VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Mohammed Farhaan (1BM24CS169)

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 Mohammed Farhaan **(1BM24CS169)**, 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.

Prof.M.Lakshmi Neelima
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.

```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
	int item;
	if(*top==STACK_SIZE-1)
		printf("Stack overflow\n");
	else
	{
		printf("\nEnter an item :");
		scanf("%d",&item);
		(*top)++;
		st[*top]=item;
	}
}
void pop(int st[],int *top)
{
	if(*top==-1)
		printf("Stack underflow\n");
	else
	{
		printf("\n%d item was deleted",st[(*top)--]);
	}
}
void display(int st[],int *top)
{
	int i;
	if(*top==-1)
		printf("Stack is empty\n");
	for(i=0;i<=*top;i++)
		printf("%d\t",st[i]);
}
void main()
{
	int st[10],top=-1, c,val_del;
```

```
	while(1)
	{
		printf("\n1. Push\n2. Pop\n3. Display\n");
		printf("\nEnter your choice :");
		scanf("%d",&c);
		switch(c)
		{
			case 1: push(st,&top);
				break;
			case 2: pop(st,&top);
				break;
			case 3: display(st,&top);
				break;
			default: printf("\nInvalid choice!!!");
				exit(0);
		}
	}
}
```

Output:

```
Enter stack size:2

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 1
Pushed 1 onto the stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 2
Pushed 2 onto the stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 1 2
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 3
Stack overflow

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped element: 2

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped element: 1

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

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
4
Exit
Process returned 0 (0x0)   execution time : 23.870 s
Press any key to continue.
```

Lab program 2:

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

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>


char stack[100];

int top = -1;


void push(char c) {

    if (top == 99) {

        printf("Stack overflow\n");

        return;

    }

    stack[++top] = c;

}


int precedence(char op) {

    switch (op) {

        case '+':

        case '-': return 1;

        case '*':

        case '/': return 2;

        case '^': return 3;

        default: return -1;

    }

}
```

```
char pop() {
    if (top == -1) {
        return '0';
    } else {
        return stack[top--];
    }
}

int isOperand(char c) {
    return isalnum(c);
}

int main() {
    char infix[100];
    char postfix[100];

    printf("Enter infix expression: ");
    fgets(infix, sizeof(infix), stdin);
    infix[strcspn(infix, "\n")] = 0;

    int len = strlen(infix);
    int k = 0;

    for (int i = 0; i < len; i++) {
        char c = infix[i];

        if (isOperand(c)) {
```

```
        postfix[k++] = c;
      } else if (c == '(') {
        push(c);
      } else if (c == ')') {
        while (top != -1 && stack[top] != '(')
          postfix[k++] = pop();
        pop();
      } else {
        while (top != -1 && precedence(stack[top]) >= precedence(c)) {
          if (c == '^' && stack[top] == '^') break;
          postfix[k++] = pop();
        }
        push(c);
      }
    }

    while (top != -1) {
      postfix[k++] = pop();
    }

    postfix[k] = '\0';

    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

Output:

```
C:\Users\Admin\Documents\1bm24cs214\infixtopostfix.exe

Enter infix expression: a+b*(c^d-e)/f)
Postfix expression: abcd^e-*f/+

Process returned 0 (0x0)   execution time : 21.948 s
Press any key to continue.
```

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**

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 size 2
int front=-1,rear=-1;
int queue[size];
void enque(int value){
   if(rear==size-1)
      printf("queue is full");
   else{
      if(front==-1)
         front=0;
      queue[++rear
      ]=value;
   }
}
void deque(){
   if(front==-1||front>rear)
         printf("queue is empty");
   else
      printf("%d",queue[front++]);
}
void display(){
   if(front==-1)
      printf("queue is empty");
   else{
      for(int i=front;i<=rear;i++)
         printf("%d ",queue[i]);
   }
}
int main(){
   int value,choice;
   while(1){
   printf(" \n1.enque\n 2.deque\n 3.display\n 4.exit\n");
   printf("enter choice:");
   scanf("%d",&choice);
   switch(choice){
   case 1:
      printf("enter value:");
      scanf("%d",&value);
      enque(value);
      break;
   case 2:
```

```
        deque();
        break;
    case 3:
        display();
        break;
    case 4:
        return 0;
    default:
        printf("invalid choice");
    }
    }
}
```

Output:

```
1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:2

1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:2

1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:4
queue is full
1.enque
 2.deque
 3.display
 4.exit
enter choice:3
2 2
1.enque
 2.deque
 3.display
 4.exit
enter choice:2
2
1.enque
 2.deque
 3.display
 4.exit
enter choice:2
2
1.enque
 2.deque
 3.display
 4.exit
enter choice:2
queue is empty
1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:4

1.enque
 2.deque
 3.display
 4.exit
enter choice:3
4
```

```
#include<stdio.h>
#define size 2
int front=-1,rear=-1;
int queue[size];
void enque(int value){
    if((rear+1)%size==front)
        printf("queue is full");
    else{
        if(front==-1)
            front=0;
        rear=(rear+1)%size;
        queue[rear]=value;
    }
}
void deque(){
    if(front==-1)
            printf("queue is empty");
    else{
    printf("%d",queue[front]);
    if(front==rear){
        front=-1;
        rear=-1;
    }
    else
        front=(front+1)%size;
    }
}
void display() {
    if (front == -1) {
        printf("queue is empty");
    } else {
        int i = front;
        while(1){
            printf("%d ", queue[i]);
            if (i == rear)
                break;
            i=(i+1)%size;
    }
}
}
int main(){
    int value,choice;
    while(1){
    printf(" \n1.enque\n 2.deque\n 3.display\n 4.exit\n");
    printf("enter choice:");
    scanf("%d",&choice);
    switch(choice){
    case 1:
        printf("enter value:");
```

```
            scanf("%d",&value);
            enque(value);
            break;
        case 2:
            deque();
            break;
        case 3:
            display();
            break;
        case 4:
            return 0;
        default:
            printf("invalid choice");
        }
        }
}
```

Output:

```
1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:2

1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:3

1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:4
queue is full
1.enque
 2.deque
 3.display
 4.exit
enter choice:3
2 3
1.enque
 2.deque
 3.display
 4.exit
enter choice:2
2
1.enque
 2.deque
 3.display
 4.exit
enter choice:2
3
1.enque
 2.deque
 3.display
 4.exit
enter choice:2
queue is empty
1.enque
 2.deque
 3.display
 4.exit
enter choice:1
enter value:2
queue is full
1.enque
 2.deque
 3.display
 4.exit
enter choice:4

Process returned 0 (0x0)   execution time : 17.852 s
Press any key to continue.
```

Lab Program 4:

WAP to Implement Singly Linked List with following operations 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 create(int n) {
    struct node *newnode, *temp;
    int val;
    for (int i = 0; i < n; i++) {
        newnode = malloc(sizeof(struct node));
        scanf("%d", &val);
        newnode->data = val;
        newnode->next = NULL;

        if (head == NULL)
            head = newnode;
        else {
            temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newnode;
        }
    }
}

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

```
void insert_begin(int val) {
    struct node *newnode = malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = head;
    head = newnode;
}

void insert_end(int val) {
    struct node *newnode = malloc(sizeof(struct node));
    struct node *temp = head;
    newnode->data = val;
    newnode->next = NULL;

    if (head == NULL)
        head = newnode;
    else {
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newnode;
    }
}

void insert_pos(int val, int pos) {
    struct node *newnode = malloc(sizeof(struct node));
    newnode->data = val;

    if (pos == 1) {
        newnode->next = head;
        head = newnode;
        return;
    }

    struct node *temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;

    if (temp == NULL) {
        printf("Invalid position\n");
        return;
    }

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

int main() {
    int choice, n, val, pos;

    while (1) {
```

```
        printf("\n1.Create\n2.Insert Begin\n3.Insert End\n4.Insert
Position\n5.Display\n6.Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scanf("%d", &n);
                create(n);
                break;
            case 2:
                scanf("%d", &val);
                insert_begin(val);
                break;
            case 3:
                scanf("%d", &val);
                insert_end(val);
                break;
            case 4:
                scanf("%d%d", &val, &pos);
                insert_pos(val, pos);
                break;
            case 5:
                display();
                break;
            case 6:
                exit(0);
        }
    }
}
```

```
1.Create
2.Insert Begin
3.Insert End
4.Insert Position
5.Display
6.Exit
1
3
1
2
3

1.Create
2.Insert Begin
3.Insert End
4.Insert Position
5.Display
6.Exit
5
1 2 3

1.Create
2.Insert Begin
3.Insert End
4.Insert Position
5.Display
6.Exit
2
4
```
Output:

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 create(int n) {
    struct node *newnode, *temp;
    int val;
    for (int i = 0; i < n; i++) {
        newnode = malloc(sizeof(struct node));
        scanf("%d", &val);
        newnode->data = val;
        newnode->next = NULL;

        if (head == NULL)
            head = newnode;
        else {
            temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newnode;
        }
    }
}

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

```
void delete_begin() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct node *temp = head;
    head = head->next;
    printf("Deleted: %d\n", temp->data);
    free(temp);
}

void delete_end() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    if (head->next == NULL) {
        printf("Deleted: %d\n", head->data);
        free(head);
        head = NULL;
        return;
    }

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

void delete_pos(int pos) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    if (pos == 1) {
        delete_begin();
        return;
    }

    struct node *temp = head, *prev = NULL;
    for (int i = 1; i < pos && temp != NULL; i++) {
        prev = temp;
        temp = temp->next;
    }
```

```
    if (temp == NULL) {
        printf("Invalid position\n");
        return;
    }

    prev->next = temp->next;
    printf("Deleted: %d\n", temp->data);
    free(temp);
}

int main() {
    int choice, n, pos;

    while (1) {
        printf("\n1.Create\n2.Delete Begin\n3.Delete End\n4.Delete
Position\n5.Display\n6.Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scanf("%d", &n);
                create(n);
                break;
            case 2:
                delete_begin();
                break;
            case 3:
                delete_end();
                break;
            case 4:
                scanf("%d", &pos);
                delete_pos(pos);
                break;
            case 5:
                display();
                break;
            case 6:
                exit(0);
        }
    }
}
```

Output:

```
1.Create
2.Delete Begin
3.Delete End
4.Delete Position
5.Display
6.Exit
1
6
1 2 3 4 5 6

1.Create
2.Delete Begin
3.Delete End
4.Delete Position
5.Display
6.Exit
5
1 2 3 4 5 6

1.Create
2.Delete Begin
3.Delete End
4.Delete Position
5.Display
6.Exit
2
Deleted: 1
```

```
1.Create
2.Delete Begin
3.Delete End
4.Delete Position
5.Display
6.Exit
3
Deleted: 6

