next = curr;
        even = even->next;
        curr = curr->next;
    }
}
even->next = NULL;
odd->next = evenHead->next;
struct ListNode* result = oddHead->next;
free(oddHead);
free(evenHead);

return result;
}

```

Output:



Lab 7:

1) Write a program to traverse a graph using the BFS method.

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_NODES 100

int queue[MAX_NODES];
int front = -1, rear = -1;

void enqueue(int node) {
    if (rear == MAX_NODES - 1) {
        printf("Queue is full.\n");
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = node;
    }
}

int dequeue() {
    int node;
    if (front == -1) {
        printf("Queue is empty.\n");
        return -1;
    } else {
        node = queue[front];
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        return node;
    }
}

bool isEmpty() {
    return front == -1;
}
```

```

void BFS(int adjMatrix[MAX_NODES][MAX_NODES], int nodes, int startNode) {
    bool visited[MAX_NODES] = {false};

    visited[startNode] = true;
    enqueue(startNode);

    while (!isEmpty()) {
        int currentNode = dequeue();
        printf("%d ", currentNode);

        for (int i = 0; i < nodes; i++) {
            if (adjMatrix[currentNode][i] == 1 && !visited[i]) {
                visited[i] = true;
                enqueue(i);
            }
        }
    }
}

int main() {
    int nodes, adjMatrix[MAX_NODES][MAX_NODES];

    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            scanf("%d", &adjMatrix[i][j]);
        }
    }

    int startNode;
    printf("Enter the starting node for BFS: ");
    scanf("%d", &startNode);

    printf("BFS Traversal starting from node %d: ", startNode);
    BFS(adjMatrix, nodes, startNode);

    return 0;
}

```

Output:

```
C:\Users\sathw\DS\report\lab8_bfs.exe
Enter the number of nodes: 7
Enter the adjacency matrix:
0 0 0 0 0 1 1
0 1 1 0 0 0 0
1 0 0 0 0 0 1
1 1 0 0 0 0 0
0 0 1 1 0 0 0
0 0 0 1 1 0 0
0 0 0 0 1 1 0
Enter the starting node for BFS: 0
BFS Traversal starting from node 0: 0 5 6 3 4 1 2
Process returned 0 (0x0)   execution time : 78.028 s
Press any key to continue.
```

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

```
#include <stdio.h>
```

```
int a[20][20], s[20], n;
```

```
void dfs(int v) {
    int i;
    s[v] = 1;
    printf("%d ", v);

    for (i = 1; i <= n; i++) {
        if (a[v][i] && !s[i]) {
            dfs(i);
        }
    }
}
```

```
int main() {
    int i, j, count = 0;

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

    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++) {
        s[i] = 0;
        for (j = 1; j <= n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
}
```

```

    }

    printf("DFS Traversal starting from node 1: ");
    dfs(1);

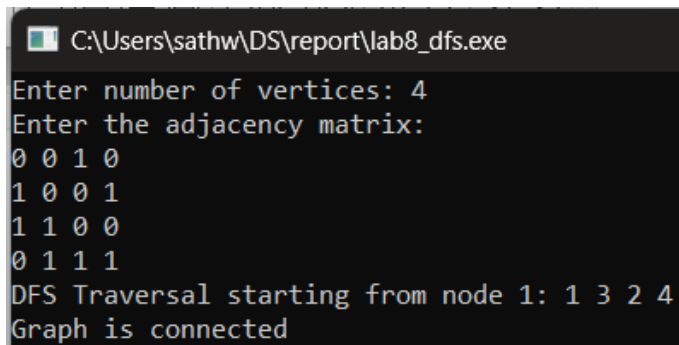
    for (i = 1; i <= n; i++) {
        if (s[i]) {
            count++;
        }
    }

    if (count == n) {
        printf("\nGraph is connected\n");
    } else {
        printf("\nGraph is not connected\n");
    }

    return 0;
}

