TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

AMRIT SCIENCE CAMPUS

Data Structures and Algorithms

Lab Report

SUBMITTED BY:
Name: Sasank Lama
Roll: 13/079
Date: 2024/05/18

SUBMITTED TO:
Satya Bahadur Maharjan
Department of CSIT

Internal Teacher's Signature External's Signature

TABLE OF CONTENTS

Stack:

1. Write a menu driven program to illustrate basic operations of stack using array.

a) Push

b) Pop

c) Traverse

d) Exit

```
//ARRAY IMPLEMENTATION OF STACK
#include<stdio.h>
#include<conio.h> // Only required for some compilers/environments
#define MAX 10
struct stack {
  int items[MAX];  // Array to store items
  int top;       // Top of stack
};

typedef struct stack st;
void create_empty_stack(st *s); // Function prototype
void push(st *s, int element);
void pop(st *s);
void display(st *s);

int main() {
  printf("Sasank Lama\n");
  int element, choice;
  st s; // Declaring a stack variable
  create_empty_stack(&s); // Pass the address of the stack variable

  do {
    printf("\n\nEnter your choice\n");
    printf("\n1: Push element");
    printf("\n2: Display elements");
    printf("\n3: Pop element");
    printf("\n4: Exit\n");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
     case 1:
        printf("\nEnter the number: ");
        scanf("%d", &element);
        push(&s, element);
        break;
     case 2:
```

```
            display(&s);
            break;

          case 3:
            pop(&s);
            break;

          case 4:
            printf("\nExiting...\n");
            return 0;

          default:
            printf("\nInvalid Choice!\n");
     }
   } while (1);
   return 0;
}
void create_empty_stack(st *s) {
   s->top = -1; // Initialize top to -1 to indicate an empty stack
}
int isempty(st *s) {
   return s->top == -1; // Return 1 if stack is empty, else return 0
}
int isfull(st *s) {
   return s->top == MAX - 1; // Return 1 if stack is full, else return 0
}

void push(st *s, int element) {
   if (isfull(s))
      printf("\nThe stack is overflow: Stack Full!\n");
   else
      s->items[++(s->top)] = element;
}

void display(st *s) {
  if (isempty(s))
      printf("\nThe stack is empty!\n");
   else {
      printf("\nElements in the stack:\n");
      for (int i = s->top; i >= 0; i--)
         printf("%d\n", s->items[i]);
   }
}void pop(st *s) {
   if (isempty(s))
      printf("\nStack Underflow: Empty Stack!\n");
else
      printf("\nThe deleted item is %d\n", s->items[(s->top)--]);
}
```

OUTPUT

C:\Users\sasank\OneDrive\Desktop\DSA\dsa1.exe

```
The deleted item is 20

Enter your choice

1: Push element
2: Display elements
3: Pop element
4: Exit

Enter your choice: 4

Exiting...

Process returned 0 (0x0)   execution time : 33.730 s
Press any key to continue.
```

2. Write a menu driven program to illustrate basic operations of stack using pointer.

a) Push

b) Pop

c) Traverse

d) Exit

```
// IMPLEMENTATION OF STACK USING POINTER

#include <stdio.h>
#include <conio.h> // Only required for some compilers/environments
#define MAX 100

struct stack {
  int item[MAX];
  int tos;
};

typedef struct stack st;
void push(st *s, int d);
int pop(st *s);
void display(st *s);

int main() {
  printf("Sasank Lama\n");
  int dta, ch, x;
  st s; // Declare a stack variable directly
  s.tos = -1; // Initialize tos
  printf("\n*menu for program*:\n");
  printf("1: Push\n2: Pop\n3: Display\n4: Exit\n");

  do {
    printf("\nEnter your choice: ");
    scanf("%d", &ch);

    switch (ch) {
      case 1:
        printf("Enter data to be inserted: ");
        scanf("%d", &dta);
        push(&s, dta);
        break;
      case 2:
        x = pop(&s);
        printf("\nPopped item is: %d\n", x);
        break;
      case 3:
```

```
                display(&s);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (ch != 4);
    return 0;
}

void push(st *s, int d) {
    if (s->tos == MAX - 1) {
        printf("Stack is full\n");
    } else {
        ++s->tos;
        s->item[s->tos] = d;
    }
}

int pop(st *s) {
    int itm;
    if (s->tos == -1) {
        printf("Stack is empty\n");
        return 0;
    } else {
        itm = s->item[s->tos];
        s->tos--;
        return itm;
    }
}

void display(st *s) {
   int i;
    if (s->tos == -1)
        printf("There are no data items to display\n");
    else {
        printf("Stack elements:\n");
        for (i = s->tos; i >= 0; i--) {
            printf("%d\t", s->item[i]);
        }
        printf("\n");
    }
}
```

OUTPUTS

C:\Users\sasank\OneDrive\Desktop

```
 Sasank Lama
Enter your choice

1: Push element
2: Display elements
3: Pop element
4: Exit

Enter your choice: 1

Enter the number: 20

Enter your choice

1: Push element
2: Display elements
3: Pop element
4: Exit

Enter your choice: 2

Elements in the stack:
20

Enter your choice

1: Push element
2: Display elements
3: Pop element
4: Exit

Enter your choice: 3

The deleted item is 20

Enter your choice

1: Push element
2: Display elements
3: Pop element
4: Exit
```

C:\Users\sasank\OneDrive\Desktop\DSA\dsa2.exe

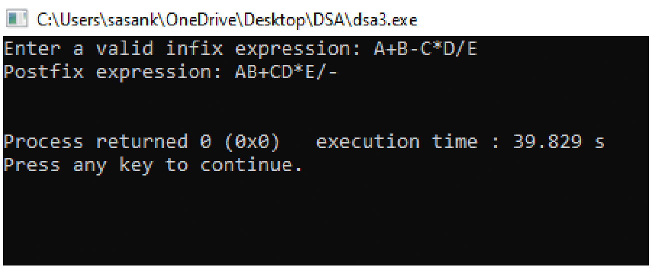
```
Sasank Lama

*menu for program*:
1: Push
2: Pop
3: Display
4: Exit

Enter your choice: 1
Enter data to be inserted: 14

Enter your choice: 3
Stack elements:
14

Enter your choice: 2

Popped item is: 14

Enter your choice: 4
Exiting...

Process returned 0 (0x0)   execution time : 123.022 s
Press any key to continue.
```

3. Write a program to convert Infix Expression into Postfix Expression.

```
//Program to convert Infix Expression into Postfix Expression.

#include <stdio.h>
#include <string.h>
#include <ctype.h>

int precedence(char);

int main() {
  printf("Sasank Lama\n");
  int i, otos = -1, ptos = -1, l, l1;
  char infix[100], poststack[100], opstack[100];

  printf("Enter a valid infix expression: ");
  fgets(infix, sizeof(infix), stdin);
  l = strlen(infix);

  for (i = 0; i < l; i++) {
    if (infix[i] == '(') {
      opstack[++otos] = infix[i];
      l1++;
    } else if (isalpha(infix[i])) {
      poststack[++ptos] = infix[i];
    } else if (infix[i] == ')') {
      while (opstack[otos] != '(') {
        poststack[++ptos] = opstack[otos--];
      }
      otos--; // Discard '(' from opstack
    } else { // Operators

      while (otos != -1 && precedence(opstack[otos]) >= precedence(infix[i])) {
        poststack[++ptos] = opstack[otos--];
      }
      opstack[++otos] = infix[i];
    }
  }

  while (otos != -1) {
    poststack[++ptos] = opstack[otos--];
  }

  // Display postfix expression
  printf("Postfix expression: ");
  for (i = 0; i <= ptos; i++) {
    printf("%c", poststack[i]);
  }
```

```
  printf("\n");

  return 0;
}

int precedence(char ch) {

  switch (ch) {
    case '+':
    case '-':
      return 1;

    case '*':
    case '/':
      return 2;

    default:
      return 0;
  }
}
```

OUTPUT

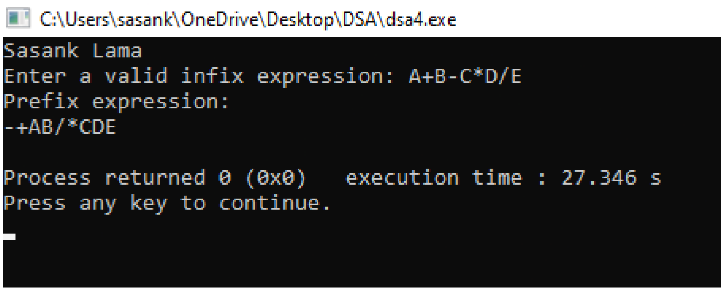


4. Write a program to convert Infix Expression into Prefix Expression.

```
// Program to convert Infix Expression into Prefix Expression.

#include <stdio.h>
#include <string.h>
#include <ctype.h>

int precedence(char);

int main() {
  printf("Sasank Lama\n");
  int i, otos = -1, ptos = -1, l;
  char infix[100], poststack[100], opstack[100], prefix[100];

  printf("Enter a valid infix expression: ");
  fgets(infix, sizeof(infix), stdin);
  l = strlen(infix);

  // Reverse the infix expression
  for (i = 0; i < l / 2; i++) {
    char temp = infix[i];
    infix[i] = infix[l - 1 - i];
    infix[l - 1 - i] = temp;
  }

  // Convert '(' to ')' and vice versa
  for (i = 0; i < l; i++) {
    if (infix[i] == '(')
      infix[i] = ')';
    else if (infix[i] == ')')
      infix[i] = '(';
  }

  for (i = 0; i < l; i++) {
    if (infix[i] == ')') {
      opstack[++otos] = infix[i];
    } else if (isalpha(infix[i])) {
      poststack[++ptos] = infix[i];
    } else if (infix[i] == '(') {
      while (opstack[otos] != ')') {
        poststack[++ptos] = opstack[otos--];
      }
      otos--; // Discard ')' from opstack
    } else { // Operators
      while (otos != -1 && precedence(opstack[otos]) > precedence(infix[i])) {
        poststack[++ptos] = opstack[otos--];
      }
```

```
            opstack[++otos] = infix[i];
        }
    }

    while (otos != -1) {
        poststack[++ptos] = opstack[otos--];
    }

    // Reverse the postfix expression to get prefix expression
    for (i = 0; i <= ptos; i++) {
        prefix[i] = poststack[ptos - i];
    }
    prefix[i] = '\0';

    // Display prefix expression
    printf("Prefix expression: %s\n", prefix);

    return 0;
}

int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;

        case '*':
        case '/':
            return 2;

        default:
            return 0;
    }
}
```

OUTPUT



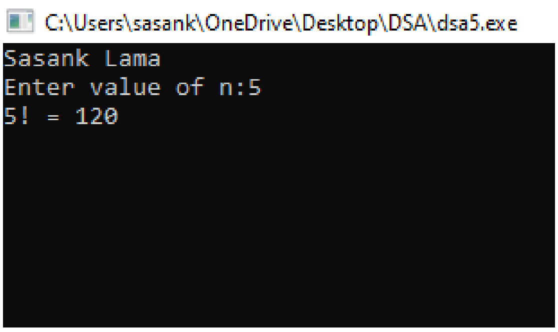
Recursion:

5. Write a recursive program to find the factorial value of given number.

//Recursive program to find the factorial value of given number

```
#include<stdio.h>
#include<conio.h>
void main()
{
        printf("Sasank Lama\n");
        int n;
        long int facto;
        long int factorial(int n);
        printf("Enter value of n:");
        scanf("%d",&n);
        facto=factorial(n);
        printf("%d! = %ld",n,facto);
        getch();
}
long int factorial(int n)
{
        if(n == 0)
                return 1;
        else
        return n * factorial(n-1);
}
```

OUTPUT



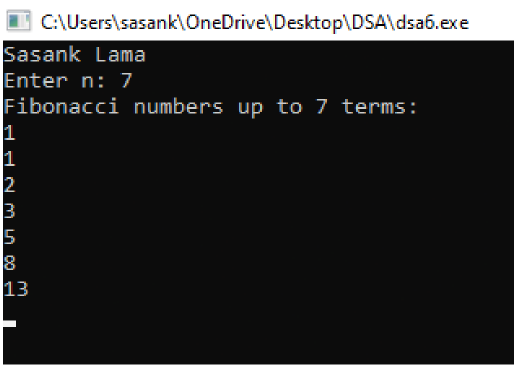
6. Write a recursive program to find a Fibonacci sequence.

// Program to generate Fibonacci series up to n terms using recursive function

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

int fibo(int);

void main() {
  printf("Sasank Lama \n");
  int n, i;
  printf("Enter n: ");
  scanf("%d", &n);
  printf("Fibonacci numbers up to %d terms:\n", n);
  for (i = 1; i <= n; i++)
    printf("%d\n", fibo(i));
  getch();
}

int fibo(int k) {
  if (k == 1 || k == 2)
    return 1;
  else
    return fibo(k - 1) + fibo(k - 2);
}
```

—

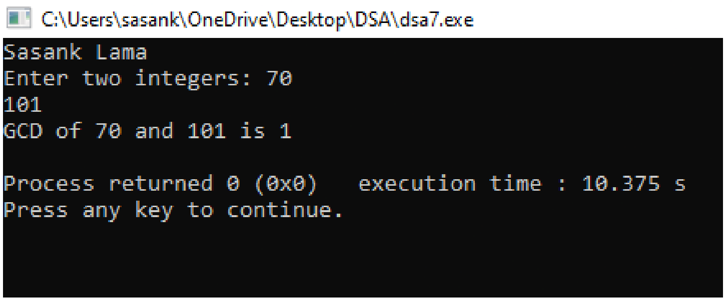
OUTPUT



7. Write a recursive program to find GCD of two integers.

```
//Program to find GCD of two integers.

#include <stdio.h>
int gcd(int a, int b);

int main() {
  printf("Sasank Lama\n");
  int num1, num2;

  printf("Enter two integers: ");
  scanf("%d %d", &num1, &num2);
    printf("GCD of %d and %d is %d\n", num1, num2, gcd(num1, num2));

  return 0;
}

int gcd(int a, int b) {
  if (b == 0) {
    return a; // GCD is the non-zero value of 'a'
  } else {
    return gcd(b, a % b); // Recursive call with 'b' and 'a % b'
  }
}
```

OUTPUT



```
C:\Users\sasank\OneDrive\Desktop\DSA\dsa7.exe
Sasank Lama
Enter two integers: 70
101
GCD of 70 and 101 is 1

Process returned 0 (0x0)   execution time : 10.375 s
Press any key to continue.
```

8. Write a recursive program to implement TOH problem. (Show the output for 3 disks)

```
//Recursive program to implement TOH problem

#include <stdio.h>
#include <conio.h>

void TOH(int, char, char, char); // Function prototype

void main() {
  printf("Sasank Lama\n");
  int n;
  printf("Enter number of disks: ");
  scanf("%d", &n);
  TOH(n, 'O', 'D', 'I');
  getch();
}

void TOH(int n, char A, char B, char C) {
  if (n > 0) {
    TOH(n - 1, A, C, B);
    printf("Move disk %d from %c to %c\n", n, A, B);
    TOH(n - 1, C, B, A);
  }
}
```

OUTPUT

C:\Users\sasank\OneDrive\Desktop\DSA\dsa8.exe

```
Sasank Lama
Enter number of disks: 3
Move disk 1 from O to D
Move disk 2 from O to I
Move disk 1 from D to I
Move disk 3 from O to D
Move disk 1 from I to O
Move disk 2 from I to D
Move disk 1 from O to D
```

Queue:

9. Write a menu driven program to illustrate basic operations of Linear queue using array implementation and pointer implementation.
a) Enqueue
b) Dequeue
c) Display all values
d) Exit

```
// Linear queue using array implementation
#include <stdio.h>
#include <conio.h>
#include<stdlib.h>
#define SIZE 20

struct queue {
   int item[SIZE];
   int rear;
   int front;
};
typedef struct queue qu;

void insert(qu*);
void delet(qu*);
void display(qu*);

void main() {
   printf("Sasank Lama\n");
   int ch;
   qu *q;
   q->rear = -1;
   q->front = 0;

   printf("Menu for program:\n");
   printf("1: Insert\n2: Delete\n3: Display\n4: Exit\n");

   do {
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            insert(q);
            break;
         case 2:
```

```
            delet(q);
            break;
        case 3:
            display(q);
            break;
        case 4:
            exit(1);
            break;
        default:
            printf("Your choice is wrong\n");
        }
    } while (ch < 5);
    getch();
}

/* Insert function */
void insert(qu *q) {
    int d;
    printf("Enter data to be inserted: ");
    scanf("%d", &d);
    if (q->rear == SIZE - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->item[q->rear] = d;
    }
}
/* Delete function */
void delet(qu *q) {
    int d;
    if (q->rear < q->front) {
        printf("Queue is empty\n");
    } else {
        d = q->item[q->front];
        q->front++;
        printf("Deleted item is: %d\n", d);
    }
}
/* Display function */
void display(qu *q) {
    int i;
    if (q->rear < q->front) {
        printf("Queue is empty\n");
    } else {
        for (i = q->front; i <= q->rear; i++) {
            printf("%d\t", q->item[i]);
        }
    }
  }
```

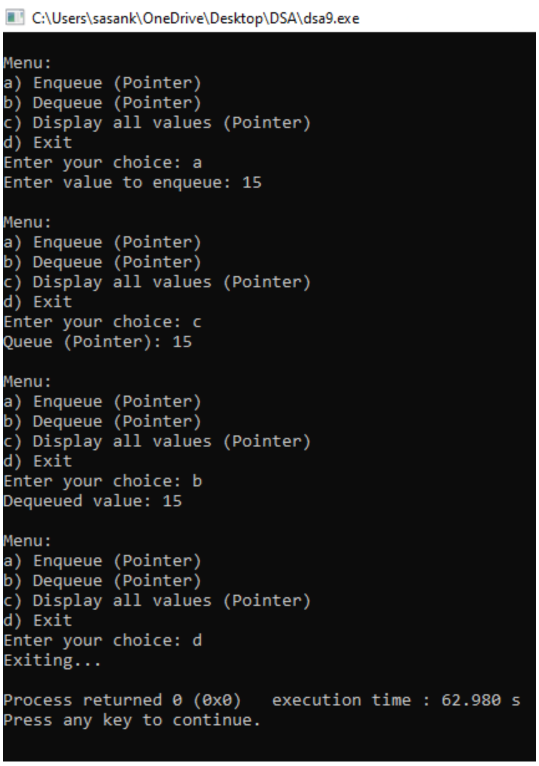
}*//Linear queue using pointer implementation*

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_SIZE 10

// Queue structure

typedef struct Node {

    int data;

    struct Node *next;

} Node;

typedef struct {

    Node *front, *rear;

} QueuePointer;

// Function prototypes

void initializeQueuePointer(QueuePointer *q);

void enqueuePointer(QueuePointer *q, int value);

int dequeuePointer(QueuePointer *q);

void displayPointer(QueuePointer q);

int main() {

    QueuePointer qPointer;

    int choice, value;

    // Initialize queue for pointer implementation
    initializeQueuePointer(&qPointer);

    do {
        // Display menu

        printf("\nMenu:\n");

        printf("a) Enqueue (Pointer)\n");

        printf("b) Dequeue (Pointer)\n");

        printf("c) Display all values (Pointer)\n");

        printf("d) Exit\n");

        printf("Enter your choice: ");

        scanf(" %c", &choice);

        switch (choice) {
            case 'a':

                printf("Enter value to enqueue: ");

                scanf("%d", &value);

                enqueuePointer(&qPointer, value);

                break;
```

```
 case 'b':
                printf("Dequeued value: %d\n", dequeuePointer(&qPointer));
                break;


            case 'c':
                printf("Queue (Pointer): ");
                displayPointer(qPointer);
                break;


            case 'd':
                printf("Exiting...\n");
                break;


            default:
                printf("Invalid choice\n");
        }
    } while (choice != 'd');

    return 0;
}

// Function to initialize a queue (pointer implementation)

void initializeQueuePointer(QueuePointer *q) {

    q->front = NULL;

    q->rear = NULL;
}

// Function to enqueue an element into the queue (pointer implementation)

void enqueuePointer(QueuePointer *q, int value) {

    Node *newNode = (Node *)malloc(sizeof(Node));

    if (newNode == NULL) {

        printf("Memory allocation failed\n");
        return;
    }

    newNode->data = value;

    newNode->next = NULL;

    if (q->rear == NULL) {

        q->front = newNode;

        q->rear = newNode;

    } else {

        q->rear->next = newNode;

        q->rear = newNode;
    }
}

// Function to dequeue an element from the queue (pointer implementation)

int dequeuePointer(QueuePointer *q) {

    if (q->front == NULL) {

        printf("Queue is empty\n");

        return -1;
    }
```

```
    int value = q->front->data;

    Node *temp = q->front;

    q->front = q->front->next;

    free(temp);

    if (q->front == NULL)

        q->rear = NULL;

    return value;
}

// Function to display all elements of the queue (pointer implementation)

void displayPointer(QueuePointer q) {

    if (q.front == NULL) {

        printf("Queue is empty\n");

        return;
    }

    Node *current = q.front;

    while (current != NULL) {

        printf("%d ", current->data)
;
        current = current->next;

    }

    printf("\n");

}
```

OUTPUT



10. Write a menu driven program to illustrate basic operations of circular queue having following menu:
a) Enqueue
b) Dequeue
c) Traverse
d) Exit

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20

struct cqueue {
   int item[SIZE];
   int rear;
   int front;
};

typedef struct cqueue qu;

void insert(qu*);
void delet(qu*);
void display(qu*);

int main() {
   printf("Sasank Lama\n");
   int ch;
   qu *q;
   q->rear = SIZE - 1;
   q->front = SIZE - 1;

   clrscr();
   printf("Menu for Circular Queue Operations:\n");
   printf("a) Enqueue\n");
   printf("b) Dequeue\n");
   printf("c) Traverse\n");
   printf("d) Exit\n");

   do {
      printf("Enter your choice: ");
      scanf(" %c", &ch);
      switch(ch) {
         case 'a':
            insert(q);
            break;
         case 'b':
            delet(q);
            break;
```

```
        case 'c':
          display(q);
          break;
        case 'd':
          exit(0);
        default:
          printf("Invalid choice! Please try again.\n");
      }
    } while (1);

    getch();
    return 0;
}

void insert(qu *q) {
    int d;
    if ((q->rear + 1) % SIZE == q->front) {
        printf("Queue is full\n");
    } else {
        q->rear = (q->rear + 1) % SIZE;
        printf("Enter data to be inserted: ");
        scanf("%d", &d);
        q->item[q->rear] = d;
    }
}

void delet(qu *q) {
    if (q->rear == q->front) {
        printf("Queue is empty\n");
    } else {
        q->front = (q->front + 1) % SIZE;
        printf("Deleted item is: %d\n", q->item[q->front]);
    }
}

void display(qu *q) {
    int i;
    if (q->rear == q->front) {
        printf("Queue is empty\n");
    } else {
        printf("Items of queue are:\n");
        for (i = (q->front + 1) % SIZE; i != q->rear; i = (i + 1) % SIZE) {
            printf("%d\t", q->item[i]);
        }
        printf("%d\t", q->item[q->rear]);
        printf("\n");
    }
}
```

—

C:\Users\sasank\OneDrive\Desktop\DSA\dsa9.exe

```
Menu:
a) Enqueue (Pointer)
b) Dequeue (Pointer)
c) Display all values (Pointer)
d) Exit
Enter your choice: a
Enter value to enqueue: 15

Menu:
a) Enqueue (Pointer)
b) Dequeue (Pointer)
c) Display all values (Pointer)
d) Exit
Enter your choice: c
Queue (Pointer): 15

Menu:
a) Enqueue (Pointer)
b) Dequeue (Pointer)
c) Display all values (Pointer)
d) Exit
Enter your choice: b
Dequeued value: 15

Menu:
a) Enqueue (Pointer)
b) Dequeue (Pointer)
c) Display all values (Pointer)
d) Exit
Enter your choice: d
Exiting...

Process returned 0 (0x0)   execution time : 62.980 s
Press any key to continue.
```

LINKED LIST:

11. Write a program that uses functions to perform the following operations on singly linked
list
a) Creation
b) Insertion
1) Insertion at beginning
2) Insertion at specified position
3) Insertion at end
c) Deletion
1) Deletion from the beginning
2) Deletion from the specified position
3) Deletion from the end
d) Traversal.
e) Exit

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

struct node{
   int data;
   struct node *next;
};
typedef struct node NODE;
NODE *start = NULL;
NODE *last = NULL;

void insert_at_begg(int);
void insert_at_spe(int, int);
void insert_at_end(int);
void del_first();
void del_last();
void del_spe();
void show(int);

int main(){
   printf("Sasank Lama");
   int choice1, choice_ins, choice_del, choice_show;
   int item,pos;
   do{
      printf("\nChoose (1/2/3/4)\n1. Insert\n2. Delete\n3. Display\n4. Exit \n= ");
            scanf("%d", &choice1);
      if(choice1==1){
         printf("\nChoose: \n1. Insert at beginning\n2. Insert at specific position\n3. Insert at end\n=");
         scanf("%d", &choice_ins);
         switch (choice_ins){
```

```
        case 1:
          printf("Enter item to be inserted = ");
          scanf("%d", &item);
          insert_at_begg(item);
          break;

        case 2:
          printf("Enter item to be inserted and position where to insert = ");
          scanf("%d%d", &item, &pos);
          insert_at_spe(item, pos);

          break;

        case 3:
          printf("Enter item to be inserted = ");
          scanf("%d", &item);
          insert_at_end(item);
          break;

        default:
          printf("Invalid choice..");
        }
      }
      else if(choice1==2){
        printf("\nChoose: \n1. Delete at beginning\n2. Delete at specific position\n3. Delete at
end\n=");
        scanf("%d", &choice_del);
        switch (choice_del){
        case 1:
          del_first();
          break;

        case 2:
          del_spe();
          break;

        case 3:
          del_last();
          break;

        default:
          printf("Invalid choice..");
        }
      }
      if (choice1==3){
        show(item);
      }
      else if (choice1 == 4)
```

```
    printf("Exiting....");
  }while(choice1<4);
  return 0;
}

void insert_at_begg(int item){
  NODE *ptr;
  ptr = (NODE*)malloc(sizeof(NODE));
  ptr->data = item;
  if(start==NULL)
  ptr->next = NULL;
  else
  ptr->next = start;
  start = ptr;
}

void insert_at_end(int item){
  NODE *ptr,*temp;
  ptr = (NODE*)malloc(sizeof(NODE));
  ptr->data = item;
  ptr->next = NULL;
  if(start==NULL)
  start = ptr;
  else{
    temp = start;
    while(temp->next != NULL)
    temp = temp->next;
  }
  temp->next = ptr;
}

void insert_at_spe(int item, int pos){
  NODE *temp, *q;
  int i;
  temp = start;
  for(i=1;i<pos-1;i++){
    temp = temp->next;
    if(temp = NULL)
    printf("There are less than %d elements.",pos);
  }
  q = (NODE*)malloc(sizeof(NODE));
  q->next = temp->next;
  q->data = item;
  temp->next = q;
}

void del_first(){
  NODE *temp;
```

```
    if(start==NULL){
        printf("List is empty.\n");
        return;
    }
    else{
        temp = start;
        printf("Deleted item is %d.",start->data);
        start = start->next;
        free(temp);
    }
}

void del_spe(){
    NODE *temp, *p;
    int pos,i;
    if(start==NULL){
        printf("List is empty.\n");
        return;
    }
    else{
        printf("Enter the position of element to be deleted = \n");
        scanf("%d",&pos);
        temp = start;
        for(i = 1;i<pos-1;i++)
        temp = temp->next;
        p = temp->next;
        printf("Deleted item is %d.",p->data);
        temp->next = p->next;
        free(p);
        free(temp);
    }
}

void del_last(){
    NODE *temp;
    if(start == NULL){
        printf("List is empty.\n");
        return;
    }
    else if(start->next = NULL){
        temp = start;
        start = NULL;
        printf("Deleted item is %d.",temp->data);
        free(temp);
    }
    else{
        temp = start;
        while (temp->next->next != NULL)
```

```
      temp = temp->next;
      printf("Deleted item is %d.",temp->next->data);
      free(temp->next);
      temp->next = NULL;
   }
}

void show(int item){
   NODE *ptr,*temp;
   if(start==NULL){
      printf("List is empty.\n");
      exit(0);
   }
   temp = start;
   while(temp!=NULL){
      printf("%d\t",temp->data);
      temp = temp->next;
   }
}
```

OUTPUT

```
C:\Users\sasank\OneDrive\Desktop\DSA\dsa11.exe
Sasank Lama
Choose (1/2/3/4)
1. Insert
2. Delete
3. Display
4. Exit
= 1

Choose:
1. Insert at beginning
2. Insert at specific position
3. Insert at end
=1
Enter item to be inserted = 50

Choose (1/2/3/4)
1. Insert
2. Delete
3. Display
4. Exit
= 1

Choose:
1. Insert at beginning
2. Insert at specific position
3. Insert at end
=3
Enter item to be inserted = 20

Choose (1/2/3/4)
1. Insert
2. Delete
3. Display
4. Exit
= 3
50      20
Choose (1/2/3/4)
1. Insert
2. Delete
3. Display
4. Exit
=
```

12. Write a program that uses functions to perform the following operations on circular linked
List
a) Creation
b) Insertion
1) Insertion at beginning
2) Insertion at specified position
3) Insertion at end
c) Deletion
1) Deletion from the beginning
2) Deletion from the specified position
3) Deletion from the end
d) Traversal.
e) Exit

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
 int data;
 struct Node *next;
}
 Node;
 Node* createList();
 Node* insertAtBeginning(Node*, int);
 Node* insertAtPosition(Node*, int, int);
 Node* insertAtEnd(Node*, int);
 Node* deleteFromBeginning(Node*);
 Node* deleteFromPosition(Node*, int);
 Node* deleteFromEnd(Node*);
 void traverse(Node*);
 void freeList(Node*);
 int main()
{
printf("Sasank Lama\n");
 Node *head = NULL;
 int choice, data, position;
do
{
 printf("\nCircular Linked List Operations:\n");
 printf("1. Creation\n");
```

```
printf("2. Insertion\n");
printf("3. Deletion\n");
printf("4. Traversal\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice)
{
case 1:
head = createList();
break;
case 2:
printf("Enter data to insert: ");
scanf("%d", &data);
printf("1. Insert at beginning\n");
printf("2. Insert at specified position\n");
printf("3. Insert at end\n");
printf("Enter insertion choice: ");
scanf("%d", &position);
switch(position)
{
case 1:
head = insertAtBeginning(head, data);
break;
case 2:
printf("Enter position to insert: ");
scanf("%d", &position);
head = insertAtPosition(head, data, position);
break;
case 3:
head = insertAtEnd(head, data);
break;
default:
printf("Invalid choice!\n");
}
break;
case 3:
printf("1. Delete from beginning\n");
printf("2. Delete from specified position\n");
printf("3. Delete from end\n");
printf("Enter deletion choice: ");
scanf("%d", &position);
switch(position)
{
case 1:
head = deleteFromBeginning(head);
break;
case 2:
```

```
 printf("Enter position to delete: ");
 scanf("%d", &position);
 head = deleteFromPosition(head, position);
 break;
 case 3:
 head = deleteFromEnd(head);
 break;
 default:
 printf("Invalid choice!\n");
 }
 break;
 case 4:
 printf("Circular Linked List: ");
 traverse(head);
 printf("\n");
 break;
 case 5:
 freeList(head);
 printf("Exiting...\n");
 break;
 default:
 printf("Invalid choice!\n");
 }
 }
while(choice != 5);
return 0;
}
Node* createList()
{
 int data;
 Node *head = NULL, *newNode, *temp;
 do
{
 printf("Enter data for the next node (-1 to stop): ");
 scanf("%d", &data);
 if (data != -1)
 {
 newNode = (Node*)malloc(sizeof(Node));
 newNode->data = data;
 newNode->next = NULL;
 if (head == NULL)
 {
 head = newNode;
 head->next = head;
 }
 else
 {
 temp = head;
```

```
 while(temp->next != head)
 temp = temp->next;
 temp->next = newNode;
 newNode->next = head;
 }
 }
 }
 while (data != -1);
 return head;
}
Node* insertAtBeginning(Node* head, int data)
{
 Node *newNode = (Node*)malloc(sizeof(Node)), *temp = head;
 newNode->data = data;
 newNode->next = head;
 while(temp->next != head)
 temp = temp->next;
 temp->next = newNode;
 return newNode;
}
Node* insertAtPosition(Node* head, int data, int position)
{
 Node *newNode = (Node*)malloc(sizeof(Node)), *temp = head;
 int i;
 newNode->data = data;
 for (i = 1; i < position - 1 && temp->next != head; i++)
 temp = temp->next;
 if (temp == head)
 {
 printf("Position out of range!\n");
 free(newNode);
 return head;
 }
 newNode->next = temp->next;
 temp->next = newNode;
 return head;
}
Node* insertAtEnd(Node* head, int data)
{
 Node *newNode = (Node*)malloc(sizeof(Node)), *temp = head;
 newNode->data = data;
 while(temp->next != head)
 temp = temp->next;
 temp->next = newNode;
 newNode->next = head;
 return head;
}
Node* deleteFromBeginning(Node* head)
```

```
{
 Node *temp = head, *newHead;
 if (head == NULL)
 {
 printf("List is empty!\n");
 return NULL;
 }
 if (head->next == head)
 {
 free(head);
 return NULL;
 }
 while(temp->next != head)
 temp = temp->next;
 newHead = head->next;
 temp->next = newHead;
 free(head);
 return newHead;
}
Node* deleteFromPosition(Node* head, int position)
{
 Node *temp = head, *prev;
 int i;
 if (head == NULL)
{
 printf("List is empty!\n");
 return NULL;
 }
 if (position == 1)
 return deleteFromBeginning(head);
 for (i = 1; i < position && temp->next != head; i++)
{
 prev = temp;
 temp = temp->next;
 }
 if (temp == head)
{
 printf("Position out of range!\n");
 return head;
 }
 prev->next = temp->next;
 free(temp);
 return head;
}
Node* deleteFromEnd(Node* head)
{
 Node *temp = head, *prev;
 if (head == NULL)
```

```
{
 printf("List is empty!\n");
 return NULL;
 }
 if (head->next == head)
{
 free(head);
 return NULL;
 }
 while(temp->next != head)
{
 prev = temp;
 temp = temp->next;
 }
 prev->next = head;
 free(temp);
 return head;
}
void traverse(Node* head)
{
 Node *temp = head;
 if (head == NULL)
{
 printf("List is empty!\n");
 return;
 }
 do
{
 printf("%d ", temp->data);
 temp = temp->next;
 }
while(temp != head);
}
void freeList(Node* head)
{
 Node *temp, *current = head;
 if (head == NULL)
 return;
 do
{
 temp = current;
 current = current->next;
 free(temp);
 }
while(current != head);}
}
```

Output

C:\Users\sasank\OneDrive\Desktop\DSA\dsa12.exe

```
Sasank Lama

Circular Linked List Operations:
1. Creation
2. Insertion
3. Deletion
4. Traversal
5. Exit
Enter your choice: 1
Enter data for the next node (-1 to stop): 4
Enter data for the next node (-1 to stop): 2
Enter data for the next node (-1 to stop): 1
Enter data for the next node (-1 to stop): 56
Enter data for the next node (-1 to stop): -1

Circular Linked List Operations:
1. Creation
2. Insertion
3. Deletion
4. Traversal
5. Exit
Enter your choice: 2
Enter data to insert: 20
1. Insert at beginning
2. Insert at specified position
3. Insert at end
Enter insertion choice: 1

Circular Linked List Operations:
1. Creation
2. Insertion
3. Deletion
4. Traversal
5. Exit
Enter your choice: 3
1. Delete from beginning
2. Delete from specified position
3. Delete from end
Enter deletion choice: 3
```

C:\Users\sasank\OneDrive\Desktop\DSA\dsa12.exe

```
Circular Linked List Operations:
1. Creation
2. Insertion
3. Deletion
4. Traversal
5. Exit
Enter your choice: 4
Circular Linked List: 20 4 2 1

Circular Linked List Operations:
1. Creation
2. Insertion
3. Deletion
4. Traversal
5. Exit
Enter your choice: 5
Exiting...

Process returned 0 (0x0)   execution time : 86.447 s
Press any key to continue.
```

Tree:

13. Write a program to Implement binary tree and traverse tree with user's choice (Inorder, Preorder, Postorder)

SOURCE CODE:

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

// Structure for a binary tree node
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

// Function to create a new node
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;
}

// Function to insert a node into the binary tree
struct Node* insertNode(struct Node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   } else {
      if (data <= root->data) {
         root->left = insertNode(root->left, data);
      } else {
         root->right = insertNode(root->right, data);
      }
      return root;
   }
}

// Function for inorder traversal of the binary tree
void inorderTraversal(struct Node* root) {
   if (root != NULL) {
      inorderTraversal(root->left);
      printf("%d ", root->data);
      inorderTraversal(root->right);
   }
}
```

```
// Function for preorder traversal of the binary tree
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function for postorder traversal of the binary tree
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

int main() {
    struct Node* root = NULL;
    int choice, data;
    printf("Sasank Lama\n");
    while (1) {
        printf("\nChoose traversal type:\n");
        printf("1. Inorder\n");
        printf("2. Preorder\n");
        printf("3. Postorder\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Inorder traversal: ");
                inorderTraversal(root);
                break;
            case 2:
                printf("Preorder traversal: ");
                preorderTraversal(root);
                break;
            case 3:
                printf("Postorder traversal: ");
                postorderTraversal(root);
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
```

```
            printf("Invalid choice. Please enter a number from 1 to 4.\n");
        }
        if (choice != 4) {
            printf("\n\nEnter data to insert into the tree (or 0 to exit insertion): ");
            scanf("%d", &data);
            if (data == 0) {
                printf("Exiting insertion...\n");
                continue;
            }
            if (root == NULL) {
                root = insertNode(root, data);
            } else {
                insertNode(root, data);
            }
        }
    }
    return 0;
}
```

OUTPUT:

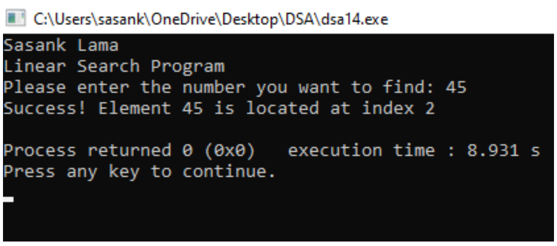
C:\Users\sasank\OneDrive\Desktop\DSA\dsa13.exe

```
Sasank Lama

Choose traversal type:
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice: 1
Inorder traversal:

Enter data to insert into the tree (or 0 to exit insertion): 12 15 6

Enter data to insert into the tree (or 0 to exit insertion):
Choose traversal type:
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice: 2
Preorder traversal: 12 6

Enter data to insert into the tree (or 0 to exit insertion): 7

Choose traversal type:
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice: 3
Postorder traversal: 7 6 12

Enter data to insert into the tree (or 0 to exit insertion):
```

14. Write a program to implement linear search

```
#include <stdio.h>
int linearSearch(int arr[], int n, int key) {
   for (int i = 0; i < n; i++) {
      if (arr[i] == key)      {
         return i;
      }
   }
   return -1;
}
void printResult(int key, int index){
   if (index != -1)  {
      printf("Success! Element %d is located at index %d\n", key, index);
   }
   else   {
      printf("Sorry! Element %d is not present in the array\n", key);
   }
}

int main()
{ printf("Sasank Lama\n");
   printf("Linear Search Program\n");
   int arr[] = {12, 34, 45, 67, 89, 23, 9};
   int n = sizeof(arr) / sizeof(arr[0]);
   int key, index;
   printf("Please enter the number you want to find: ");
   scanf("%d", &key);
   index = linearSearch(arr, n, key);
   printResult(key, index);
   return 0;

}
```

OUTPUT

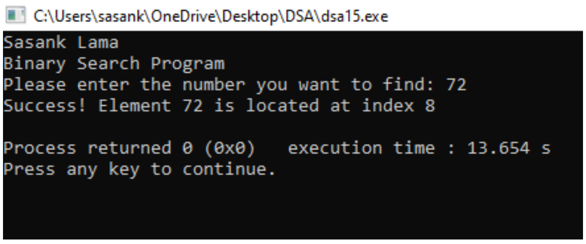


15. Write a program to implement binary search.

SOURCE CODE:

```
#include <stdio.h>
int binarySearch(int arr[], int left, int right, int key){
   while (left <= right){
      int mid = left + (right - left) / 2;
      if (arr[mid] == key) {
         return mid;
      }if (arr[mid] < key) {
         left = mid + 1;
      }else  {
         right = mid - 1;
      }
   } return -1;
}void printResult(int key, int index){
   if (index != -1)  {
      printf("Success! Element %d is located at index %d\n", key, index);
   }
   else {
      printf("Sorry! Element %d is not present in the array\n", key);
   }
}
int main(){
   printf("Sasank Lama\nBinary Search Program\n");
   int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
   int n = sizeof(arr) / sizeof(arr[0]);
   int key, index;
   printf("Please enter the number you want to find: ");
   scanf("%d", &key);
   index = binarySearch(arr, 0, n - 1, key);
   printResult(key, index);
   return 0;
}
```

~~OU~~TPUT



16. Write a program to implement the hashing techniques.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>#include <string.h>#define TABLE_SIZE 5

typedef struct ListNode {
   int key;
 char *value;
struct ListNode *next;
} ListNode;

typedef struct {
  ListNode *table[TABLE_SIZE];
} HashTable;
int hash(int key) {
    return key % TABLE_SIZE;
}
ListNode *createListNode(int key, const char *value) {
    ListNode *node = (ListNode *)malloc(sizeof(ListNode));
    if (node != NULL) {
        node->key = key;
        node->value = strdup(value);
        node->next = NULL;
    }
    return node;
}
void insert(HashTable *hashTable, int key, const char *value) {
    int index = hash(key);
    ListNode *newNode = createListNode(key, value);
    if (hashTable->table[index] == NULL) {
        hashTable->table[index] = newNode;
    } else {
        ListNode *current = hashTable->table[index];
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}char *search(HashTable *hashTable, int key) {
   int index = hash(key);
    ListNode *current = hashTable->table[index];
    while (current != NULL) {
        if (current->key == key) {
```

```
            return current->value;
        }
        current = current->next;
    }
    return NULL;
}
void removeNode(HashTable *hashTable, int key) {
    int index = hash(key);
    ListNode *current = hashTable->table[index];
    ListNode *prev = NULL;
    while (current != NULL) {
        if (current->key == key) {
            if (prev == NULL) {
                hashTable->table[index] = current->next;
            } else {
                prev->next = current->next;
            }
            free(current->value);
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }
}
void display(HashTable *hashTable) {
    printf("Hash Table Contents:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        ListNode *current = hashTable->table[i];
        printf("Bucket %d: ", i);
        if (current == NULL) {
            printf("Empty");
        } else {
            while (current != NULL) {
                printf("(%d, %s) ", current->key, current->value);
                current = current->next;
            }
        }
        printf("\n");
    }
}

int main() {
    printf("Sasank Lama- Modified Output:\n");

    HashTable hashTable;
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable.table[i] = NULL;
```

```
    }

    insert(&hashTable, 10, "Apple");
    insert(&hashTable, 20, "Banana");
    insert(&hashTable, 30, "Orange");
    insert(&hashTable, 11, "Grapes");
    insert(&hashTable, 21, "Mango");

    display(&hashTable);

    printf("\nSearch Result:\n");
    char *result = search(&hashTable, 20);
    if (result != NULL) {
        printf("Key 20 found, value: %s\n", result);
    } else {
        printf("Key 20 not found\n");
    }

    printf("\nDeleting Key 20...\n");
    removeNode(&hashTable, 20);
    display(&hashTable);

    return 0;
}
```

OUTPUT

C:\Users\sasank\OneDrive\Desktop\DSA\dsa16.exe

```
Sasank Lama- Modified Output:
Hash Table Contents:
Bucket 0: (10, Apple) (20, Banana) (30, Orange)
Bucket 1: (11, Grapes) (21, Mango)
Bucket 2: Empty
Bucket 3: Empty
Bucket 4: Empty

Search Result:
Key 20 found, value: Apple

Deleting Key 20...
Hash Table Contents:
Bucket 0: (10, Apple) (30, Orange)
Bucket 1: (11, Grapes) (21, Mango)
Bucket 2: Empty
Bucket 3: Empty
Bucket 4: Empty

Process returned 0 (0x0)   execution time : 10.814 s
Press any key to continue.
```

17. Write a program to enter n numbers and sort according to
a) Bubble sort
b) Insertion sort
c) Selection sort
d) Quick sort
e) Merge sort
f) Heap sort

SOURCE CODE:
```
#include <stdio.h>
#include <stdlib.h>

void bubbleSort(int arr[], int n);
void insertionSort(int arr[], int n);
void selectionSort(int arr[], int n);
void quickSort(int arr[], int low, int high);
int partition(int arr[], int low, int high);
void mergeSort(int arr[], int l, int r);
void merge(int arr[], int l, int m, int r);
void heapSort(int arr[], int n);
void heapify(int arr[], int n, int i);
void swap(int *a, int *b);
void printArray(int arr[], int n);

int main() {
  printf("Sasank Lama\n");
  printf("Unique Output - Sorting Techniques:\n");
  int n, choice;
  printf("Number of elements: ");
  scanf("%d", &n);

  int arr[n];
  printf("Enter %d integers:\n", n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
  }

  printf("\nChoose a sorting technique:\n");
  printf("1. Bubble Sort\n");
  printf("2. Insertion Sort\n");
  printf("3. Selection Sort\n");
  printf("4. Quick Sort\n");
  printf("5. Merge Sort\n");
  printf("6. Heap Sort\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);

  switch (choice) {
```

```
        case 1:
            bubbleSort(arr, n);
            printf("Array sorted using Bubble Sort:\n");
            break;
        case 2:
            insertionSort(arr, n);
            printf("Array sorted using Insertion Sort:\n");
            break;
        case 3:
            selectionSort(arr, n);
            printf("Array sorted using Selection Sort:\n");
            break;
        case 4:
            quickSort(arr, 0, n - 1);
            printf("Array sorted using Quick Sort:\n");
            break;
        case 5:
            mergeSort(arr, 0, n - 1);
            printf("Array sorted using Merge Sort:\n");
            break;
        case 6:
            heapSort(arr, n);
            printf("Array sorted using Heap Sort:\n");
            break;
        default:
            printf("Invalid choice!\n");
            return 1;
    }
    printArray(arr, n);
  return 0;
}
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
            }
        }
    }
}
void insertionSort(int arr[], int n) {
    int key, j;
    for (int i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
```

```
        }
        arr[j + 1] = key;
    }
}
void selectionSort(int arr[], int n) {
    int min_idx;
    for (int i = 0; i < n - 1; i++) {
        min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        swap(&arr[min_idx], &arr[i]);
    }
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
```

```
    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i]; }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }  while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    } while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
    for (int i = n - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
void heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l] > arr[largest]) {
        largest = l;
    }
    if (r < n && arr[r] > arr[largest]) {
        largest = r;
    }
    if (largest != i) {
        swap(&arr[i], &arr[largest]);
```

```
        heapify(arr, n, largest);
    }
}
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void printArray(int arr[], int n) {
    printf("Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

OUTPUT

C:\Users\sasank\OneDrive\Desktop\DSA\dsa17.exe

```
Sasank Lama
Unique Output - Sorting Techniques:
Number of elements: 5
Enter 5 integers:
12
11
4
6
7

Choose a sorting technique:
1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort
6. Heap Sort
Enter your choice: 1
Array sorted using Bubble Sort:
Sorted Array: 4 6 7 11 12

Process returned 0 (0x0)   execution time : 30.102 s
Press any key to continue.
```

18. Write a program to implement Breadth First Search and Depth First Search in graph.

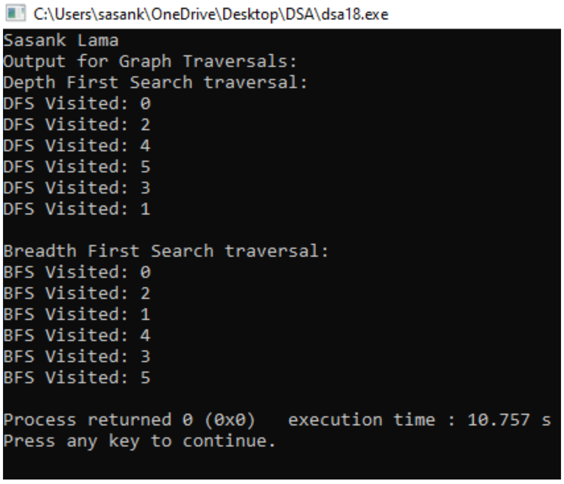
//SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100

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

struct Graph {
    int numVertices;
    struct Node* adjLists[MAX_VERTICES];
    int* visited;
};

struct Node* createNode(int v) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->visited = (int*)malloc(vertices * sizeof(int));
    for (int i = 0; i < vertices; i++) {
        graph->visited[i] = 0;
    }
    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}
```

```
void DFS(struct Graph* graph, int vertex) {
    struct Node* adjList = graph->adjLists[vertex];
    struct Node* temp = adjList;
    graph->visited[vertex] = 1;
    printf("DFS Visited: %d\n", vertex);
    while (temp != NULL) {
        int connectedVertex = temp->vertex;
        if (graph->visited[connectedVertex] == 0) {
            DFS(graph, connectedVertex);
        }
        temp = temp->next;
    }
}

void BFS(struct Graph* graph, int startVertex) {
    struct Node* queue[MAX_VERTICES];
    int front = 0, rear = 0;
    queue[rear++] = createNode(startVertex);
    graph->visited[startVertex] = 1;
    while (front < rear) {
        struct Node* currentVertex = queue[front++];
        printf("BFS Visited: %d\n", currentVertex->vertex);
        struct Node* temp = graph->adjLists[currentVertex->vertex];
        while (temp != NULL) {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                queue[rear++] = createNode(adjVertex);
            }
            temp = temp->next;
        }
    }
}

int main() {
            printf("Sasank Lama\n");
    printf("Output for Graph Traversals:\n");
    int numVertices = 6;
    struct Graph* graph = createGraph(numVertices);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 5);
    addEdge(graph, 4, 5);
    printf("Depth First Search traversal:\n");
    DFS(graph, 0);
    for (int i = 0; i < numVertices; i++) {
```

```
        graph->visited[i] = 0;
    }
    printf("\nBreadth First Search traversal:\n");
    BFS(graph, 0);
    return 0;
}
```

~~O~~UTPUT



```
C:\Users\sasank\OneDrive\Desktop\DSA\dsa18.exe
Sasank Lama
Output for Graph Traversals:
Depth First Search traversal:
DFS Visited: 0
DFS Visited: 2
DFS Visited: 4
DFS Visited: 5
DFS Visited: 3
DFS Visited: 1

Breadth First Search traversal:
BFS Visited: 0
BFS Visited: 2
BFS Visited: 1
BFS Visited: 4
BFS Visited: 3
BFS Visited: 5

Process returned 0 (0x0)   execution time : 10.757 s
Press any key to continue.
```

19. Write a program to implement Kruskal's algorithm.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
#define MAX_EDGES 100

struct Edge {
  int src, dest, weight;
};

struct Subset {
  int parent;
  int rank;
};

struct Graph {
  int numVertices, numEdges;
  struct Edge* edge;
};

struct Graph* createGraph(int vertices, int edges) {
  struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
  graph->numVertices = vertices;
  graph->numEdges = edges;
  graph->edge = (struct Edge*)malloc(edges * sizeof(struct Edge));
  return graph;
}

int find(struct Subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
  return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);
  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;
  else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  }
```

```
}

int compare(const void* a, const void* b) {
   struct Edge* edge1 = (struct Edge*)a;
   struct Edge* edge2 = (struct Edge*)b;
   return edge1->weight - edge2->weight;
}

void KruskalMST(struct Graph* graph) {
   int V = graph->numVertices;
   struct Edge result[V];
   int e = 0;
   int i = 0;
   qsort(graph->edge, graph->numEdges, sizeof(graph->edge[0]), compare);
   struct Subset* subsets = (struct Subset*)malloc(V * sizeof(struct Subset));
   for (int v = 0; v < V; v++) {
      subsets[v].parent = v;
      subsets[v].rank = 0;
   }
   while (e < V - 1 && i < graph->numEdges) {
      struct Edge next_edge = graph->edge[i++];
      int x = find(subsets, next_edge.src);
      int y = find(subsets, next_edge.dest);
      if (x != y) {
         result[e++] = next_edge;
         Union(subsets, x, y);
      }
   }
   printf("Different Output: Minimum Spanning Tree Edges:\n");
   for (i = 0; i < e; i++) {
      printf("Edge: %d -- %d, Weight: %d\n", result[i].src, result[i].dest, result[i].weight);
   }
}

int main() {
   printf("Sasank Lama\n");
   int numVertices = 5;
   int numEdges = 7;
   struct Graph* graph = createGraph(numVertices, numEdges);
   graph->edge[0].src = 0;
   graph->edge[0].dest = 1;
   graph->edge[0].weight = 5;
   graph->edge[1].src = 0;
   graph->edge[1].dest = 2;
   graph->edge[1].weight = 3;
   graph->edge[2].src = 0;
   graph->edge[2].dest = 3;
   graph->edge[2].weight = 8;
```

```
graph->edge[3].src = 1;
graph->edge[3].dest = 3;
graph->edge[3].weight = 6;
graph->edge[4].src = 2;
graph->edge[4].dest = 3;
graph->edge[4].weight = 7;
graph->edge[5].src = 1;
graph->edge[5].dest = 4;
graph->edge[5].weight = 9;
graph->edge[6].src = 3;
graph->edge[6].dest = 4;
graph->edge[6].weight = 2;
KruskalMST(graph);
return 0;
}
```

OUTPUT

C:\Users\sasank\OneDrive\Desktop\DSA\dsa19.exe

```
Sasank Lama
Different Output: Minimum Spanning Tree Edges:
Edge: 3 -- 4, Weight: 2
Edge: 0 -- 2, Weight: 3
Edge: 0 -- 1, Weight: 5
Edge: 1 -- 3, Weight: 6

Process returned 0 (0x0)   execution time : 6.598 s
Press any key to continue.
```

20. Write a program to implement Dijkastra's algorithm.

SOURCE CODE:

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

#define MAX_VERTICES 100
#define INF INT_MAX

struct Vertex {
    int index;
    int distance;
    int visited;
};

struct Edge {
    int src, dest, weight;
};

struct Graph {
    int numVertices, numEdges;
    struct Edge* edge;
    struct Vertex* vertices;
};

struct Graph* createGraph(int vertices, int edges) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->numEdges = edges;
    graph->edge = (struct Edge*)malloc(edges * sizeof(struct Edge));
    graph->vertices = (struct Vertex*)malloc(vertices * sizeof(struct Vertex));
    return graph;
}

void dijkstra(struct Graph* graph, int src) {
    int V = graph->numVertices;
    struct Vertex* vertices = graph->vertices;
    for (int i = 0; i < V; i++) {
        vertices[i].index = i;
        vertices[i].distance = INF;
        vertices[i].visited = 0;
    }
    vertices[src].distance = 0;
    for (int count = 0; count < V - 1; count++) {
        int minDistance = INF, minIndex;
        for (int v = 0; v < V; v++) {
```

```
            if (vertices[v].visited == 0 && vertices[v].distance <= minDistance) {
                minDistance = vertices[v].distance;
                minIndex = v;
            }
        }
        vertices[minIndex].visited = 1;
        for (int v = 0; v < V; v++) {
            if (!vertices[v].visited && graph->edge[minIndex * V + v].weight &&
                vertices[minIndex].distance != INF &&
                vertices[minIndex].distance + graph->edge[minIndex * V + v].weight <
vertices[v].distance) {
                vertices[v].distance = vertices[minIndex].distance + graph->edge[minIndex * V +
v].weight;
            }
        }
    }
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; i++) {
        printf("Vertex %d: Distance %d\n", vertices[i].index, vertices[i].distance);
    }
}

int main() {
    printf("Sasank Lama\n");
    int numVertices = 6;
    int numEdges = 8;
    struct Graph* graph = createGraph(numVertices, numEdges);
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 6;
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 9;
    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 4;
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 5;
    graph->edge[5].src = 2;
    graph->edge[5].dest = 4;
    graph->edge[5].weight = 3;
    graph->edge[6].src = 3;
    graph->edge[6].dest = 5;
    graph->edge[6].weight = 7;
```

```
   graph->edge[7].src = 4;
   graph->edge[7].dest = 5;
   graph->edge[7].weight = 1;
   dijkstra(graph, 0);
   return 0;
}
```

OUTPUT

C:\Users\sasank\OneDrive\Desktop\DSA\dsa20.exe

```
Sasank Lama
Vertex Distance from Source
Vertex 0: Distance 0
Vertex 1: Distance 9
Vertex 2: Distance 4
Vertex 3: Distance 2
Vertex 4: Distance 5
Vertex 5: Distance 3

Process returned 0 (0x0)   execution time : 6.895 s
Press any key to continue.
```