1.Create
2.Delete Begin
3.Delete End
4.Delete Position
5.Display
6.Exit
4 2
Deleted: 3

1.Create
2.Delete Begin
3.Delete End
4.Delete Position
5.Display
6.Exit
5
2 4 5
```

Lab program 6:

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

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

Program a)

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



typedef struct node {
    int val;
    struct node *next;
} node_t;

node_t *create_node(int val) {
    node_t *new_node = (node_t *)malloc(sizeof(node_t));
    if (new_node == NULL)
        return NULL;
    new_node->val = val;
    new_node->next = NULL;
    return new_node;
}

node_t *create_list(int n) {
    node_t *head = NULL, *temp = NULL;
    int val;
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        node_t *new_node = create_node(val);
        if (head == NULL) {
            head = new_node;
            temp = head;
        } else {
            temp->next = new_node;
            temp = new_node;
        }
    }
    return head;
}

void print_list(node_t *head) {
    node_t *current = head;
    while (current != NULL) {
```

```
        printf("%d -> ", current->val);
        current = current->next;
    }
    printf("NULL\n");
}

node_t *concatenate_lists(node_t *head1, node_t *head2) {if (head1 == NULL) return head2;
    node_t *current = head1;
    while (current->next != NULL)
        current = current->next;
    current->next = head2;
    return head1;
}

node_t *reverse_list(node_t *head) {
    node_t *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

node_t *sort_list(node_t *head) {
    int swapped;
    node_t *ptr1;
    node_t *lptr = NULL;

    if (head == NULL)
        return head;



    do {
        swapped = 0;
        ptr1 = head;
        while (ptr1->next != lptr) {
            if (ptr1->val > ptr1->next->val) {
                int temp = ptr1->val;
                ptr1->val = ptr1->next->val;
                ptr1->next->val = temp;
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);

    return head;
```

```
}

int main() {
    int n1, n2;
    printf("Enter number of nodes for list 1: ");
    scanf("%d", &n1);
    node_t *head1 = create_list(n1);

    printf("Enter number of nodes for list 2: ");
    scanf("%d", &n2);
    node_t *head2 = create_list(n2);

    printf("List 1: ");
    print_list(head1);
    printf("List 2: ");
    print_list(head2);

    head1 = concatenate_lists(head1, head2);
    printf("After Concatenation: ");
    print_list(head1);

    head1 = reverse_list(head1);
    printf("After Reversal: ");
    print_list(head1);

    head1 = sort_list(head1);
    printf("After Sorting: ");
    print_list(head1);

    return 0;
}
```

Output:

```
Enter number of nodes for list 1: 3
1 2 3
Enter number of nodes for list 2: 3
4 5 6
List 1: 1 -> 2 -> 3 -> NULL
List 2: 4 -> 5 -> 6 -> NULL
After Concatenation: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
After Reversal: 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> NULL
After Sorting: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
```

Program 6b) STacks

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

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

Node* top = NULL;

void push(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (top == NULL) {
        printf("Stack empty\n");
        return -1;
    }
    int val = top->data;
    Node* temp = top;
    top = top->next;
    free(temp);
    return val;
}

int peek() {
    if (top == NULL) {
        printf("Stack empty\n");
        return -1;
    }
    return top->data;
}

void display() {
    if (top == NULL) {
        printf("Stack empty\n");
        return;
    }
    Node* temp = top;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");}
```

```
int main() {
    int choice, val;

    while (1) {
        printf("\n1.Push\n2.Pop\n3.Peek\n4.Display\n5.Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scanf("%d", &val);
                push(val);
                break;
            case 2:
                printf("%d\n", pop());
                break;
            case 3:
                printf("%d\n", peek());
                break;
            case 4:
                display();
                break;
            case 5:
                exit(0);
        }
    }
}
```

```
1.Push
2.Pop
3.Peek
4.Display
5.Exit
1
2

1.Push
2.Pop
3.Peek
4.Display
5.Exit
1
4

1.Push
2.Pop
3.Peek
4.Display
5.Exit
1
5

1.Push
2.Pop
3.Peek
4.Display
5.Exit
4
5 4 2
```

Output:

```
1.Push
2.Pop
3.Peek
4.Display
5.Exit
2
5

1.Push
2.Pop
3.Peek
4.Display
5.Exit
3
4

1.Push
2.Pop
3.Peek
4.Display
5.Exit
4
4 2

1.Push
2.Pop
3.Peek
4.Display
5.Exit
5
```

Program 6b) Queue

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

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

Node* front = NULL;
Node* rear = NULL;

void enqueue(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

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

int dequeue() {
    if (front == NULL) {
        printf("Queue empty\n");
        return -1;
    }
    int val = front->data;
    Node* temp = front;
    front = front->next;
    if (front == NULL)
        rear = NULL;
    free(temp);
    return val;
}

int peek() {
    if (front == NULL) {
        printf("Queue empty\n");
        return -1;
    }
    return front->data;
}

void display() {
```

```
    if (front == NULL) {
        printf("Queue empty\n");
        return;
    }
    Node* temp = front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, val;

    while (1) {
        printf("\n1.Enqueue\n2.Dequeue\n3.Peek\n4.Display\n5.Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scanf("%d", &val);
                enqueue(val); break;
            case 2:
                printf("%d\n", dequeue());
                break;
            case 3:
                printf("%d\n", peek());
                break;
            case 4:
                display();
                break;
            case 5:
                exit(0);
        }
    }
}
```