```

Output:



```

C:\Users\sathw\DS\report\lab8_dfs.exe
Enter number of vertices: 4
Enter the adjacency matrix:
0 0 1 0
1 0 0 1
1 1 0 0
0 1 1 1
DFS Traversal starting from node 1: 1 3 2 4
Graph is connected

```

3) LeetCode - Delete Node in a BST

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

struct TreeNode* smallest(struct TreeNode* root){
    struct TreeNode * cur=root;

```

```

while(cur->left!=NULL) {
    cur=cur->left;
}
return cur;
}

struct TreeNode* deleteNode(struct TreeNode* root, int key){
    if(root==NULL){
        return root;
    }

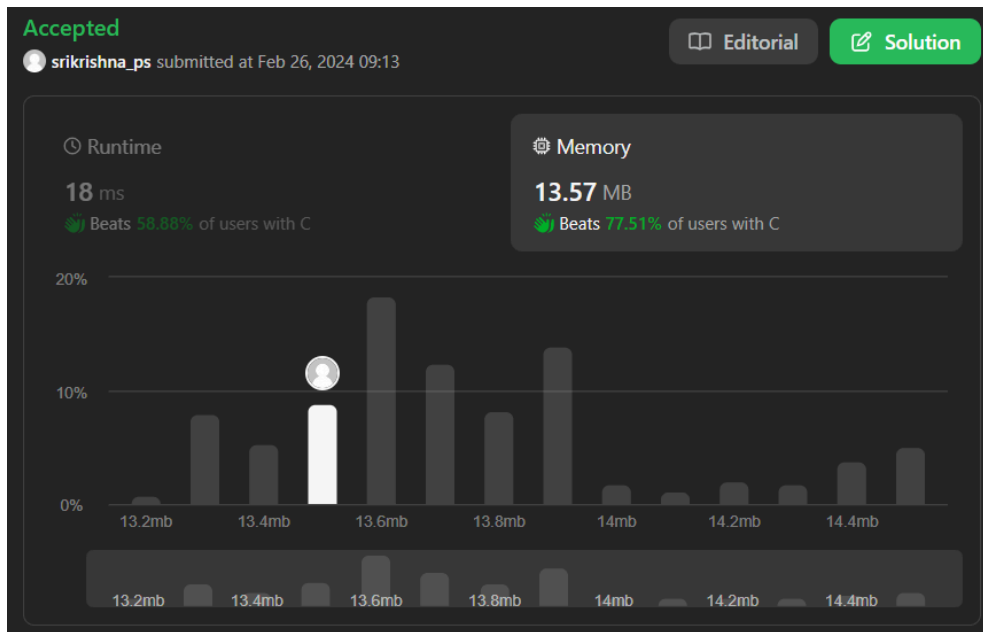
    if(key < root->val){
        root->left = deleteNode(root->left,key);
    }

    else if(key > root->val) {
        root->right=deleteNode(root->right,key);
    }

    else {
        if (root->left == NULL) {
            struct TreeNode *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL){
            struct TreeNode *temp = root->left;
            free(root);
            return temp;
        }
        struct TreeNode *temp = smallest(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, root->val);
    }
    return root;
}

```

Output:



4) LeetCode - Find bottom left tree value

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

int findBottomLeftValue(struct TreeNode* root) {
    struct TreeNode* queue[10000];
    int front = 0, rear = 0, levelSize = 0, leftmostValue = 0;

    if (root == NULL) {
        return 0;
    }

    queue[rear++] = root;

    while (front < rear) {
        levelSize = rear - front;

        for (int i = 0; i < levelSize; i++) {
            struct TreeNode* currentNode = queue[front++];

```

```

    if (i == 0) {
        leftmostValue = currentNode->val;
    }

    if (currentNode->left != NULL) {
        queue[rear++] = currentNode->left;
    }

    if (currentNode->right != NULL) {
        queue[rear++] = currentNode->right;
    }
}

return leftmostValue;
}

```

Output:

