

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

**SUJNYAN KINI (1BM22CS340)**

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” was carried out by **SUJNYAN KINI (1BM22CS340)**, 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 202324. 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. Sneha S Bagalkot  
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	LAB PROGRAM-1	4-6
2	LAB PROGRAM-2	7-9
3	LAB PROGRAM-3	9-14
4	LAB PROGRAM-4	15-17
5	LEETCODE PROBLEM-1	18-20
6	LAB PROGRAM-5	20-23
7	LEETCODE PROBLEM-2	23-25
8	LAB PROGRAM-6	25-32
9	LAB PROGRAM-7	33-35
10	LEETCODE PROBLEM-3	35-37
11	LAB PROGRAM-8	37-40
12	LEETCODE PROBLEM-4	41-42
13	LAB PROGRAM-9	42-48
14	HACKERRANK PROBLEM	48-51
15	LAB PROGRAM-10	51-53

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
SIZE 4 int top = -1; int
inp_array[SIZ]; void
push(); void pop();
void show();

void main()
{   int ch;   while
(1)
{
    printf("Operations on the stack:\n");
    printf("1.Push the element\n2.Pop the element\n3.Show\n4.End\n");    printf("Enter
the choice:\n ");
    scanf("%d",&ch);

    switch (ch)
    {        case 1:
push();        break;
case 2:        pop();
break;        case 3:
show();        break;
case 4:        exit(0);
default:
```

```

        printf("Invalid choice\n");
    }
}

void push()
{
    int x;
    if (top == SIZE - 1)
    {
        printf("Overflow\n");
    } else
    {
        printf("Enter the element to be added in the stack:\n ");
        scanf("%d", &x);    top = top + 1;
        inp_array[top] = x;
    }
} void pop() {
    if (top == -1)
    {
        printf("Underflow\n");
    } else
    {
        printf("Popped element: %d\n", inp_array[top]);
        top = top - 1;
    }
} void show() {
    if (top == -1)
    {
        printf("Underflow\n");
    } else
    {
        printf("Elements in the stack are: \n");    for
(int i = top; i >= 0; --i)
        printf("%d\n", inp_array[i]);
    }
}

```

Output:

```
C:\Users\Admin\Desktop\1BM22CS227\DS\stackimplementation.exe
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
1
Enter the element to be added in the stack:
4
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
1
Enter the element to be added in the stack:
5
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
2
Popped element: 5
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
3
Elements in the stack are:
4
Operations on the stack:
1.Push the element
2.Pop the element
3.Show
4.End
Enter the choice:
4

Process returned 0 (0x0)   execution time : 12.759 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 <stdlib.h>
#include <ctype.h>
```

```

#define SIZE 100

char stack[MAX];
int top = -1;

void push(char); char pop();
int precedence(char);
void infixToPos ix(char infix[], char pos ix[]);

void push(char item) {    if (top
== MAX - 1) {
    prin ("Overflow!\n");
} else {    top++;
    stack[top] = item;
}
}

char pop() {
    if (top == -1) {
        prin ("Underflow!\n");
    } else {
        char popped = stack[top];
        top--;
        return popped;
    }
}

int precedence(char symbol) {    if
(symbol == '^') {    return 3;
} else if (symbol == '*' || symbol == '/') {
    return 2;
} else if (symbol == '+' || symbol == '-') {
    return 1;    } else
{    return -1;
}
}

void infixToPos ix(char infix[], char pos ix[]) {    int i =
0, j = 0;
    char symbol, temp;

    push('#');

```

```

while ((symbol = infix[i++]) != '\0') {
    if (symbol == '(') {
        push(symbol);    } else if
(isalnum(symbol)) {      pos ix[j++]
= symbol;    } else if (symbol == ')')
{          while (stack[top] != '(') {
            pos ix[j++] = pop();
        }
        temp = pop(); // Remove '(' from the stack
    } else {
        while (precedence(stack[top]) >= precedence(symbol)) {
            pos ix[j++] = pop();
        }
        push(symbol);
    }
}

while (stack[top] != '#') {
    pos ix[j++] = pop();
}

pos ix[j] = '\0';
}

int main() {
    char infix[MAX], pos ix[MAX];

    prin ("Enter a valid parenthesized infix expression: ");    scanf("%s", infix);

    infixToPos ix(infix, pos ix);

    prin ("The pos ix exp is: %s\n", pos ix);    return 0;
}

```

Output:

```

Enter a valid parenthesized infix expression:
a*b+c*d-e
The postfix exp is: ab*cd*+e-

Process returned 0 (0x0)    execution time : 20.804 s
Press any key to continue.

```



### Lab program 3:

**3a) WAP to simulate the working of a queue of integers 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<string.h>
```

```
#include<math.h>
```

```
#define MAX 100
```

```
#define pf printf
```

```
#define sf scanf
```

```
int stack[MAX];
```

```
char postfix[MAX],infix[MAX];
```

```
int top=-1;
```

```
void push(char);
```

```
char pop();
```

```
int space(char);
```

```
void inToPost();
```

```
int precedence(char);
```

```
int isempty();
```

```
void print();
```

```
int post_eval();
```

```
int main()
```

```
{
```

```
int result;
```

```
printf("Enter the infix expression\n");
```

```
gets(infix);
```

```
inToPost();
```

```
print();
```

```
result=post_eval();
```

```
pf("\n\nThe value of the above post fix expression is: %d",result);
```

```
return 0;
```

```
}
```

```
void inToPost()
```

```
{
```

```

int i,j=0;
char symbol,next;
for(i=0;infix[i]!='\0';i++)
{
    if(!space(infix[i]))
    {
        symbol=infix[i];
        switch(symbol){

            case '(':
                push(symbol);
                break;

            case ')':
                while((next=pop())!='(')
                    postfix[j++]=next;
                break;

            case '+':
            case '-':
            case '/':
            case '*':
            case '^':
                while(!isempty() &&
precedence(stack[top])>=precedence(symbol))
                    postfix[j++]=pop();
                push(symbol);
                break;
            default:
                postfix[j++]=symbol;
                break;
        }
    }
}
while(!isempty())
{
    postfix[j++]=pop();
}
postfix[j]='\0';
}

void push(char c)
{
    stack[++top]=c;
}

```

```
}
```

```
char pop()
{
return stack[top--];
}
```

```
int isempty()
{
if(top==-1)
    return 1;
else
    return 0;
}
```

```
int precedence(char symbol)
{
switch(symbol){

    case '^': return 3;break;
    case '+':
    case '-':
        return 1;break;
    case '/':
    case '*':
        return 2;break;
    default:return 0;
}
}
```

```
void print()
{
int i=0;
printf("\n The postfix expression will be\n");
while(postfix[i])
{
    printf("%c",postfix[i++]);
}
printf("\n");
}
```

```
int space(char c){
if(c==' ' || c=='\t')
    return 1;
else
```

```

        return 0;
    }
    int post_eval()
    {
        int i,a,b;
        for(i=0;i<strlen(postfix);i++)
        {
            if(postfix[i]>='0'&& postfix[i]<='9')
                push(postfix[i]-'0');
            else{
                a=pop();
                b=pop();
                switch(postfix[i]){
                    case'+':
                        push(b+a);break;
                    case'-':
                        push(b-a);break;
                    case'/':
                        push(b/a);break;
                    case'*':
                        push(b*a);break;
                    case'^':
                        push(pow(b,a));break;
                }
            }
        }
        pop();
    }
}

```

Output:

```

Enter the infix expression
3*5-2+1/2

The postfix expression will be
35*2-12/+

The value of the above post fix expression is: 13

```

**3b ) WAP to simulate the working of a circular queue of integers 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>
#define size 5

int queue[size];
int front=-1;
int rear =-1;

void enqueue(int a)
{   if((front==rear+1) || (front==0 && rear==size-1))
    {
        printf("Queue overflow\n"); return;    }
    else {   if(front==-1)
front=0;
rear=(rear+1)%size;
queue[rear]=a;
    }
}

void dequeue()
{
    if(front==-1)
    {
        printf("Queue Empty\n");
    }
    else
    {
        int a=queue[front];
if(front==rear)
    {
front=-1;
rear=-1;
    }
    else
    {
front=(front+1)%size;
    }
}
```

```

        printf("Deleted element=%d\n",a);
return(a);
    }
}

void display()
{
    if(front==rear)
    {
        printf("Queue Empty\n");
return; } else {
    int
i;
        printf("\nFront=%d", front);
printf("\nItems=");
for(i=front;i!=rear;i=(i+1)%size)
    {
        printf("%d",queue[i]);
    }
    printf("%d",queue[i]);
printf("\nRear=%d",rear);
}
}

void main()
{
    int
choice;
int a;
while(1)
{
    printf("\n1.Insert\n2.Delete\n3.Display\nChoice:");
scanf("%d",&choice);

    switch (choice)
    {
        case 1:printf("Enter an element:");
scanf("%d",&a);        enqueue(a);
display();        break;

        case 2:dequeue();
display();
break;

        case 3:display();
break;
    }
}
}

```

}

Output:

```
1.Insert
2.Delete
3.Display
Choice:1
Enter an element:2
```

```
Front=0
Items=2
Rear=0
1.Insert
2.Delete
3.Display
Choice:1
Enter an element:3
```

```
Front=0
Items=23
Rear=1
1.Insert
2.Delete
3.Display
Choice:1
Enter an element:4
```

```
Front=0
Items=234
Rear=2
1.Insert
2.Delete
3.Display
Choice:2
Deleted element=2
```

```
Front=1
Items=34
Rear=2
```

#### **Lab program 4:**

**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.**

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

```
/*wap to create a linked list*/
#include<stdio.h>
#include<stdlib.h>
#define pf printf
#define sf scanf

struct node{
    int data;
    struct node *next;
};
struct node *head,*newnode,*temp,*last;

void push()
{
    newnode=(struct node *)malloc(sizeof(struct node));
    pf("Enter the data of the node to be inserted at the beginning:");
    sf("%d",&newnode->data);
    newnode->next=head;
    head=newnode;
    pf("\n---The element %d has been inserted at beg---",newnode->data);
}

void append()
{
    newnode=(struct node *)malloc(sizeof(struct node));
    pf("Enter the data of the node 1:");
    sf("%d",&newnode->data);
    newnode->next=NULL;
    temp=head;
    while(temp->next!=NULL)
    {
        temp=temp->next;
    }
    temp->next=newnode;
    pf("\nThe element %d has been inserted at the end:",newnode->data)
```



```

}

void insert_pos()
{
    int pos, i=0;
    newnode=(struct node *)malloc(sizeof(struct node));
    pf("Enter the data of the node :");
    sf("%d",&newnode->data);
    pf("\nEnter the pos:");
    sf("%d",&pos);
    temp=last=head;

    for(i=0;i<pos;i++)
    {
        last=temp;
        temp=temp->next;
    }
    last->next=newnode;
    newnode->next=temp;

}

void display()
{
    pf("The linked list you have created is: \n");
    temp=head;
    while(temp!=NULL)
    {

        pf("%d\t",temp->data);
        temp=temp->next;
    }

}

int main()
{

    int ch;
    while(1){
        pf("\n\nEnter your choice:\n1.Insert at beginning\n2.Insert at End\n3.Insert at a pos\n4.Display:");
        sf("%d",&ch);
        switch(ch){
            case 1:
                push();

```