Output:

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

1.Enqueue
2.Dequeue
3.Peek
4.Display
5.Exit
1
2

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

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

1.Enqueue
2.Dequeue
3.Peek
4.Display
5.Exit
3
2

1.Enqueue
2.Dequeue
3.Peek
4.Display
5.Exit
4
2 3

1.Enqueue
2.Dequeue
3.Peek
4.Display
5.Exit
5
```

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>

typedef struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
} Node;

Node* head = NULL;

void create(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void insertLeft(int target, int data) {
    if (head == NULL) {
        printf("List empty\n");
        return;
    }

    Node* temp = head;
    while (temp != NULL && temp->data != target)
        temp = temp->next;

    if (temp == NULL) {
        printf("Not found\n");
        return;
    }
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = temp;
    newNode->prev = temp->prev;

    if (temp->prev != NULL)
        temp->prev->next = newNode;
    else
        head = newNode;

    temp->prev = newNode;
}

void deleteNode(int data) {
    if (head == NULL) {
        printf("List empty\n");
        return;
    }

    Node* temp = head;
    while (temp != NULL && temp->data != data)
        temp = temp->next;

    if (temp == NULL) {
        printf("Not found\n");
        return;
    }

    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        head = temp->next;

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

    free(temp);
}

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

    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
```

```
    }
    printf("\n");
}

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

    while (1) {
        printf("\n1.Create\n2.Insert Left\n3.Delete\n4.Display\n5.Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scanf("%d", &val);
                create(val);
                break;
            case 2:
                scanf("%d%d", &target, &val);
                insertLeft(target, val);
                break;
            case 3:
                scanf("%d", &val);
                deleteNode(val);
                break;
            case 4:
                display();
                break;
            case 5:
                exit(0);
        }
    }
}
```

```
1.Create
2.Insert Left
3.Delete
4.Display
5.Exit
1
1

1.Create
2.Insert Left
3.Delete
4.Display
5.Exit
1
|
2

1.Create
2.Insert Left
3.Delete
4.Display
5.Exit
1
3

1.Create
2.Insert Left
3.Delete
4.Display
```
Output:

```
3.Delete
4.Display
5.Exit
4
1 2 3

1.Create
2.Insert Left
3.Delete
4.Display
5.Exit
3
1

1.Create
2.Insert Left
3.Delete
4.Display
5.Exit
2
3
4

1.Create
2.Insert Left
3.Delete
4.Display
5.Exit
4
2 4 3
```

Lab program 8:

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, *right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insertBST(struct Node* root, int data) {
    if (root == NULL)
        return createNode(data);

    if (data < root->data)
        root->left = insertBST(root->left, data);
    else
        root->right = insertBST(root->right, data);

    return root;
}

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

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

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

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

    while (1) {
        printf("\n1.Insert\n2.Inorder\n3.Preorder\n4.Postorder\n5.Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scanf("%d", &val);
                root = insertBST(root, val);
                break;

            case 2:
                inorder(root);
                printf("\n");
                break;

            case 3:
                preorder(root);
                printf("\n");
                break;

            case 4:
                postorder(root);
                printf("\n");
                break;

            case 5:
                exit(0);
        }
    }
}
```

Output:

```
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
1
5

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
1
3

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
1
2

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
1
7
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
1
6

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
2
2 3 5 6 7

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
3
5 3 2 7 6

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Exit
4
2 3 6 7 5
```

Lab program 9:

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

```
#include <stdio.h>

int g[100][100], vis[100], n;

int main() {
    int e, u, v, start;
    int q[100], f = 0, r = 0;

    printf("Enter number of nodes and edges: ");
    scanf("%d %d", &n, &e);

    printf("Enter each edge (u v):\n");
    for (int i = 0; i < e; i++) {
        scanf("%d %d", &u, &v);
        g[u][v] = g[v][u] = 1;
    }

    printf("Enter start node: ");
    scanf("%d", &start);

    printf("BFS Order: ");

    q[r++] = start;
    vis[start] = 1;

    while (f < r) {
        u = q[f++];
        printf("%d ", u);
        for (v = 0; v < n; v++) {
            if (g[u][v] && !vis[v]) {
                vis[v] = 1;
                q[r++] = v;
            }
        }
    }

    return 0;
}
```

Output:

```
Enter number of nodes and edges: 5 5
Enter each edge (u v):
0 1
0 2
1 3
1 4
3 4
Enter start node: 0
BFS Order: 0 1 2 3 4
```

```c
#include <stdio.h>

int g[100][100], vis[100], n;

int main() {
    int e, u, v, start;
    int stack[100], top = -1;

    printf("Enter number of nodes and edges: ");
    scanf("%d %d", &n, &e);

    printf("Enter each edge (u v):\n");
    for (int i = 0; i < e; i++) {
        scanf("%d %d", &u, &v);
        g[u][v] = g[v][u] = 1;
    }

    printf("Enter start node: ");
    scanf("%d", &start);

    printf("DFS Order: ");

    stack[++top] = start;

    while (top != -1) {
        u = stack[top--];
        if (!vis[u]) {
            vis[u] = 1;
            printf("%d ", u);
            for (v = n - 1; v >= 0; v--)
                if (g[u][v] && !vis[v])
                    stack[++top] = v;
        }
    }
    return 0;
}
```

Output:

```
Enter number of nodes and edges: 5 5
Enter each edge (u v):
0 1
0 2
1 3
1 4
3 4
Enter start node: 0
DFS Order: 0 1 3 4 2
```

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 M 100

typedef struct {
    int key;
    int used;
} Record;

int hash(int k) {
    return k % M;
}

void insert(Record ht[], int k) {
    int i = hash(k);
    int start = i;
    while (ht[i].used) {
        i = (i + 1) % M;
        if (i == start)
            return;
    }
    ht[i].key = k;
    ht[i].used = 1;
}

int search(Record ht[], int k) {
    int i = hash(k);
    int start = i;
    while (ht[i].used) {
        if (ht[i].key == k)
            return i;
        i = (i + 1) % M;
        if (i == start)
            break;
    }
    return -1;
}

int main() {
```

```
    Record ht[M];
    int n, k, i, pos, choice;

    for (i = 0; i < M; i++)
        ht[i].used = 0;

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

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

    do {
        printf("Enter key to search: ");
        scanf("%d", &k);
        pos = search(ht, k);
        if (pos == -1)
            printf("Key not found\n");
        else
            printf("Key found at address %02d\n", pos);

        printf("Search another key? (1-Yes / 0-No): ");
        scanf("%d", &choice);
    } while (choice == 1);

    return 0;
}
```

Output:

```
Enter number of employee records: 5
Enter 4-digit key: 1234
Enter 4-digit key: 2345
Enter 4-digit key: 3456
Enter 4-digit key: 4567
Enter 4-digit key: 5678
Enter key to search: 1235
Key not found
Search another key? (1-Yes / 0-No): 1
Enter key to search: 4567
Key found at address 67
Search another key? (1-Yes / 0-No): 0
```

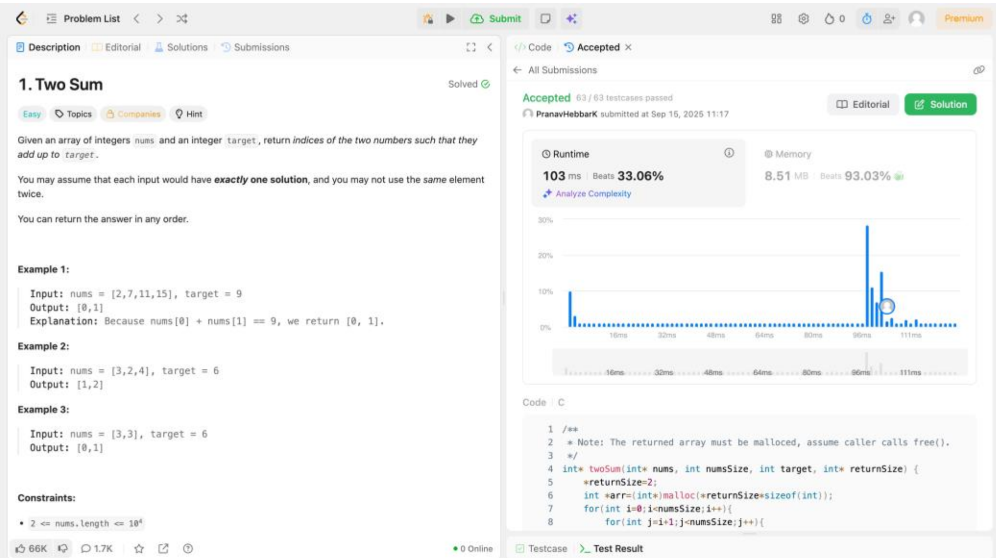
Leetcode Problems:

1) Two Sum

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* twoSum(int* nums, int numsSize, int target, int* returnSize) {
    *returnSize=2;
    int *arr=(int*)malloc(*returnSize*sizeof(int));
    for(int i=0;i<numsSize;i++){
        for(int j=i+1;j<numsSize;j++){
            if(nums[i]+nums[j]==target){
                arr[0]=i;
                arr[1]=j;
                return arr;
            }
        }
    }
    *returnSize=0;
    return malloc(sizeof(int)*0);