```

        display();
        break;
    case 2: append();
            display();
            break;
    case 3:
            insert_pos();
            display();

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

}

```

Output:

```

Initial Linked List: 1 -> 2 -> 3 -> NULL
After insertion at the beginning: Linked List: 0 -> 1 -> 2 -> 3 -> NULL
After insertion at position 3: Linked List: 0 -> 1 -> 10 -> 2 -> 3 -> NULL
After insertion at the end: Linked List: 0 -> 1 -> 10 -> 2 -> 3 -> 20 -> NULL

Process returned 0 (0x0)   execution time : 0.023 s
Press any key to continue.

```

## Leetcode Problem-1:

### Min Stack Problem

```

#include<stdio.h>
#include<stdlib.h> typedef
struct
{   int
value;   int
min;

} StackNode;

typedef struct
{

```

```

    StackNode
*array;    int
capacity;  int top;
}MinStack;

MinStack* minStackCreate()
{
    MinStack* stack=(MinStack*)malloc(sizeof(MinStack));  stack-
>capacity = 10;    stack->array = (StackNode*)malloc(stack->capacity *
sizeof(StackNode));  stack->top = -1;  return stack;

}

void minStackPush(MinStack* obj, int val)
{
    if (obj->top == obj->capacity - 1)
    {
        obj->capacity
*= 2;
        obj->array = (StackNode*)realloc(obj->array, obj->capacity * sizeof(StackNode));
    }
    StackNode newNode;  newNode.value = val;  newNode.min = (obj->top == -1)
? val : (val < obj->array[obj->top].min) ? val : obj-
>array[obj->top].min;  obj->array[++(obj-
>top)] = newNode;
}

void minStackPop(MinStack* obj)
{
    if (obj->top != -
1)
    {
        obj-
>top--;
    }

}

int minStackTop(MinStack* obj)
{
    if (obj->top != -
1)
    {
        return obj->array[obj->top].value;
    }
    return -1;

}

int minStackGetMin(MinStack* obj)

```

```

{   if (obj->top != -
1)
{
    return obj->array[obj->top].min;
}
return -1;

}

void minStackFree(MinStack* obj)
{   free(obj-
>array);
free(obj);
}

```

Result:

#### Min Stack

##### Submission Detail

31 / 31 test cases passed.

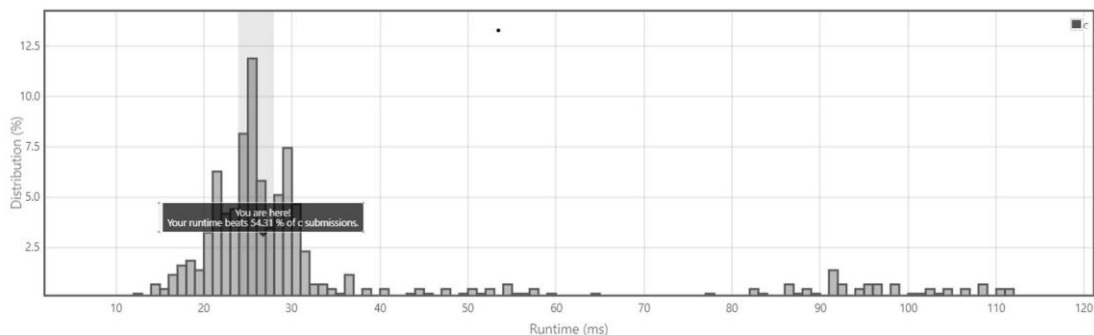
Runtime: 26 ms

Memory Usage: 13.8 MB

Status: **Accepted**

Submitted: 1 month, 2 weeks ago

##### Accepted Solutions Runtime Distribution



### Lab program 5:

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

- Create a linked list.
- Deletion of first element, specified element and last element in the list.
- Display the contents of the linked list.

```

#include<stdio.h>
#include<stdlib.h>
#define pf printf
#define sf scanf

```

```

struct node{
    int data;
    struct node *next;
};
struct node *head,*newnode,*temp,*temp1;

void create()
{
    newnode=(struct node *)malloc(sizeof(struct node));
    pf("Enter the data of the node to be inserted at the beginning:");
    sf("%d",&newnode->data);
    newnode->next=head;
    head=newnode;
    pf("\n---The element %d has been inserted at beg---",newnode->data);
    /*display*/
    pf("\nThe linked list you have created is: \n");
    temp=head;
    while(temp!=NULL)
    {

        pf("%d\t",temp->data);
        temp=temp->next;
    }
}

void delete_at_beg(){
    temp=head;
    if(head==NULL){
        pf("\nThe linked list is empty");
    }
    else if(head->next==NULL)
    {
        head=NULL;
        pf("\nThe only node is deleted, the list is now empty") ;
    }
    else{
        head=temp->next;
        free(temp);
    }
}

void delete_at_end()
{
    if(head==NULL)
    {

```

```

        pf("\nThe linked list is empty");
    }
    else if(head->next==NULL)
    {
        head=NULL;
        pf("\nThe only node is deleted, the list is now empty") ;
    }
    else
    {
        temp=head;
        while(temp->next!=NULL)
        {
            temp1=temp;
            temp=temp->next;

        }
        temp1->next=NULL;
        free(temp);
    }
}

void delete_atpos()
{
    temp=head;
    int pos,i;

    if(head==NULL){
        pf("\nThe linked list is empty");
    }
    else if(head->next==NULL)
    {
        head=NULL;
        pf("\nThe only node is deleted, the list is now empty");
    }
    else{
        pf("\nEnter the position of element to be deleted:");
        sf("%d",&pos);

        for(i=1;i<pos;i++)
        {
            temp1=temp;
            temp=temp->next;
            if(temp==NULL){
                pf("\nThere are less than %d elements in the list",pos);
                return 0;
            }
        }
    }
}

```

```

        }
    }
    temp1->next=temp->next;
    free(temp);
}

}

void display()
{
    pf("The linked list you have created is: \n");
    temp=head;
    while(temp!=NULL)
    {
        pf("%d\t",temp->data);
        temp=temp->next;
    }

}

int main()
{
    int ch;
    while(1)
    {
        pf("\n----Menu----\n 1.Create\n2.Delete_at_beg\n3.delete at end\n4.Delete at pos\n5.Display: ");
        sf("%d",&ch);
        switch(ch)
        {

            case 1:create();
                break;
            case 2:delete_at_beg();
                break;
            case 3:delete_at_end();
                break;
            case 4:delete_atpos();
                break;
            case 5: display();
                return 0;
        }
    }
}

```

```
}  
}
```

Output:

```
---The element 80 has been inserted at beg---  
The linked list you have created is:  
80      60      40      20  
----Menu----  
 1.Create  
2.Delete_at_beg  
3.delete at end  
4.Delete at pos  
5.Display: 2  
  
----Menu----  
 1.Create  
2.Delete_at_beg  
3.delete at end  
4.Delete at pos  
5.Display: 3  
  
----Menu----  
 1.Create  
2.Delete_at_beg  
3.delete at end  
4.Delete at pos  
5.Display: 4  
  
Enter the position of element to be deleted:2  
  
----Menu----  
 1.Create  
2.Delete_at_beg  
3.delete at end  
4.Delete at pos  
5.Display: 5  
The linked list you have created is:  
60  
-----  
Process exited after 48.74 seconds with return value 0  
Press any key to continue . . .
```



## Leetcode Problem -2:

### Reversal Linked List-II

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
 *   struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {

    if (head == NULL) return NULL;

    if (left == right) return head;

    struct ListNode* prev = NULL;
    struct ListNode* curr = head;

    int index = 1;
    while (index < left)
    {
        prev = curr; curr
        = curr->next;
        index++;
    }

    struct ListNode* leftMinusOneNode = prev;

    struct ListNode* leftNode = curr;
    struct ListNode* next = NULL;

    while (left <= right)
    {
        next = curr->next;

        curr->next = prev;

        prev = curr;
        curr = next;
        left++;
    }

    if (leftMinusOneNode == NULL) // means head
    changes head = prev; else
```

```
leftMinusOneNode->next = prev;
```

```
leftNode->next = curr;
```

```
return head;
```

```
}
```

Result:

#### Reverse Linked List II

##### Submission Detail

44 / 44 test cases passed.

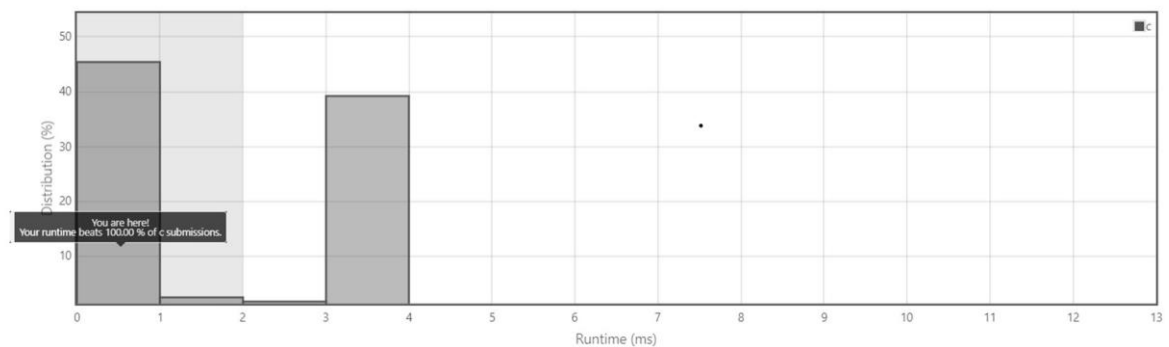
Runtime: 0 ms

Memory Usage: 5.9 MB

Status: **Accepted**

Submitted: 1 month, 1 week ago

##### Accepted Solutions Runtime Distribution



### Lab program 6:

**6a) 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;
```

```
};
```

```
typedef struct Node Node;
```

```
Node *createNode(int value)
```

```
{
```

```
Node *newNode=(Node*)malloc(sizeof(Node));
newNode->data=value;  newNode->next=NULL;  return newNode;
```

```
}
```

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

```
Node *sortList(Node *head)
{
    if(head==NULL || head->next==NULL)
    {        return
head;
    }
```

```
    int swapped;
Node *temp;
    Node *end=NULL;

    do
    {

        swapped=0;
temp=head;    while(temp->next !=end)
        {
            if(temp->data > temp->next->data)
            {
                int tempData=temp->data;
temp->data=temp->next->data;
temp->next->data=tempData;
swapped=1;

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

```

        }while(swapped);

    return head;
}

Node *reverseList(Node *head)
{
    Node *prev=NULL;
    Node *current=head;
    Node *nextNode=NULL;

    while(current!=NULL)
    {
        nextNode=current->next;
        current->next=prev;
        prev=current;
        current=nextNode;
    } return
    prev;

}

Node *concatenatedLists(Node *list1,Node *list2)
{

    if(list1==NULL)
    { return
    list2;
    }

    Node *temp=list1; while(temp-
    >next!=NULL)
    {
        temp=temp->next;
    }
    temp->next=list2;
    return list1;

}

int main()
{
    Node *list1=createNode(3); list1-
    >next=createNode(1); list1->next-
    >next=createNode(4);

```

```

    Node *list2=createNode(2);    list2->
next=createNode(5);

    printf("Original list 1:");
    displayList(list1);

    printf("Original list 2:");
    displayList(list2);

    list1=sortList(list1);
    printf("Sorted list 1:");
    displayList(list1);

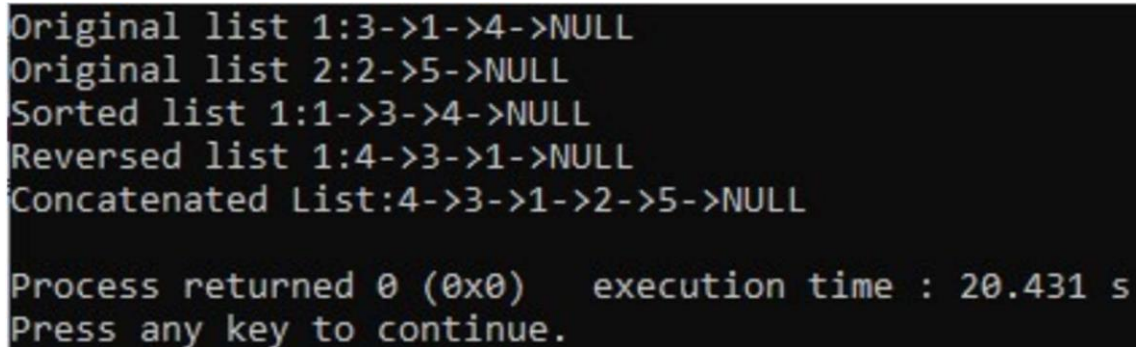
    list1=reverseList(list1);
    printf("Reversed list 1:");
    displayList(list1);

    Node *Concatenated= concatenatedLists(list1,list2);
    printf("Concatenated List:");
    displayList(Concatenated);    return 0;

}

```

Output:



```

Original list 1:3->1->4->NULL
Original list 2:2->5->NULL
Sorted list 1:1->3->4->NULL
Reversed list 1:4->3->1->NULL
Concatenated List:4->3->1->2->5->NULL

Process returned 0 (0x0)    execution time : 20.431 s
Press any key to continue.

```

## 6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
//Stack
```

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

```
struct Node
```

```

{   int data;
struct Node *next;
};

typedef struct Node Node;

Node *createNode(int value)
{
    Node *newNode=(Node*)malloc(sizeof(Node));
    newNode->data=value;   newNode->next=NULL;   return newNode;
}

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

typedef struct
{
    Node *top;
}LinkedList;

void push(LinkedList *stack,int value)
{
    Node *newNode=createNode(value);
    newNode->next=stack->top;   stack->top=newNode;
}

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

```

```

        return poppedValue;
    }

int main()
{

    LinkedList stack;
    stack.top=NULL;

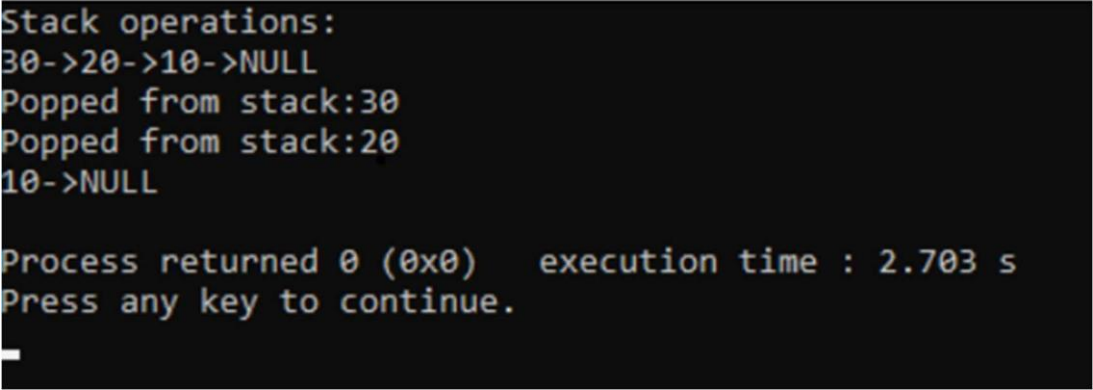
    printf("Stack operations:\n");
    push(&stack,10);   push(&stack,20);
    push(&stack,30);
    displayList(stack.top);

    printf("Popped from stack:%d\n",pop(&stack));
    printf("Popped from stack:%d\n",pop(&stack));

    displayList(stack.top);
    return 0;
}

```

Output:



```

Stack operations:
30->20->10->NULL
Popped from stack:30
Popped from stack:20
10->NULL

Process returned 0 (0x0)   execution time : 2.703 s
Press any key to continue.
_

```

//Queue

```

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

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

typedef struct Node Node;

```

```

Node *createNode(int value)
{
    Node *newNode=(Node*)malloc(sizeof(Node));
    newNode->data=value;  newNode->next=NULL;  return newNode;
}

```

```

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

```

```

typedef struct
{
    Node *front;
    Node *rear;
}LinkedList;

```

```

void enqueue(LinkedList *queue,int value)
{
    Node *newNode=createNode(value);
    if(queue->front==NULL)
    {
        queue->front=newNode;    queue->rear=newNode;
    }
    else
    {
        queue->rear->next=newNode;    queue->rear=newNode;
    }
}

```

```

int dequeue(LinkedList *queue)
{
    if(queue->front==NULL)
    {

```



```

        printf("Queue is empty\n");
return -1;
    }
    int dequeuedvalue=queue->front->data;
Node *temp=queue->front;    queue-
>front=queue->front->next;    free(temp);

    return dequeuedvalue;
}

int main()
{
    LinkedList queue;
queue.front=NULL;
queue.rear=NULL;

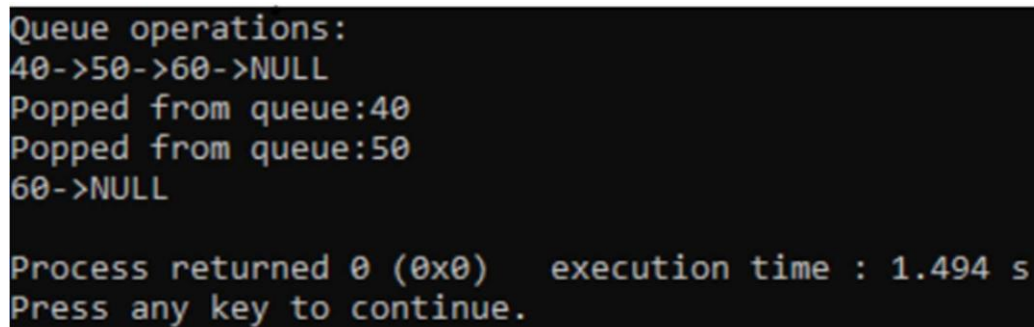
    printf("Queue operations:\n");
enqueue(&queue,40);
enqueue(&queue,50);
enqueue(&queue,60);
displayList(queue.front);

    printf("Popped from queue:%d\n",dequeue(&queue));
printf("Popped from queue:%d\n",dequeue(&queue));

    displayList(queue.front);
return 0;
}

```

Output:



```

Queue operations:
40->50->60->NULL
Popped from queue:40
Popped from queue:50
60->NULL

Process returned 0 (0x0)    execution time : 1.494 s
Press any key to continue.
_

```

## 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*
prev;    struct
Node* next;
};
```

```
struct Node* createNode(int data)
{    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL)
{
    printf("Memory allocation failed\n");
    exit(1);
}
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
```

```
void insertNodeToLeft(struct Node* head, struct Node* target, int data)
{
    struct Node* newNode = createNode(data);
    if (target->prev != NULL)
    {
        target->prev->next = newNode;
        newNode->prev = target->prev;
    }
    else
    {
        head = newNode;
    }
    newNode->next = target;    target-
    >prev = newNode;
}
```

```

void deleteNode(struct Node* head, int value)
{
    struct Node* current =
head;   while (current !=
NULL)
    {
        if (current->data ==
value)
        {
            if (current->prev != NULL)
            {
                current->prev->next = current->next;
            }
else
            {
                head = current->next;
            }
            if (current->next != NULL)
            {
                current->next->prev = current->prev;
            }
free(current);
return;
        }
        current = current->next;
    }
    printf("Node with value %d not found\n", value);
}

```

```

void displayList(struct Node* head)
{
    printf("Doubly Linked List:
");   while (head != NULL)
    {
        printf("%d <-> ", head->data);
head = head->next;
    }
    printf("NULL\n");
}

```

```

int main()
{
    struct Node* head = NULL;

```

```

    head = createNode(1);    head->next =
createNode(2);    head->next->prev =
head;    head->next->next =
createNode(3);    head->next->next-
>prev = head->next;

    displayList(head);

    insertNodeToLeft(head, head->next, 10);
printf("After insertion:\n");
displayList(head);

    deleteNode(head, 2);
printf("After deletion:\n");
displayList(head);

    return 0;
}

```

Output:

```

Doubly Linked List: 1 <--> 2 <--> 3 <--> NULL
After insertion:
Doubly Linked List: 1 <--> 10 <--> 2 <--> 3 <--> NULL
After deletion:
Doubly Linked List: 1 <--> 10 <--> 3 <--> NULL

Process returned 0 (0x0)    execution time : 0.076 s
Press any key to continue.

```

### Leetcode Problem -3:

#### Split Linked List in Parts

```

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

```

\* Note: The returned array must be  
    malloced, assume caller calls free().

```
*/ int getLength(struct ListNode*  
head) {    int length = 0;    while (head  
!= NULL) {        length++;        head  
= head->next;  
    }  
    return length;  
}
```

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize)  
{    int length = getLength(head);    int partSize = length / k;    int remainder =  
length % k;
```

```
    struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct ListNode*));  
    *returnSize = k;
```

```
    for (int i = 0; i < k; i++) {  
        int currentPartSize = partSize + (i < remainder ? 1 : 0);
```

```
        if (currentPartSize == 0) {  
result[i] = NULL;  
        } else {  
result[i] = head;  
            for (int j = 0; j < currentPartSize - 1; j++) {  
head = head->next;  
            }  
        }
```

```
        struct ListNode* temp = head-  
>next;        head->next = NULL;  
head = temp;  
    }  
}
```

```
    return result;  
}
```

Result:

### Split Linked List in Parts

#### Submission Detail

43 / 43 test cases passed.

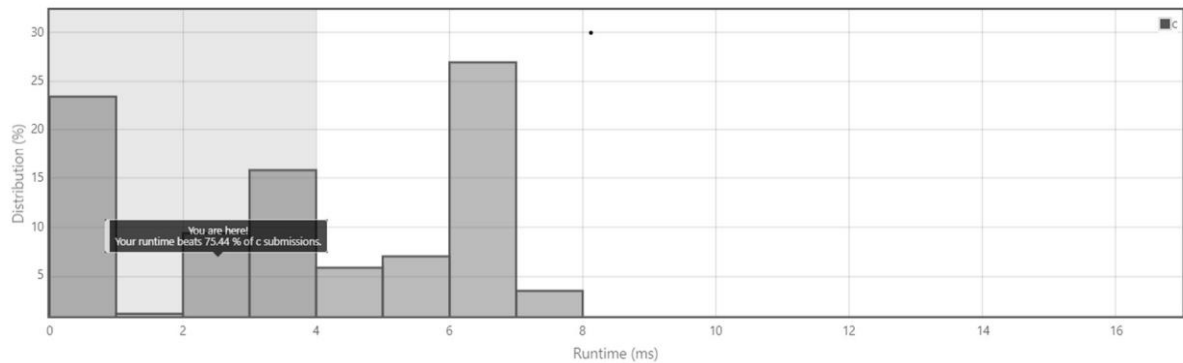
Runtime: 2 ms

Memory Usage: 6.2 MB

Status: **Accepted**

Submitted: 1 month ago

Accepted Solutions Runtime Distribution



#### Lab program 8:

##### Write a program

- To construct a binary Search tree.
- To traverse the tree using all the methods i.e., in-order, preorder and post order
- 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 = node->right = NULL;
return node;
}
```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

do {
printf("Enter your choice:\n1. Insert\n2. Print Inorder\n3. Print Preorder\n4. Print
Postorder\n5. Exit\n"); scanf("%d", &choice);

```

```

switch (choice) {
case 1:
printf("Enter the value to be inserted:
"); scanf("%d", &data); root =
insert(root, data); break; case 2:
printf("Inorder traversal of binary tree is \n");

```

```

inorder(root);
printf("\n");
break; case 3:
printf("Preorder traversal of binary tree is
\n"); preorder(root); printf("\n");
break; case 4:
printf("Postorder traversal of binary tree is
\n"); postorder(root); printf("\n");
break; case 5: printf("Exiting...\n");
break; default:

```

```

        printf("Invalid choice. Please try again.\n");
    }
} while (choice != 5);

return 0;
}

```

Output:

```

Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
1
Enter the value to be inserted: 20
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
1
Enter the value to be inserted: 10
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
1
Enter the value to be inserted: 30
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
1
Enter the value to be inserted: 5
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
1
Enter the value to be inserted: 15
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
1
Enter the value to be inserted: 45
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
2
Inorder traversal of binary tree is
5 10 15 20 30 45
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
3
Preorder traversal of binary tree is
20 10 5 15 30 45
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
4
Postorder traversal of binary tree is
5 15 10 45 30 20
Enter your choice:
1. Insert
2. Print Inorder
3. Print Preorder
4. Print Postorder
5. Exit
5
Exiting...

Process returned 0 (0x0)   execution time : 42.620 s
Press any key to continue.

```



## Leetcode Problem -4:

### Rotate List

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k) {

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

        return head;

    int len = 1;

    struct ListNode *tail = head;

    while (tail->next != NULL) {

        tail = tail->next;

        len++;

    }

    k = k % len;

    if (k == 0)

        return head;

    struct ListNode *p = head;

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

        p = p->next;

    }

    tail->next = head;

    head = p->next;
```

```

p->next = NULL;

return head;

}

```

Result:

[Rotate List](#)

#### Submission Detail

232 / 232 test cases passed.

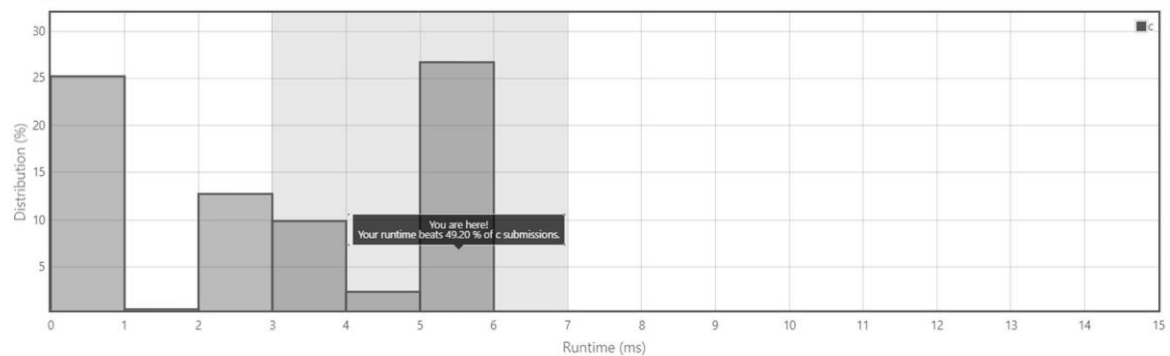
Runtime: 5 ms

Memory Usage: 6.1 MB

Status: **Accepted**

Submitted: 2 weeks, 3 days ago

Accepted Solutions Runtime Distribution



### Lab program 9:

**9a) Write a program to traverse a graph using BFS method.**

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

```
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
```

```
void bfs(int v)
```

```
{
```

```
for(i=1; i<=n; i++)
```

```
if(a[v][i] && !visited[i])
```

```
q[++r]=i;
```

```
if(f<=r)
```

```

{
visited[q[f]]=1;

bfs(q[f++]);

}

}

int main()

{

int v;

printf("\n Enter the number of vertices:");

scanf("%d", &n);

for(i=1; i<=n; i++)

{

q[i]=0;

visited[i]=0;

}

printf("Enter graph data in matrix form:\n");

for(i=1; i<=n; i++)

for(j=1; j<=n; j++)

scanf("%d", &a[i][j]);

printf("\n Enter the starting vertex:");

scanf("%d", &v);

bfs(v);

```

```

printf("The nodes which are reachable are:\n");

for(i=1; i<=n; i++)

if(visited[i])

printf("%d\t", i);

return 0;

}

```

Output:

```

Enter the number of vertices:6
Enter graph data in matrix form:
0 1 1 1 1 0
1 0 0 1 0 0
1 0 0 1 1 0
1 1 1 0 0 0
1 0 1 0 0 0
0 0 0 0 0 0

Enter the starting vertex:4
The nodes which are reachable are:
1      2      3      4      5
-----

```

**9b) Write a program to check whether given graph is connected or not using DFS method.**

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

```
#include<conio.h>
```

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

```
void dfs(int v)
```

```
{
```

```

int i;

s[v]=1;

for(i=1; i<=n; i++)

if(a[v][i] && !s[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

int main()

{

int i, j, count=0;

printf("\n Enter number of vertices:");

scanf("%d", &n);

for(i=1; i<=n; i++)

{

s[i]=0;

for(j=1; j<=n; j++)

a[i][j]=0;

}

printf("Enter the adjacency matrix:\n");

for(i=1; i<=n; i++)

for(j=1; j<=n; j++)

```

```

scanf("%d", &a[i][j]);

dfs(1);

printf("\n");

for(i=1; i<=n; i++)

{

if(s[i])

count++;

}

if(count==n)

printf("Graph is connected");

else

printf("Graph is not connected");

return 0;

}

```

Output:

```

Enter number of vertices:6
Enter the adjacency matrix:
0 1 1 1 1 0
1 0 0 1 0 0
1 0 0 1 1 1
1 1 1 0 0 1
1 0 1 0 0 0
0 0 1 1 0 0

1->2
2->4
4->3
3->5
3->6
Graph is connected
-----

```

## HackerRank Problem

### Swap Nodes

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

typedef struct Node {
    int data;    struct
Node* left;    struct
Node* right;
} Node;

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

void inOrderTraversal(Node* root, int* result, int* index)
{    if (root == NULL) return;    inOrderTraversal(root-
>left, result, index);    result[(*index)++] = root->data;
    inOrderTraversal(root->right, result, index);
}

void swapAtLevel(Node* root, int k, int level) {
    if (root == NULL) return;    if (level % k == 0)
    {        Node* temp = root->left;        root->left
= root->right;        root->right = temp;
    }
    swapAtLevel(root->left, k, level + 1);
    swapAtLevel(root->right, k, level + 1);
}

int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count,
int* queries, int* result_rows, int* result_columns) {
    // Build the tree
    Node** nodes = (Node**)malloc((indexes_rows + 1) * sizeof(Node*));
    for (int i = 1; i <= indexes_rows; i++) {        nodes[i] = createNode(i);
    }

    for (int i = 0; i < indexes_rows; i++) {        int leftIndex =
indexes[i][0];        int rightIndex = indexes[i][1];        if
```

```

(leftIndex != -1) nodes[i + 1]->left = nodes[leftIndex];    if
(rightIndex != -1) nodes[i + 1]->right = nodes[rightIndex];
    }
    // Perform swaps and store results    int** result =
(int**)malloc(queries_count * sizeof(int*));
    *result_rows = queries_count;
    *result_columns = indexes_rows;    for (int i = 0; i <
queries_count; i++) {        swapAtLevel(nodes[1], queries[i], 1);
int* traversalResult = (int*)malloc(indexes_rows * sizeof(int));
int index = 0;        inOrderTraversal(nodes[1], traversalResult,
&index);        result[i] = traversalResult;
    }

    free(nodes);
return result;
}

int main() {
int n;
    scanf("%d", &n);

    int** indexes = malloc(n * sizeof(int*));
for (int i = 0; i < n; i++) {        indexes[i]
= malloc(2 * sizeof(int));
        scanf("%d %d", &indexes[i][0], &indexes[i][1]);
    }

    int queries_count;
    scanf("%d", &queries_count);

    int* queries = malloc(queries_count * sizeof(int));
for (int i = 0; i < queries_count; i++) {
scanf("%d", &queries[i]);
    }

    int result_rows;    int result_columns;    int** result = swapNodes(n, 2,
indexes, queries_count, queries, &result_rows, &result_columns);

    for (int i = 0; i < result_rows; i++) {
for (int j = 0; j < result_columns; j++) {
printf("%d ", result[i][j]);
        }
printf("\n");
        free(result[i]); // Free memory allocated for each row
    }
    free(result); // Free memory allocated for the result array

```



```

// Free memory allocated for indexes and queries
arrays   for (int i = 0; i < n; i++) {       free(indexes[i]);
    }
free(indexes);
free(queries);

return 0;
}

```

Result:

Prepare > Data Structures > Trees > Swap Nodes [Algo]

## Swap Nodes [Algo] ★

Problem

Submissions

Leaderboard

Discussions

Editorial

You made this submission 3 days ago.

Score: 40.00   Status: **Accepted**

### 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  $H(K) = K \bmod m$  (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 TABLE_SIZE 10
```

```

int hashFunction(int key) {
return key % TABLE_SIZE;
}

void insertValue(int hashTable[], int key)
{   int i = 0;   int hkey =
hashFunction(key);   int index;

    do {
        index = (hkey + i) % TABLE_SIZE;
    if (hashTable[index] == -1) {
hashTable[index] = key;
        printf("Inserted key %d at index %d\n", key,
index);        return;    }        i++;
    } while (i < TABLE_SIZE);

    printf("Unable to insert key %d. Hash table is full.\n", key);
}

int searchValue(int hashTable[], int key)
{   int i = 0;   int hkey =
hashFunction(key);   int index;

    do {        index = (hkey + i) %
TABLE_SIZE;        if (hashTable[index]
== key) {
        printf("Key %d found at index %d\n", key, index);
return index;
        }
    i++;
    } while (i < TABLE_SIZE);

    printf("Key %d not found in hash table.\n", key);
return -1;
}

int main() {   int
hashTable[TABLE_SIZE];
for (int i = 0; i < TABLE_SIZE;
i++) {        hashTable[i] = -1;
    }

    insertValue(hashTable, 1234);
insertValue(hashTable, 5678);
insertValue(hashTable, 9876);

```

```
    searchValue(hashTable, 5678);  
searchValue(hashTable, 1111);  
  
    return 0;  
}
```

Output:

```
Inserted key 1234 at index 4  
Inserted key 5678 at index 8  
Inserted key 9876 at index 6  
Key 5678 found at index 8  
Key 1111 not found in hash table.  
  
Process returned 0 (0x0)    execution time : 0.074 s  
Press any key to continue.  
|
```