}
```

Output:

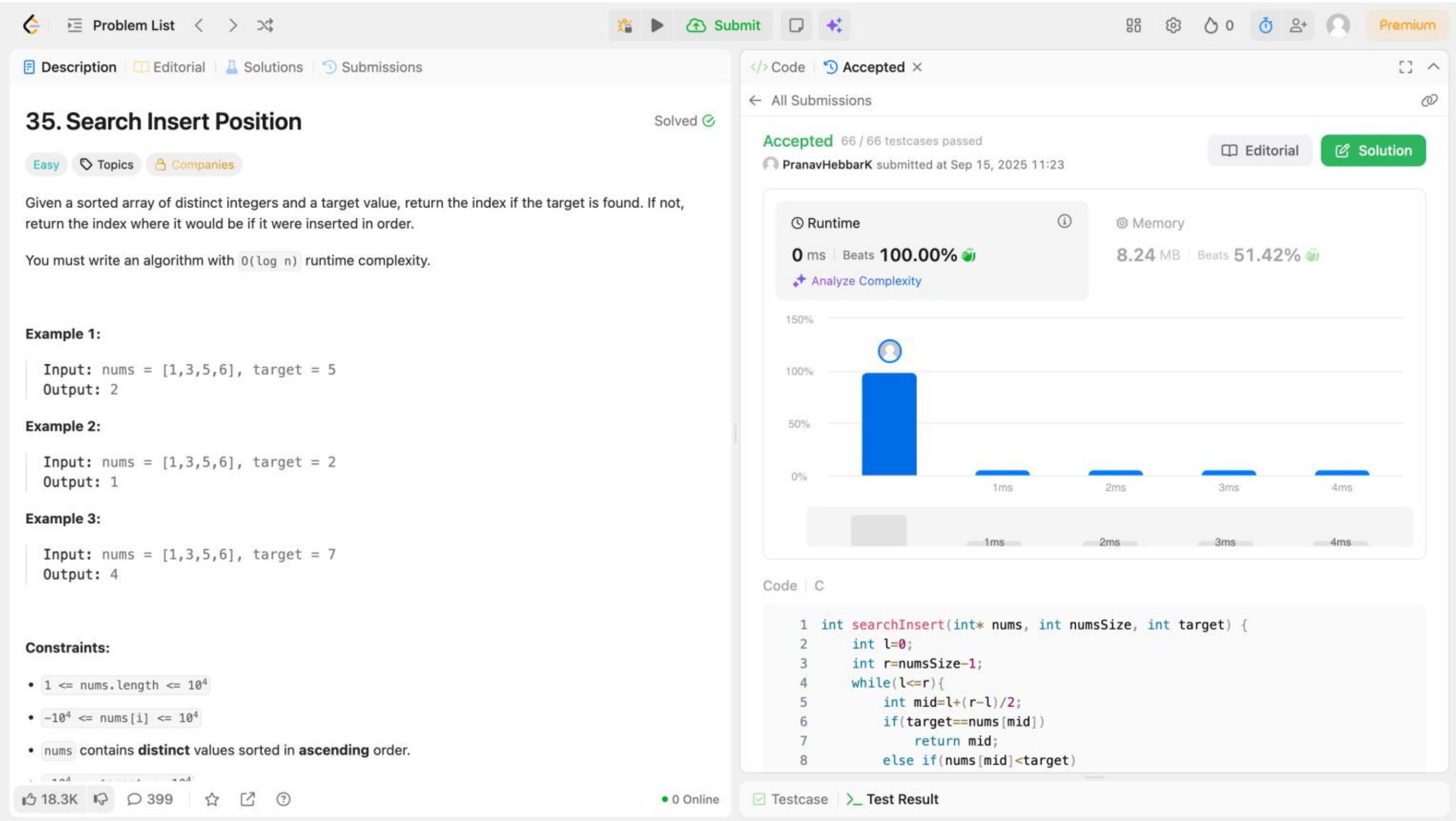


2) Search Insert Position

```c
int searchInsert(int* nums, int numsSize, int target) {
    int l=0;
    int r=numsSize-1;
    while(l<=r){
        int mid=l+(r-l)/2;
        if(target==nums[mid])
            return mid;
        else if(nums[mid]<target)
            l=mid+1;
        else
            r=mid-1;
    }
    return l;

}
```

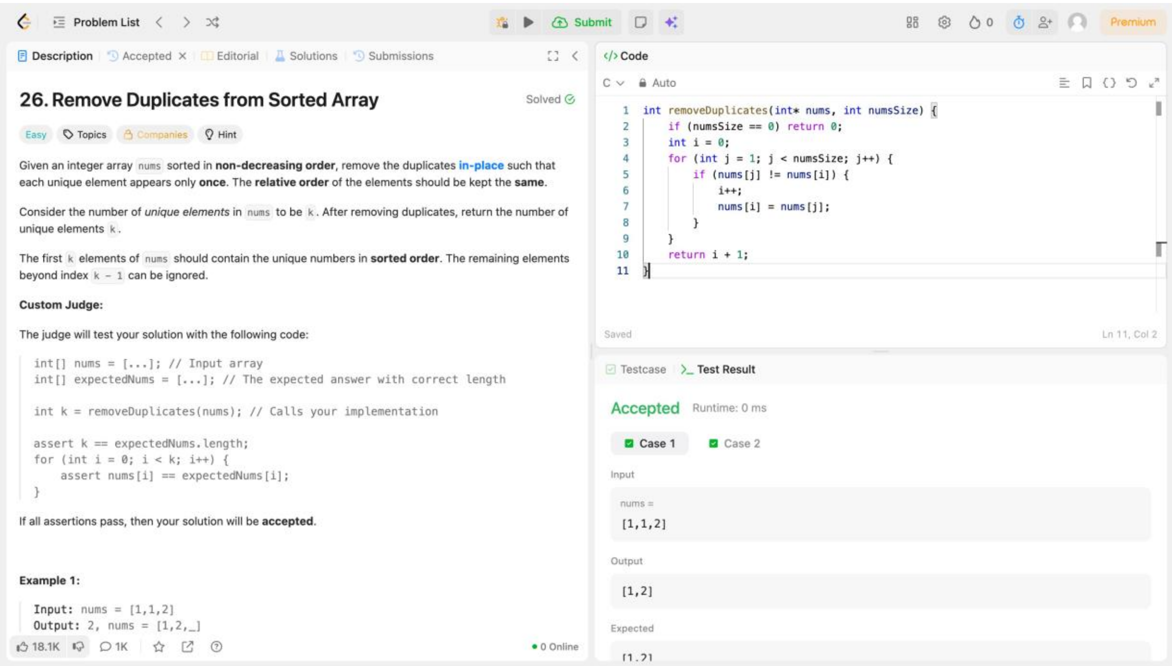
Output :



3) Remove Duplicates from Sorted Array

```c
int removeDuplicates(int* nums, int numsSize) {
    if (numsSize == 0) return 0;
    int i = 0;
    for (int j = 1; j < numsSize; j++) {
        if (nums[j] != nums[i]) {
            i++;
            nums[i] = nums[j];
        }
    }
    return i + 1;
}
```

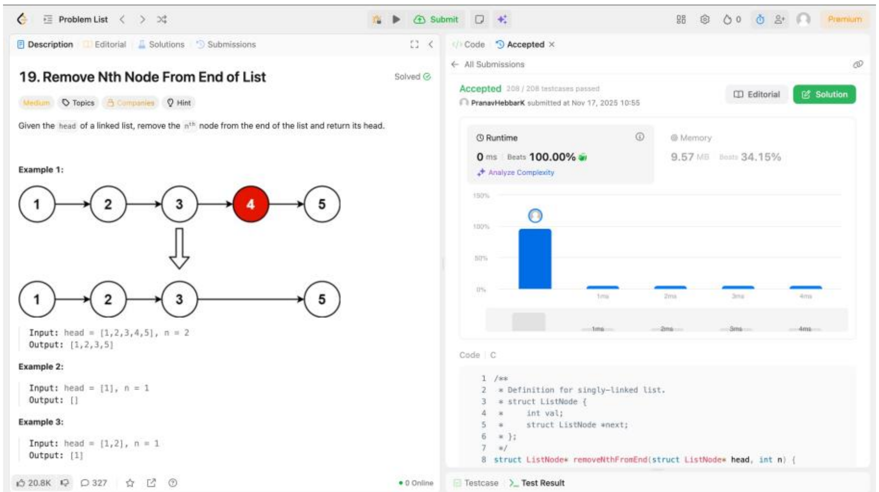
Output:



4) Remove Nth Node From End of List

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* removeNthFromEnd(struct ListNode* head,
int n) {
    if(!head || !head->next)
        return NULL;
    struct ListNode* fast=head;
    struct ListNode* slow=head;
    while(n--){
        fast=fast->next;
    }
    if(!fast){
        struct ListNode* temp=head;
        head=head->next;
        free(temp);
        return head;
    }
    while(fast->next){
        fast=fast->next;
        slow=slow->next;
    }
    struct ListNode* curr=slow->next;
    slow->next=slow->next->next;
    free(curr);
    return head;
}
```

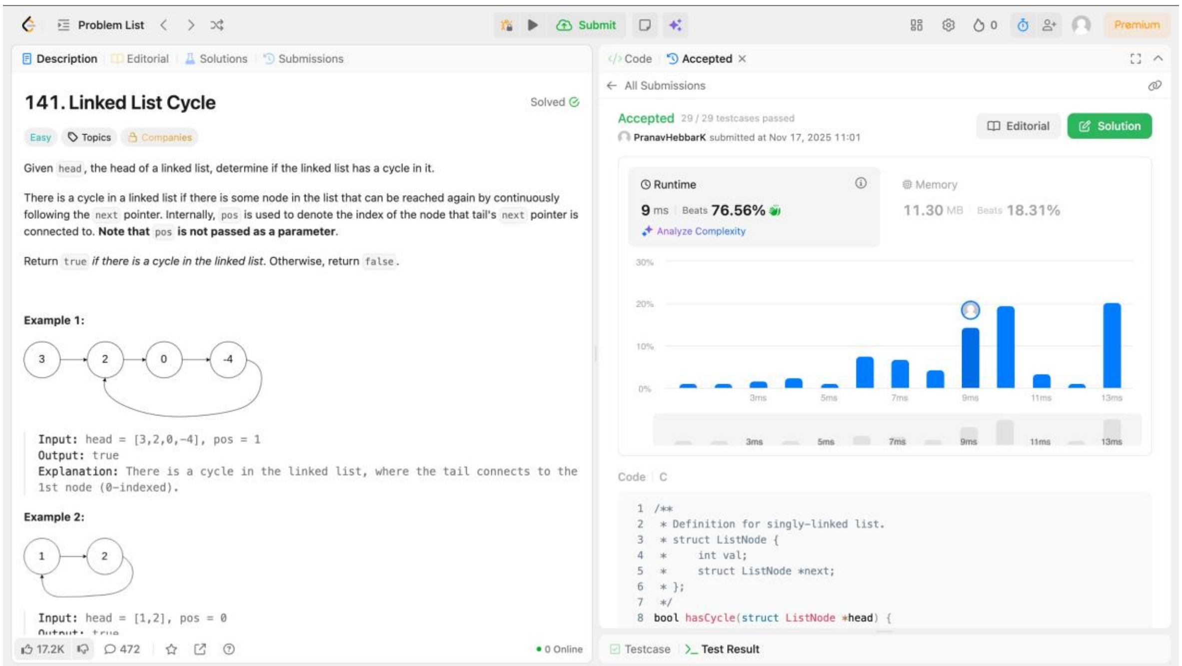
Output:



5) Linked List Cycle

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool hasCycle(struct ListNode *head) {
    struct ListNode* fast=head;
    struct ListNode* slow=head;
    while(fast && fast->next){
        fast=fast->next->next;
        slow=slow->next;
        if(fast==slow)
            return true;
    }
    return false;
}
```

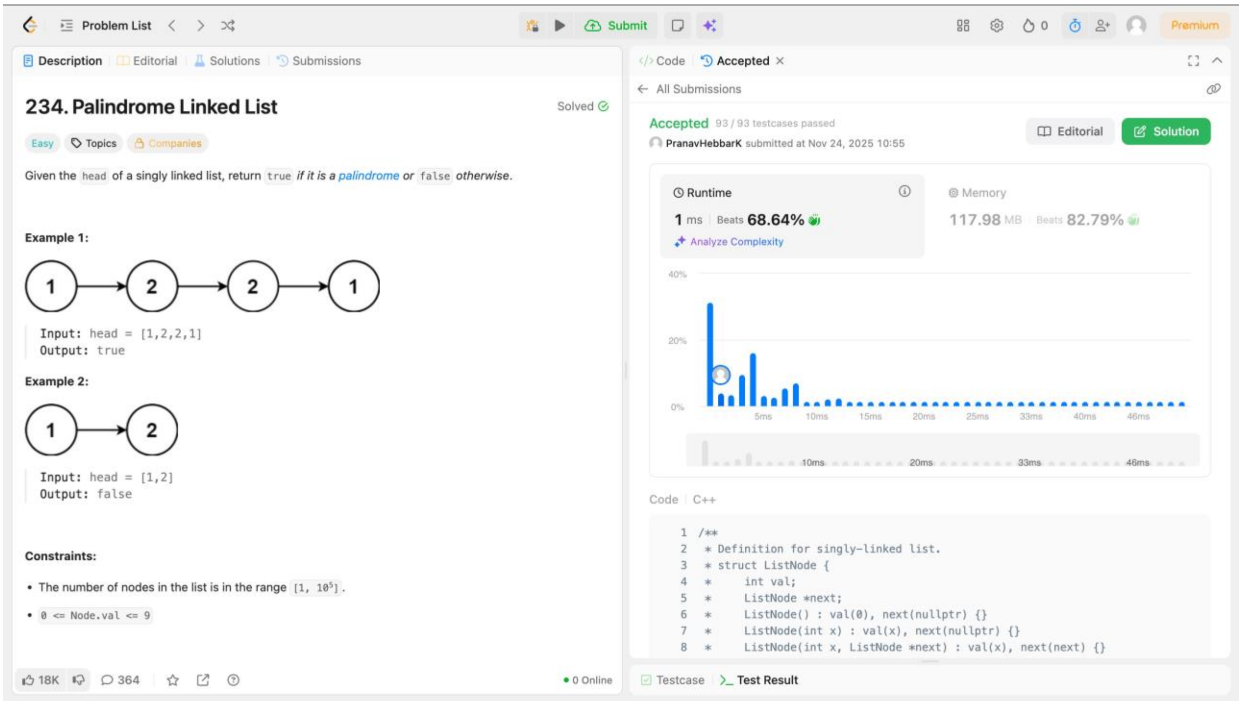
Output:



6) Palindrome Linked List

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x),
next(next) {}
 * };
 */
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        struct ListNode* slow=head;
        struct ListNode* fast=head;
        while(fast && fast->next){
            fast=fast->next->next;
            slow=slow->next;
        }
        if(fast)
            slow=slow->next;
        ListNode* prev=NULL;
        while(slow){
            struct ListNode* temp=slow->next;
            slow->next=prev;
            prev=slow;
            slow=temp;
        }
        struct ListNode* a=head;
        struct ListNode* b=prev;
        while(b){
            if(a->val==b->val){
                a=a->next;
                b=b->next;
            }
            else
                return false;
        }
        return true;
    }
};
```

Output:


Problem List
Submit
Premium
Description
Editorial
Solutions
Submissions
234. Palindrome Linked List
Solved
Easy
Topics
Companies
Given the head of a singly linked list, return true if it is a palindrome or false otherwise.
Example 1:
Input: head = [1,2,2,1]
Output: true
Example 2:
Input: head = [1,2]
Output: false
Constraints:
The number of nodes in the list is in the range [1, 10^5].
0 <= Node.val <= 9
18K
364
0 Online
Code
Accepted
All Submissions
Accepted 93 / 93 testcases passed
PranavHebbarK submitted at Nov 24, 2025 10:55
Editorial
Solution
Runtime
1 ms Beats 68.64%
Analyze Complexity
Memory
117.98 MB Beats 82.79%
Code C++
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
Testcase
Test Result

7) Find if Path Exists in Graph

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

void dfs(int node, int **adj, int *adjSize, bool
*visited) {
    visited[node] = true;
    for (int i = 0; i < adjSize[node]; i++) {
        int next = adj[node][i];
        if (!visited[next]) {
            dfs(next, adj, adjSize, visited);
        }
    }
}

bool validPath(int n, int** edges, int edgesSize, int*
edgesColSize, int source, int destination){
    int **adj = malloc(n * sizeof(int*));
    int *adjSize = calloc(n, sizeof(int));

    for (int i = 0; i < edgesSize; i++) {
        adjSize[edges[i][0]]++;
        adjSize[edges[i][1]]++;
    }

    for (int i = 0; i < n; i++) {
        adj[i] = malloc(adjSize[i] * sizeof(int));
        adjSize[i] = 0;
    }

    for (int i = 0; i < edgesSize; i++) {
        int u = edges[i][0], v = edges[i][1];
        adj[u][adjSize[u]++] = v;
        adj[v][adjSize[v]++] = u;
    }

    bool *visited = calloc(n, sizeof(bool));
    dfs(source, adj, adjSize, visited);

    bool result = visited[destination];

    for (int i = 0; i < n; i++) free(adj[i]);
    free(adj);
    free(adjSize);
```

```
    free(visited);

    return result;
}
```

Output: