1(A). ARRAY IMPLEMENTATION OF STACK ADT

PROGRAM :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int top = 0, a[100], max = 50;
void main()
{
    void push();
    void pop();
    void display();
    int ch;
    clrscr();
    printf("\n1.Push\ n2.Pop\ n3.Display\ n4.Exit");
    while (1)
    {
        printf("\nEnter your choice: ");
        scanf(" % d", &ch);
        switch (ch)
        {
                case 1:
                        push();
    break;
                case 2:
                        pop();
                        break;
                case 3:
                        display();
                        break;
                default:
                        exit();
        }
    }
}

void push()
{
    if (top == max)
        printf("\n Array is full");
    else
    {
        printf("\n Enter the data: ");
        scanf(" % d", &x);
        st[top++] = x;
    }
}

void pop()
{
    if (top == 0)
        printf("\n Array is empty");
```

```
    else
    {
        top--;
    }
}

void display()
{
    int i;
    if (top == 0)
       printf("\n Stack is empty");
    else
    {
        printf("\n Array elememts are: ");
        for (i = 0; i < top; i++)
                printf(" % d\ t", st[i]);
    }
}
```

OUTPUT:

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the data: 4
Enter your choice: 1
Enter the data: 6
Enter your choice: 1
Enter the data: 8
Enter your choice: 3
Array elements are: 4 6 8
Enter your choice: 2
Enter your choice: 3
Array elements are: 4 6
```

1.(B). ARRAY IMPLEMENTATION OF QUEUE ADT

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int ch, que[50], max = 50, front = 0, rear = 0;
void main()
{
    void enqueue();
    void dequeue();
    void display();
    clrscr();
    printf("\n1.Insertion(enqueue)\ n2.Deletion(dequeue)\ n3.Display\ n4.Exit");
    while (1)
    {
        printf("Enter your choice:");
        scanf(" % d", &ch);
        switch (ch)
{
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            default:
                exit(0);
        }
    }
}

void enqueue()
{
    int ele;
    if (rear == max)
       printf("The queue is full");
    else
    {
        printf("Enter the data:");
        scanf(" % d", &ele); que[rear++] = ele;
    }
}

void dequeue()
{
    int i;
    if (rear == 0)
        printf("The queue is empty");
    else
    {
        printf("\n Deleted element is % d, qu[front]);
        front++;
    }
```

```
}

void display()
{
    int i;
    if (rear == 0)
        printf("The queue is empty”);
    else
    {
       printf(“\n Queue elememts are: ”);
       for (i = front; i < rear; i++)
                printf(“ % d\ t”, que[i]);
    }
}
```

OUTPUT:

```
1. Insertion (enqueue)
2. Deletion (dequeue)
3. Display
4. Exit
Enter your choice: 1
Enter the data: 6
Enter your choice: 1
Enter the data: 8
Enter your choice: 3
Queue elements are: 6 8
Enter your choice: 2
Deleted element is: 6
Enter your choice: 1
Enter the data: 10
Enter your choice: 3
Queue elements are: 8 10
```

1.(C). ARRAY IMPLEMENTATION OF CIRCULAR QUEUE ADT

PROGRAM:

```
#include <stdio.h >
#define max 6
int queue[max];
int front = -1;
int rear = -1;

void enqueue(int element)
{
    if (front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
        queue[rear] = element;
    }
    else if ((rear + 1) % max == front)
    {
        printf("Queue is overflow..");
    }
    else
    {
        rear = (rear + 1) % max;
        queue[rear] = element;
    }
}



int dequeue()
{
    if ((front == -1) && (rear == -1))
        printf("\nQueue is underflow.");
}
else if (front == rear)
{
    printf("\nThe dequeued element is %d", queue[front]);
    front = -1;
    rear = -1;
}
else
{
    printf("\nThe dequeued element is %d", queue[front]);
    front = (front + 1) % max;
}
}
// function to display the elements of a queue
void display()
{
    int i = front;
    if (front == -1 && rear == -1)
    {
        printf("\n Queue is empty..");
    }
```

```
    else
    {
        printf("\nElements in a Queue are :");
        while (i <= rear)
        {
            printf("%d,", queue[i]);
            i = (i + 1) % max;
        }
    }

int main()
{
    int choice = 1, x;    // variables declaration
    while (choice < 4 && choice != 0)  // while loop
    {
        printf("\n Press 1: Insert an element");
        printf("\nPress 2: Delete an element");
        printf("\nPress 3: Display the element");
        printf("\nEnter your choice");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter the element which is to be inserted");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
        }
}
    return 0;
 }
```

OUTPUT:

```
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
10

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
20

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
30

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
3

Elements in a Queue are :10,20,30,
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
2

The dequeued element is 10
```

2. IMPLEMENTATION OF SINGLY LINKED LIST

PROGRAM:

```
#include<stdio.h>
    #include<conio.h>
    #include<stdlib.h>
    struct node
    {
    int data;
    struct node*link;
    };
    struct node*head,*x,*y,*z;
    void main()
    {
    void create();
    void insbeg();
    void insmid();
    void insend();
    void delbeg();
    void delmid();
    void delend();
    void display();
    int ch;
    clrscr();
    while(1)
    {
    printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at middle \n 4.Insertion at
    end \n 5.Deletion at beginning  \n 6.Deletion at middle \n 7.Deletion at end \n 8.Display \n
    9.Exit");
    printf("Enter your choices");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:
    create();
    break;
    case 2:
    insbeg();
    break;
    case 3:
    insmid();
    break;
    case 4:
    insend();
    break;
    case 5:
    delbeg();
    break;
    case 6:
    delmid();
    break;
    case 7:
    delend();
    break;
    case 8:
    display();
    break;
    default:
```

```
exit(0);
}
}
}
void create()
{
int c;
head = NULL;
x=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the data ");
scanf("%d",&x->data);
x->link =NULL;
head=x;
printf("\n Do u wish to continue press 1 otherwise 0:");
scanf("%d",&c);
while(c!=0)
{
y=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d",&y->data);
y->link=NULL;
x->link=y;
x=y;
printf("\n Do you wish to continue press 1 otherwise 0:");
scanf("%d",&c);
}
}
void insbeg()
{
y=(struct node*)malloc(sizeof(struct node));
printf("\n Enter data:");
scanf("%d",&y->data);
y->link=head;
head=y;
}
void insmid()
{
int pos,c=1;
y=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d",&y->data);
printf("\n Enter the position to be insterted:");
scanf("%d",&pos);
x=head;
while(c<pos)
{
z=x;
x=x->link;
c++;
}
y->link=x;
z->link=y;
}
void insend()
{
y=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d",&y->data);
y->link=NULL;
```

```
x=head;
while(x->link!=NULL)
{
x=x->link;
}
x->link=y;
}
void delbeg()
{
if(head==NULL)
printf("List is empty");
else
{
x=head;
head=x->link;
free(x);
}
}
void delmid()
{
int pos,c=1;
if(head==NULL)
printf("\n List is empty");
else
{
printf("\nEnter the postion to be Deletion:");
scanf("%d",&pos);
x=head;
while(c<pos)
{
z=x;
x=x->link;
c++;
}
z->link=x->link;
free(x);
}
}
void delend()
{
if(head==NULL)
printf("list is empty");
else
{
x=head;
while(x->link!=NULL)
{
y=x;
x=x->link;
}
y->link=NULL;
free(x);
}
}
void display()
{
if(head==NULL)
printf("\n List is empty");
else
```

```
{
x=head;
printf("\n THE LIST ELEMENTS ARE \n");
while(x->link!=NULL)
{
printf("%d->",x->data);
x=x->link;
}
printf("%d",x->data);
}
}
```

OUTPUT:

```
1. Creation
2. Insertion at beginning
3. Insertion at middle
4. Insertion at end
5. Deletion at beginning
6. Deletion at middle
7. Deletion at end
8. Display
9. Exit
Enter ur choice: 1
Enter the data: 10
Enter ur choice: 1
Enter the data: 20
Enter ur choice: 1
Enter the data: 30
Enter ur choice: 1
Enter the data: 40
Enter ur choice: 1
Enter the data: 50
Do u wish to continue press 1 otherwise 0:1
Enter ur choice: 8
List elements are 10→20→30→40→50
Do u wish to continue press 1 otherwise 0:
Enter ur choice: 1
Enter the data: 25
Enter the position to be inserted: 2
Enter ur choice: 8
List elements are 10→20→25→30→40→50
Do u wish to continue press 1 otherwise 0:
Enter ur choice: 6
Enter the position to be deleted: 3
Do u wish to continue press 1 otherwise 0:1
Enter ur choice: 8
List elements are 10→20→25→40→50
Do u wish to continue press 1 otherwise 0:1
Enter ur choice: 7
Enter ur choice: 8
List elements are 10→20→25→30→40
```

3.(A). LINKED LIST IMPLEMENTATION OF STACK ADT

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node * link;
};

struct node *top = NULL, *x;
void main()
{
    void push();
    void pop();
    void display();
    int ch;
    clrscr();
    while (1)
    {
        printf("\n OPTIONS");
        printf("\n 1.Insertion (push)");
        printf("\n 2.Deletion (pop)");
        printf("\n 3.Display");
        printf("\n 4.Exit");
        printf("\n Enter ur choice:");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            default:
                exit(0);
        }
    }
}

void push()
{
    if (top == NULL)
    {
        x = (struct node *) malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d", &x->data);
        x->link = NULL;
        top = x;
    }
    else
    {
        x = (struct node *) malloc(sizeof(struct node));
```

```
            printf("\n Enter the data:");
            scanf("%d", &x->data);
            x->link = top;
            top = x;
        }
    }


    void pop()
    {
        if (top == NULL)
            printf("\n Stack is empty");
        else
        {
            x = top;
            printf("\n Popped Element is %d", x->data);
            top = top->link;
            free(x);
        }
    }

    void display()
    {
        x = top;
        printf("\n Stack Elements are\n");
        while (x->link != NULL)
        {
            printf("%d->", x->data);
            x = x->link;
        }

        printf("%d", x->data);
    }
```

OUTPUT:

```
OPTIONS
1. Insertion (push)
2. Deletion (pop)
3. Display
4. Exit
Enter ur choice: 1
Enter the data: 10
Enter ur choice: 1
Enter the data: 20
Enter ur choice: 1
Enter the data: 30
Enter ur choice: 3
Stack Elements are: 10 20 30
Enter ur choice: 2
Popped Element is: 30
Enter ur choice: 3
Stack Elements are: 10 20
```

3.(B). LINKED LIST IMPLEMENTATION OF LINEAR QUEUE ADT

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node * link;
};
struct node *front = NULL, *rear = NULL, *x;
void main()
{
    void enqueue();
    void dequeue();
    void display();
    int ch;
    clrscr();
    while (1)
    {
        printf("\n OPTIONS");
        printf("\n 1.Insertion (enqueue)");
        printf("\n 2.Deletion (dequeue)");
        printf("\n 3.Display");
        printf("\n 4.Exit");
        printf("\n Enter ur choice:");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            default:
                exit(0);
        }
    }
}

void enqueue()
{
    if (rear == NULL)
    {
        x = (struct node *) malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d", &x->data);
        x->link = NULL;
        rear = x;
```

```
            front = x;
        }
        else
        {
            x = (struct node *) malloc(sizeof(struct node));
            printf("\n Enter the data:");
            scanf("%d", &x->data);
            x->link = NULL;
            rear->link = x;
            rear = x;
        }
}


void dequeue()
{
    if (front == NULL)
        printf("\n Queue is empty");
    else
    {
        x = front;
        printf("\n Dequeued Element is %d", x->data);
        front = x->link;
        free(x);
    }
}

void display()
{
    x = front;
    printf("\n Queue Elements are\n");
    while (x->link != NULL)
    {
        printf("%d->", x->data);
        x = x->link;
    }

    printf("%d", x->data);
}
```

OUTPUT:

```
OPTIONS
1. Insertion (enqueue)
2. Deletion (dequeue)
3. Display
4. Exit
Enter ur choice: 1
Enter the data: 10
Enter ur choice: 1
Enter the data: 20
Enter ur choice: 1
Enter the data: 30
Enter ur choice: 1
Enter the data: 40
Enter ur choice: 3
Queue Elements are 10   20      30      40
Enter ur choice: 2
Dequeued Element is 10
Enter ur choice: 3
Queue Elements are 20   30      40
```

4. IMPLEMENTATION OF POLYNOMIAL MANIPULATION USING LINKED LIST

PROGRAM:

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>
struct link
{
    int coeff;
    int pow;
    struct link * next;
};
struct link *poly1 = NULL, *poly2 = NULL, *poly = NULL;
void create(struct link *node)
{
    char ch;
    do {
        printf("\n enter coeff:");
        scanf("%d", &node->coeff);
        printf("\n enter power:");
        scanf("%d", &node->pow);
        node->next = (struct link *) malloc(sizeof(struct link));
        node = node->next;
        node->next = NULL;
        printf("\n continue(y/n):");
        ch = getch();
    }

    while (ch == 'y' || ch == 'Y');
}

void show(struct link *node)
{
    while (node->next != NULL)
    {
        printf("%dx^%d", node->coeff, node->pow);
        node = node->next;
        if (node->next != NULL)
            printf("+");
    }
}

void polyadd(struct link *poly1, struct link *poly2, struct link *poly)
{
    while (poly1->next && poly2->next)
    {
        if (poly1->pow > poly2->pow)
        {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        else if (poly1->pow < poly2->pow)
        {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
```

```
            }
            else
            {
                    poly->pow = poly1->pow;
                    poly->coeff = poly1->coeff + poly2->coeff;
                    poly1 = poly1->next;
                    poly2 = poly2->next;
            }

            poly->next = (struct link *) malloc(sizeof(struct link));
            poly = poly->next;
            poly->next = NULL;
      }

      while (poly1->next || poly2->next)
      {
            if (poly1->next)
            {
                    poly->pow = poly1->pow;
                    poly->coeff = poly1->coeff;
                    poly1 = poly1->next;
            }

            if (poly2->next)
            {
                    poly->pow = poly2->pow;
                    poly->coeff = poly2->coeff;
                    poly2 = poly2->next;
            }

            poly->next = (struct link *) malloc(sizeof(struct link));
            poly = poly->next;
            poly->next = NULL;
      }
}

main()
{
      char ch;
      do
{
            poly1 = (struct link *) malloc(sizeof(struct link));
            poly2 = (struct link *) malloc(sizeof(struct link));
            poly = (struct link *) malloc(sizeof(struct link));
            printf("\nenter 1st number:");
            create(poly1);
            printf("\nenter 2nd number:");
            create(poly2);
            printf("\n1st Number:");
            show(poly1);
            printf("\n2nd Number:");
            show(poly2);
            polyadd(poly1, poly2, poly);
            printf("\nAdded polynomial:");
            show(poly);
            printf("\n add two more numbers:");
            ch = getch();
      }
      while (ch == 'y' || ch == 'Y');}
```

Output:

Enter 1st number:
Enter coeff: 5
Enter power: 4
Continue(y/n): y
Enter 2st number:
Enter coeff: 6
Enter power: 4
Continue(y/n): n
1st number: 5^4
2nd number: 6^4
Added polynomial: 11^4

5.(A) IMPLEMENTATION OF EVALUATING POSTFIX EXPRESSIONS

```
#include <stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
    stack[++top] = x;
}
int pop()
{
    return stack[top--];
}
int main()
{
    char exp[20];
    char *e;
    int n1, n2, n3, num;
    printf("Enter the expression :: ");
    scanf("%s", exp);
    e = exp;
    while (*e != '\0')
    {
        if (isdigit(*e))
        {
            num = *e - 48;
            push(num);
        }
        else
        {
            n1 = pop();
            n2 = pop();
            switch (*e)
            {
                case '+':
                    {
                        n3 = n1 + n2;
                        break;
                    }

                case '-':
                    {
                        n3 = n2 - n1;
                        break;
                    }

                case '*':
                    {
                        n3 = n1 * n2;
                        break;
                    }

                case '/':
                    {
                        n3 = n2 / n1;
                        break;
```

```
                                        }
                        }

                        push(n3);
                }

                e++;
        }
        printf("\nThe result of expression %s  =  %d\n\n", exp, pop());
        return 0;
}
```

Output:

```
Enter the expression: 245+*
The result of expression 245+*=18
```

5.(B). IMPLEMENTATION OF INFIX TO POSTFIX CONVERSION

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char str[50], stack1[50], stack2[50];
int i = 0, j, n, top1 = -1, top2 = -1;
int prec(char);
void main()
{
    int t = 0;
    clrscr();
    printf("\nEnter the infix expression:");
    scanf("%s", str);
    printf("%s", str);
    n = strlen(str);
    for (i = 0; i < n; i++)
    {
        t = 0;
        if (str[i] == ')')
        {
            stack1[++top1] = stack2[top2--];
            top2--;
            t = 1;
        }

        if (str[i] == '(')
        {
            stack2[++top2] = '(';
            t = 1;
        }

        if (t == 0)
        {
            if (((str[i] >= 'a') && (str[i] <= 'z')) || ((str[i] >= 'A') && (str[i] <= 'Z')))
                stack1[++top1] = str[i];
            else
            {
                if (top2 == -1)
                    stack2[++top2] = str[i];
                else
                {
                    if (prec(str[i]) > prec(stack2[top2]))
                    {
                        stack2[++top2] = str[i];
                    }
                    else
                    {
                        while (prec(str[i]) <= prec(stack2[top2]))
                            stack1[++top1] = stack2[top2--];
                        stack2[++top2] = str[i];
                    }
                }
            }
```

```
            }
        }

        printf("\nANSWER\n");
        while (top2 >= 0)
            stack1[++top1] = stack2[top2--];
        for (i = 0; i <= top1; i++)
            printf("%c", stack1[i]);
        getch();
}
int prec(char ch)
{
        int tt;
        if ((ch == '+') || (ch == '-'))
            tt = 2;
        if ((ch == '*') || (ch == '/'))
            tt = 3;
        if (ch == '(')
            tt = 1;
        return tt;
}
```

OutPut:

```
Enter the infix expression: (a+b)*c/d+e/f
ANSWER: ab+c*d/ef/+
```

6. IMPLEMENTATION OF BINARY SEARCH TREES

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
struct tree
{
    int data;
    struct tree * lchild;
    struct tree * rchild;
}
*t, *temp;
int element;
void inorder(struct tree *);
void preorder(struct tree *);
void postorder(struct tree *);
struct tree* create(struct tree *, int);
struct tree* find(struct tree *, int);
struct tree* insert(struct tree *, int);
struct tree* del(struct tree *, int);
struct tree* findmin(struct tree *);
struct tree* findmax(struct tree *);
void main()
{
    int ch;

    do {
        printf("\n\t\t\tBINARY SEARCH TREE");
        printf("\n\t\t\t****** **********");
        printf("\nMain Menu\n");
        printf("\n1.Create\n2.Insert\n3.Delete\n4.Find\n5.FindMin\n6.FindMax");
        printf("\n7.Inorder\n8.Preorder\n9.Postorder\n10.Exit\n");
        printf("\nEnter ur choice :");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("\nEnter the data:");
                scanf("%d", &element);
                t = create(t, element);
                inorder(t);
                break;
            case 2:
                printf("\nEnter the data:");
                scanf("%d", &element);
                t = insert(t, element);
                inorder(t);
                break;
            case 3:
                printf("\nEnter the data:");
```

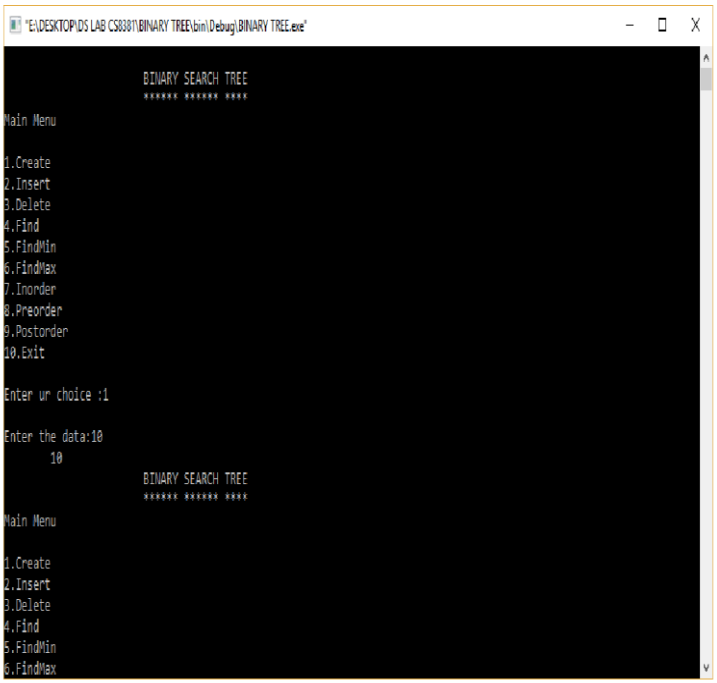
```
                        scanf("%d", &element);
                        t = del(t, element);
                        inorder(t);
                        break;
                case 4:
                        printf("\nEnter the data:");
                        scanf("%d", &element);
                        temp = find(t, element);
                        if (temp->data == element)
                                printf("\nElement %d is at %d", element, temp);
                        else
                                printf("\nElement is not found");
                        break;
                case 5:
                        temp = findmin(t);
                        printf("\nMax element=%d", temp->data);
                        break;
                case 6:
                        temp = findmax(t);
                        printf("\nMax element=%d", temp->data);
                        break;
                case 7:
                        inorder(t);
                        break;
                case 8:
                        preorder(t);
                        break;
                case 9:
                        postorder(t);
                        break;
                case 10:
                        exit(0);
            }
        }
while (ch <= 10);
}

struct tree* create(struct tree *t, int element)
{
    t = (struct tree *) malloc(sizeof(struct tree));
    t->data = element;
    t->lchild = NULL;
    t->rchild = NULL;
    return t;
}

struct tree* find(struct tree *t, int element)
{
    if (t == NULL)
        return NULL;
    if (element < t->data)
        return (find(t->lchild, element));
    else
    if (element > t->data)
        return (find(t->rchild, element));
```

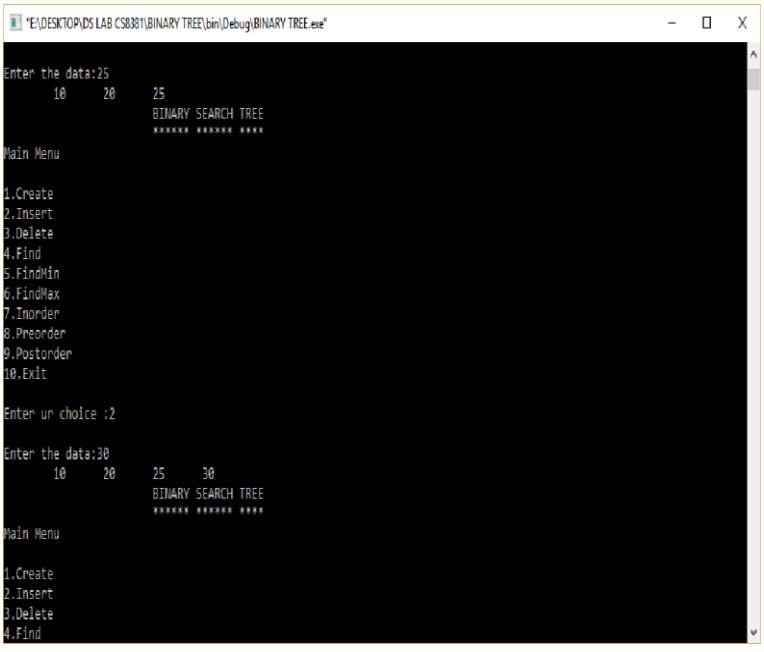
```
    else
        return t;
}

struct tree* findmin(struct tree *t)
{
    if (t == NULL)
        return NULL;
    else
    if (t->lchild == NULL)
        return t;
    else
        return (findmin(t->lchild));
}

struct tree* findmax(struct tree *t)
{
    if (t != NULL)
    {
        while (t->rchild != NULL)
            t = t->rchild;
    }

    return t;
}

struct tree* insert(struct tree *t, int element)
{
    if (t == NULL)
    {
        t = (struct tree *) malloc(sizeof(struct tree));
        t->data = element;
        t->lchild = NULL;
        t->rchild = NULL;
        return t;
    }
    else
    {
        if (element < t->data)
        {
            t->lchild = insert(t->lchild, element);
        }
        else
        if (element > t->data)
        {
            t->rchild = insert(t->rchild, element);
        }
        else
        if (element == t->data)
        {
            printf("element already present\n");
        }

        return t;
    }
```

```
}

struct tree* del(struct tree *t, int element)
{
    if (t == NULL)
        printf("element not found\n");
    else
    if (element < t->data)
        t->lchild = del(t->lchild, element);
    else
    if (element > t->data)
        t->rchild = del(t->rchild, element);
    else
    if (t->lchild && t->rchild)
    {
        temp = findmin(t->rchild);
        t->data = temp->data;
        t->rchild = del(t->rchild, t->data);
    }
    else
    {
        temp = t;
        if (t->lchild == NULL)
                t = t->rchild;
        else
        if (t->rchild == NULL)
                t = t->lchild;
        free(temp);
    }

    return t;
}
void inorder(struct tree *t)
{
    if (t == NULL)
        return;
    else
    {
        inorder(t->lchild);
        printf("\t%d", t->data);
        inorder(t->rchild);
    }
}

void preorder(struct tree *t)
{
    if (t == NULL)
        return;
    else
    {
        printf("\t%d", t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}
```

```
void postorder(struct tree *t)
{
    if (t == NULL)
        return;
    else
    {
        postorder(t->lchild);
        postorder(t->rchild);
        printf("\t%d", t->data);
    }
}
```

OUTPUT:





7. IMPLEMENTATION OF AVL TREES

PROGRAM:

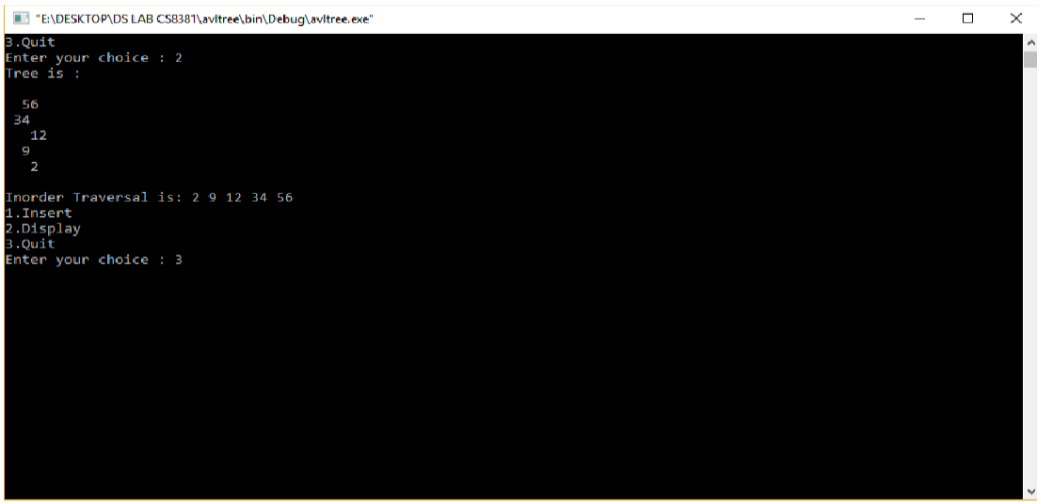
```
#include <stdio.h>
#include <malloc.h>
typedef enum
{
    FALSE, TRUE
}
bool;
struct node
{
    int info;
    int balance;
    struct node * lchild;
    struct node * rchild;
};
struct node* insert(int, struct node *, int *);
struct node* search(struct node *, int);
main()
{
    bool ht_inc;
    int info;
    int choice;
    struct node *root = (struct node *) malloc(sizeof(struct node));
    root = NULL;
    while (1)
    {
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Quit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter the value to be inserted : ");
                scanf("%d", &info);
                if (search(root, info) == NULL)
                    root = insert(info, root, &ht_inc);
                else
                    printf("Duplicate value ignored\n");
                break;
            case 2:
                if (root == NULL)
                {
                    printf("Tree is empty\n");
                    continue;
                }

                printf("Tree is :\n");
                display(root, 1);
                printf("\n\n");
                printf("Inorder Traversal is: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
```

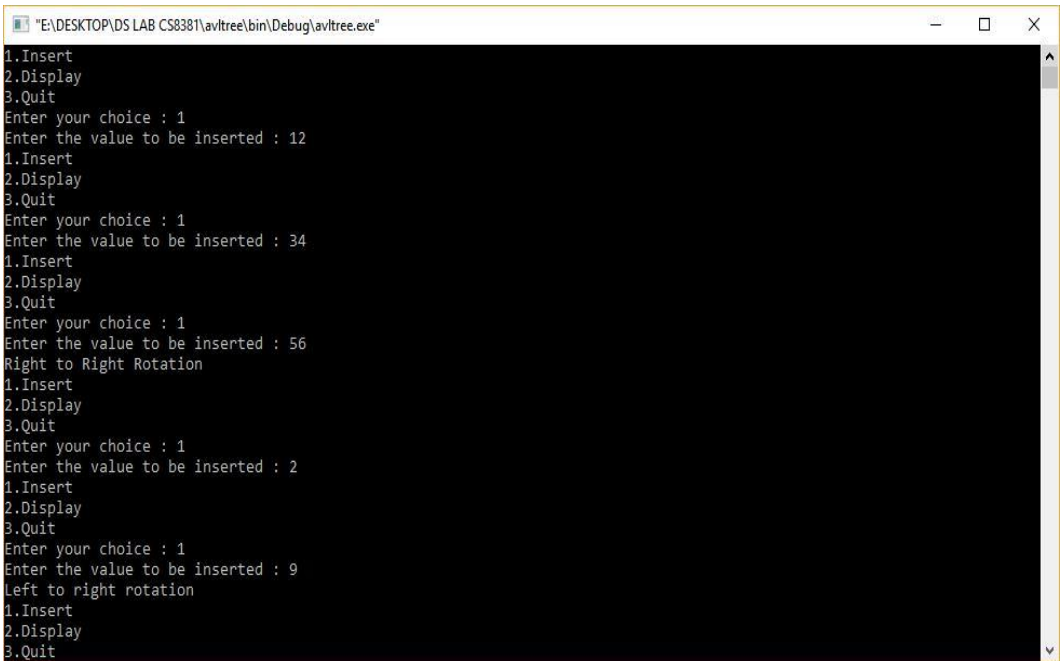
```
                        exit(1);
                default:
                        printf("Wrong choice\n");
            } /*End of switch*/
        } /*End of while*/
} /*End of main()*/
struct node* search(struct node *ptr, int info)
{
        if (ptr != NULL)
            if (info < ptr->info)
                    ptr = search(ptr->lchild, info);
            else if (info > ptr->info)
            ptr = search(ptr->rchild, info);
        return (ptr);
} /*End of search()*/
struct node* insert(int info, struct node *pptr, int *ht_inc)
{
        struct node * aptr;
        struct node * bptr;
        if (pptr == NULL)
        {
            pptr = (struct node *) malloc(sizeof(struct node));
            pptr->info = info;
            pptr->lchild = NULL;
            pptr->rchild = NULL;
            pptr->balance = 0;
            *ht_inc = TRUE;
            return (pptr);
        }

        if (info < pptr->info)
        {
            pptr->lchild = insert(info, pptr->lchild, ht_inc);
            if (*ht_inc == TRUE)
            {
                    switch (pptr->balance)
                    {
                            case -1:
                                    /*Right heavy */
                                    pptr->balance = 0;
                                    *ht_inc = FALSE;
                                    break;
                            case 0:
                                    /*Balanced */
                                    pptr->balance = 1;
                                    break;
                            case 1:
                                    /*Left heavy */
                                    aptr = pptr->lchild;
                                    if (aptr->balance == 1)
                                    {
                                            printf("Left to Left Rotation\n");
                                            pptr->lchild = aptr->rchild;
                                            aptr->rchild = pptr;
                                            pptr->balance = 0;
                                            aptr->balance = 0;
                                            pptr = aptr;
                                    }
                                    else
                                    {
```

```
                                        printf("Left to right rotation\n");
                                        bptr = aptr->rchild;
                                        aptr->rchild = bptr->lchild;
                                        bptr->lchild = aptr;
                                        pptr->lchild = bptr->rchild;
                                        bptr->rchild = pptr;
                                        if (bptr->balance == 1)
                                                pptr->balance = -1;
                                        else
                                                pptr->balance = 0;
                                        if (bptr->balance == -1)
                                                aptr->balance = 1;
                                        else
                                                aptr->balance = 0;
                                        bptr->balance = 0;
                                        pptr = bptr;
                                }

                                *ht_inc = FALSE;
                        } /*End of switch */
                } /*End of if */
        } /*End of if*/
        if (info > pptr->info)
        {
                pptr->rchild = insert(info, pptr->rchild, ht_inc);
                if (*ht_inc == TRUE)
                {
                        switch (pptr->balance)
                        {
                                case 1:
                                        /*Left heavy */
                                        pptr->balance = 0;
                                        *ht_inc = FALSE;
                                        break;
                                case 0:
                                        /*Balanced */
                                        pptr->balance = -1;
                                        break;
                                case -1:
                                        /*Right heavy */
                                        aptr = pptr->rchild;
                                        if (aptr->balance == -1)
                                        {
                                                printf("Right to Right Rotation\n");
                                                pptr->rchild = aptr->lchild;
                                                aptr->lchild = pptr;
                                                pptr->balance = 0;
                                                aptr->balance = 0;
                                                pptr = aptr;
                                        }
                                        else
                                        {
                                                printf("Right to Left Rotation\n");
                                                bptr = aptr->lchild;
                                                aptr->lchild = bptr->rchild;
                                                bptr->rchild = aptr;
                                                pptr->rchild = bptr->lchild;
                                                bptr->lchild = pptr;
                                                if (bptr->balance == -1)
                                                        pptr->balance = 1;
```

```
                                        else
                                                pptr->balance = 0;
                                        if (bptr->balance == 1)
                                                aptr->balance = -1;
                                        else
                                                aptr->balance = 0;
                                        bptr->balance = 0;
                                        pptr = bptr;
                                } /*End of else*/
                                *ht_inc = FALSE;
                        } /*End of switch */
                } /*End of if*/
        } /*End of if*/
        return (pptr);
} /*End of insert()*/
display(struct node *ptr, int level)
{
        int i;
        if (ptr != NULL)
        {
                display(ptr->rchild, level + 1);
                printf("\n");
                for (i = 0; i < level; i++)
                        printf(" ");
                printf("%d", ptr->info);
                display(ptr->lchild, level + 1);
        } /*End of if*/
} /*End of display()*/
inorder(struct node *ptr)
{
        if (ptr != NULL)
        {
                inorder(ptr->lchild);
                printf("%d ", ptr->info);
                inorder(ptr->rchild);
        }
}
```

OUTPUT:


"E:\DESKTOP\DS LAB CS8381\avltree\bin\Debug\avltree.exe"
3.Quit
Enter your choice : 2
Tree is :
56
34
12
9
2
Inorder Traversal is: 2 9 12 34 56
1.Insert
2.Display
3.Quit
Enter your choice : 3


"E:\DESKTOP\DS LAB CS8381\avltree\bin\Debug\avltree.exe"
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 12
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 34
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 56
Right to Right Rotation
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 2
1.Insert
2.Display
3.Quit
Enter your choice : 1
Enter the value to be inserted : 9
Left to right rotation
1.Insert
2.Display
3.Quit

8. IMPLEMENTATION OF HEAPS USING PRIORITY QUEUES

PROGRAM:

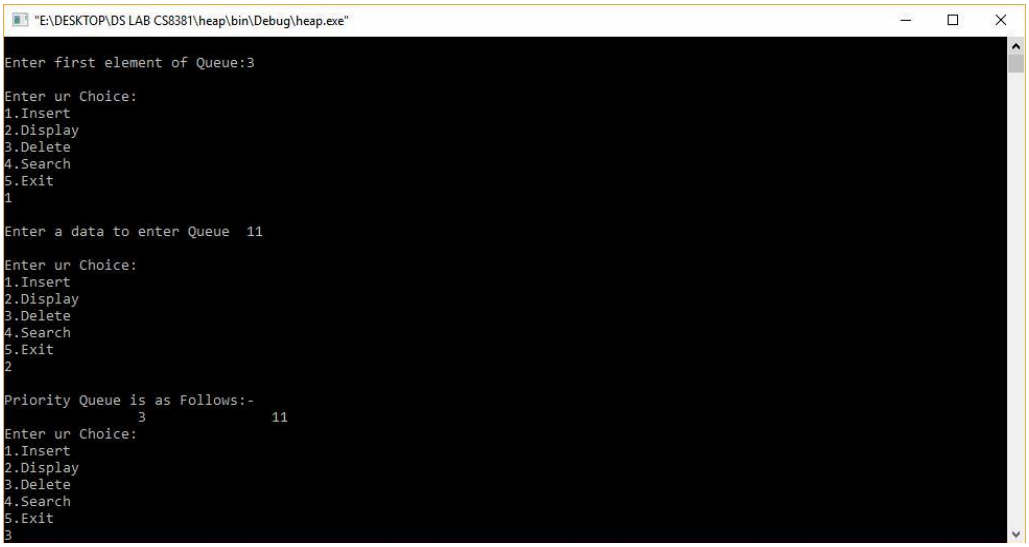
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
enum
{
    FALSE = 0, TRUE = -1
};
struct Node
{
    struct Node * Previous;
    int Data;
    struct Node * Next;
}
Current;
struct Node * head;
struct Node * ptr;
static int NumOfNodes;
int PriorityQueue(void);
int Maximum(void);
int Minimum(void);
void Insert(int);
int Delete(int);
void Display(void);
int Search(int);
void main()
{
    int choice;
    int DT;
    PriorityQueue();
    while (1)
    {
        printf("\nEnter ur Choice:");
        printf("\n1.Insert\n2.Display\n3.Delete\n4.Search\n5.Exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("\nEnter a data to enter Queue");

                scanf("%d", &DT);
                Insert(DT);
                break;
            case 2:
                Display();
                break;
            case 3:
                {
                    int choice, DataDel;
                    printf("\nEnter ur choice:");

printf("\n1.Maximum Priority queue\n2.Minimum priority Queue\n");
                    scanf("%d", &choice);
```

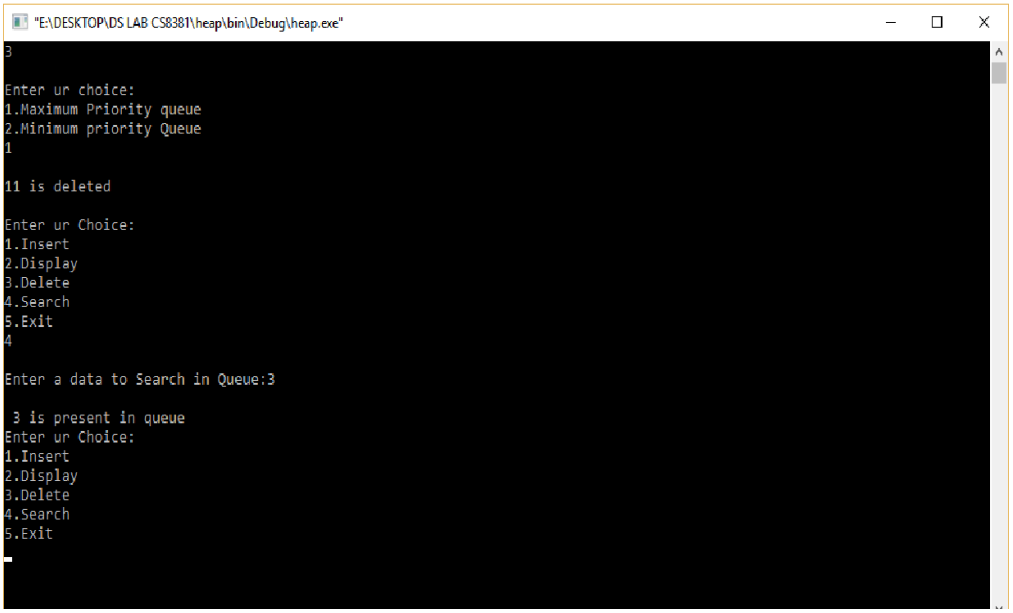
```
                                        switch (choice)
                                        {
                                                case 1:
                                                        DataDel = Maximum();
                                                        Delete(DataDel);

                                                        printf("\n%d is deleted\n", DataDel);
                                                        break;

                                                case 2:
                                                        DataDel = Minimum();
                                                        Delete(DataDel);
                                                        printf("\n%d is deleted\n", DataDel);
                                                        break;
                                                default:

                                                        printf("\nSorry Not a correct Choice\n");
                                        }
                                }

                                break;
                        case 4:

                                printf("\nEnter a data to Search in Queue:");
                                scanf("%d", &DT);

                                if (Search(DT) != FALSE)
                                        printf("\n %d is present in queue", DT);
                                else

                                        printf("\n%d is not present in queue", DT);
                                break;
                        case 5:
                                exit(0);
                        default:

                                printf("\nCannot process ur choice\n");
                }
        }
}



int PriorityQueue(void)
{
    Current.Previous = NULL;

    printf("\nEnter first element of Queue:");
    scanf("%d", &Current.Data);

    Current.Next = NULL;
    head = &Current;
    ptr = head;

    NumOfNodes++;
    return;
}
int Maximum(void)
{
    int Temp;
```

```
    ptr = head;
    Temp = ptr->Data;
    while (ptr->Next != NULL)

    {
        if (ptr->Data > Temp)

                Temp = ptr->Data;
        ptr = ptr->Next;
    }

    if (ptr->Next == NULL && ptr->Data > Temp)
        Temp = ptr->Data;
    return (Temp);
}


int Minimum(void)
{
    int Temp;
    ptr = head;
    Temp = ptr->Data;
    while (ptr->Next != NULL)
    {
        if (ptr->Data < Temp)
                Temp = ptr->Data;

        ptr = ptr->Next;
    }
    if (ptr->Next == NULL && ptr->Data < Temp)
        Temp = ptr->Data;
    return (Temp);
}
void Insert(int DT)
{
    struct Node * newnode;

    newnode = (struct Node *) malloc(sizeof(struct Node));
    newnode->Next = NULL;
    newnode->Data = DT;

    while (ptr->Next != NULL)
        ptr = ptr->Next;
    if (ptr->Next == NULL)
    {
        newnode->Next = ptr->Next;
        ptr->Next = newnode;
    }
    NumOfNodes++;
}

int Delete(int DataDel)
{
    struct Node *mynode, *temp;
    ptr = head;
    if (ptr->Data == DataDel)
    {
        temp = ptr;

        ptr = ptr->Next;
```

```
            ptr->Previous = NULL;
            head = ptr;
            NumOfNodes--;
            return (TRUE);
        }
        else
        {
            while (ptr->Next->Next != NULL)
            {
                if (ptr->Next->Data == DataDel)
                {
                    mynode = ptr;
                    temp = ptr->Next;
                    mynode->Next = mynode->Next->Next;
                    mynode->Next->Previous = ptr;
                    free(temp);
                    NumOfNodes--;
                    return (TRUE);
                }
                ptr = ptr->Next;
            }
            if (ptr->Next->Next == NULL && ptr->Next->Data == DataDel)
            {
                temp = ptr->Next;
                free(temp);
                ptr->Next = NULL;
                NumOfNodes--;
                return (TRUE);
            }
        }
        return (FALSE);
    }
    int Search(int DataSearch)
    {
        ptr = head;
        while (ptr->Next != NULL)
        {
            if (ptr->Data == DataSearch)
                return ptr->Data;
            ptr = ptr->Next;
        }
        if (ptr->Next == NULL && ptr->Data == DataSearch)

            return ptr->Data;
        return (FALSE);
    }
    void Display(void)
    {
        ptr = head;
        printf("\nPriority Queue is as Follows:-\n");
        while (ptr != NULL)
        {
            printf("\t\t%d", ptr->Data);
            ptr = ptr->Next;
        }
    }
```

OUTPUT:


"E:\DESKTOP\DS LAB CS8381\heap\bin\Debug\heap.exe"
Enter first element of Queue:3
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
1
Enter a data to enter Queue 11
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
2
Priority Queue is as Follows:-
3 11
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
3


"E:\DESKTOP\DS LAB CS8381\heap\bin\Debug\heap.exe"
3
Enter ur choice:
1.Maximum Priority queue
2.Minimum priority Queue
1
11 is deleted
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
4
Enter a data to Search in Queue:3
3 is present in queue
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit

9. IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

PROGRAM:

```
#include <limits.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

int printSolution(int dist[], int n)
    {
        printf("Vertex Distance from Source");
        for (int i = 0; i < V; i++)
        printf("%d \t %d", i, dist[i]);
    }

void dijkstra(int graph[V][V], int src)
    {
            int dist[V];
            bool sptSet[V];
            for (int i = 0; i < V; i++)
            dist[i] = INT_MAX, sptSet[i] = false;
            dist[src] = 0;
            for (int count = 0; count < V - 1; count++)
                {
                    int u = minDistance(dist, sptSet);
                    sptSet[u] = true;
                    for (int v = 0; v < V; v++)
                    if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] +
    graph[u][v] < dist[v]) dist[v] = dist[u] + graph[u][v];
                }

            printSolution(dist, V);
                }

int main()
{
    int graph[V][V] = {  { 0, 6, 0, 0, 0, 0, 0, 8, 0},
                        { 6, 0, 8, 0, 0, 0, 0, 13, 0 },
                        { 0, 8, 0, 7, 0, 6, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 6, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 13, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 }
                    };

        dijkstra(graph, 0);
```

```
    return 0;
}
```

Output:

```
Vertex Distance from Source
0 0
1 6
2 14
3 21
4 21
5 11
6 9
7 8
8 15
```

10. IMPLEMENTATION OF PRIM'S ALGORITHM

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX], spanning[MAX][MAX], n;
int prims();
int main()
{
    int i, j, total_cost;
    printf("Enter no. of vertices:");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &G[i][j]);
    total_cost = prims();
    printf("\nspanning tree matrix:\n");
    for (i = 0; i < n; i++)
    {
        printf("\n");
        for (j = 0; j < n; j++)
            printf("%d\t", spanning[i][j]);
    }
    printf("\n\nTotal cost of spanning tree=%d", total_cost);
    return 0;
}
int prims()
{
    int cost[MAX][MAX];
    int u, v, min_distance, distance[MAX], from[MAX];
    int visited[MAX], no_of_edges, i, min_cost, j;
    //create cost[][] matrix,spanning[][]
    for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
    {
        if (G[i][j] == 0)
            cost[i][j] = infinity;
        else
            cost[i][j] = G[i][j];
            spanning[i][j] = 0;
    }
    //initialise visited[],distance[] and from[]
    distance[0] = 0;
    visited[0] = 1;
    for (i = 1; i < n; i++)
    {
        distance[i] = cost[0][i];
        from[i] = 0;
        visited[i] = 0;
    }
    min_cost = 0;           //cost of spanning tree
    no_of_edges = n - 1; //no. of edges to be added
```

```
    while (no_of_edges > 0)
    {
        //find the vertex at minimum distance from the tree
        min_distance = infinity;
        for (i = 1; i < n; i++)
            if (visited[i] == 0 && distance[i] < min_distance)
            {
                v = i;
                min_distance = distance[i];
            }
        u = from[v];
        //insert the edge in spanning tree
        spanning[u][v] = distance[v];
        spanning[v][u] = distance[v];
        no_of_edges--;
        visited[v] = 1;
        //updated the distance[] array
        for (i = 1; i < n; i++)
            if (visited[i] == 0 && cost[i][v] < distance[i])
            {
                distance[i] = cost[i][v];
                from[i] = v;
            }
        min_cost = min_cost + cost[u][v];
    }
    return (min_cost);
}
```

OUTPUT:

```
Enter no. of vertices:6
Enter the adjacency matrix:
    0 3 1 6 0 0
    3 0 5 0 3 0
    1 5 0 5 6 4
    6 0 5 0 0 2
    0 3 6 0 0 6
    0 0 4 2 6 0

spanning tree matrix:
0 3 1 0 0 0
    3 0 0 0 3 0
    1 0 0 0 0 4
    0 0 0 0 0 2
    0 3 0 0 0 0
    0 0 4 2 0 0
Total cost of spanning tree=13
```

PROGRAM:

```
#include<stdio.h>
int a[10],i,n,flag=0;
void insert()
{
printf("Enter the size of an array: "); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the elements ofthearray%d:",i+1);
scanf("%d",&a[i]);
}
}
void delet()
{
int del;
printf("Enter the number to be delete: "); scanf("%d",&del);

for(i=0;i<n;i++){ if(a[i]==del){
a[i]=-1;
flag=1; break;
}
}
if(flag==0)
printf("The number is not in the list");
else
printf("The number is deleted");
}
void display()
{
printf("\nThe Element are:"); for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
}
void search()
{
int key;
printf("Enter the number to be search: "); scanf("%d",&key);
for(i=0;i<n;i++)
{
if(a[i]==key)
{
flag=1; break;
}
}
if(flag==0)
printf("The number is not in the list");
```

```
else
printf("The number is found");
}
int main()
{
int ch; while(ch!=5)
{
printf("\n1. Insert \n2. Delete \n3. Display \n4. Search\n5. Exit"); printf("\n Enter your Choice:");
scanf("%d",&ch); switch(ch)
{
case 1:
insert(); break; case 2:
delet(); break; case 3:
display(); break; case 4:
search(); break; case 5:
exit(0); break; default:
printf("\nInvalid Choice");
}
}
}
```

OUTPUT:

```
Insert
Delete
Display
Search
Exit
Enter your Choice:1
Enter the size of an array: 5
Enter the elements of the array 1:10 Enter the elements of the array 2:20 Enter the elements of the array 3:30 Enter the elements of the array 4:40 Enter the elements of the array 5:50
Insert
Delete
Display
Search
Exit
Enter your Choice:2
Enter the number to be delete: 30 The number is deleted
Insert
Delete
Display
Search
Exit
Enter your Choice:3
The Element are:10 20 -1 40 50
Insert
```

Delete
Display
Search
Exit

Enter your Choice:4
Enter the number to be search: 50 The number is found
Insert
Delete
Display
Search
Exit
Enter your Choice:

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct treenode;
typedef struct   treenode*position,*searchtree,ptrtonode;
struct treenode
{
int element;
struct treenode *left; struct treenode*right;
};
searchtree insert(int x,searchtree t)
{

if(t== NULL)
{
t=(ptrtonode*)malloc(sizeof(struct treenode)); if(t== NULL)
{
printf("\nOut of Space");
}
else
{
t->element=x; t->left=NULL;
t->right=NULL;
}}
else if(x<t->element)
{
t->left=insert(x,t->left);
}
else if(x>t->element)
{
t->right=insert(x,t->right);
}
return t;
}
position find(int x,searchtree t)
{
if(t== NULL)
{
return NULL;
}
else if(x<t->element)
{
return find(x,t->left);
}
else if(x>t->element)

{
return find(x,t->right);
```

```
}
else
{
return t;
}}
searchtree delet(int x,searchtree t)
{
position tmp; if(t== NULL)
{ }
else if(x<t->element)
{
t->left=delet(x,t->left);
}
else if(x>t->element)
{
t->right=delet(x,t->right);
}
else
{
if(t->right&&t->left)
{
tmp=find(x,t->right);
t->element=tmp->element;
t->right=delet(tmp->element,t->right);
}
else
{
tmp=t;
if(t->left== NULL)
{
t=t->right;
}

else if(t->right== NULL)
{
t=t->left;
}
free(tmp);
}}
return t;
}
void display(searchtree t)
{ if(t!=NULL)
{
display(t->left); printf("%d\t",t->element); display(t->right);
}}
void main()
{
```

```
int op,item; char ch; searchtree t; position p; clrscr(); t=NULL;
do
{
printf("\n1.Insert\n2.Delete\n3.Find\n4.Display"); printf("\nEnter your choice:");
scanf("%d",&op); switch(op)
{
case 1:printf("\nEnter the item to be inserted:"); scanf("%d",&item);
t=insert(item,t); printf("\nItem is inserted");

break;
case 2:printf("\nEnter the item to be deleted:"); scanf("%d",&item);
if(find(item,t))
{
t=delet(item,t); printf("\nItem is deleted");
}
else
{
printf("\nThe item is not found");
}
break;
case 3:printf("\nEnter the item to be found:"); scanf("%d",&item);
if(find(item,t))
{
printf("\nItem is found");
}
else
{
printf("\nThe item is not found");
}
break; case 4:if(t)
{
printf("The Tree is \n");
display(t);
}
else
{
printf("\nThe Tree is empty");
}
break; default:

printf("\nWrong Choice"); exit(1);
}
printf("\nDo u wish to continue (y for YES/n for NO):"); ch=getche();
}while(ch== 'y'); getch();
}
```

OUTPUT:

Insert 2.Delete 3.Find 4.Display

Enter your choice: 1
Enter the item to be inserted: 2 Item is inserted
Do u wish to continue (y for YES/n for NO): y 1.Insert
Delete 3.Find 4.Display
Enter your choice: 1
Enter the item to be inserted: 4 Item is inserted
Do u wish to continue (y for YES/n for NO): y 1.Insert
2.Delete 3.Find 4.Display
Enter your choice: 1
Enter the item to be inserted: 6 Item is inserted
Do u wish to continue (y for YES/n for NO): y 1.Insert
2.Delete 3.Find 4.Display
Enter your choice: 4 The Tree is ....
2 4 6
Do u wish to continue (y for YES/n for NO): y 1.Insert
2.Delete 3.Find 4.Display
Enter your choice: 3
Enter the item to be found: 4 Item is found
Do u wish to continue (y for YES/n for NO): y 1.Insert
2.Delete 3.Find 4.Display
Enter your choice: 2
Enter the item to be deleted: 4 Item is deleted
Do u wish to continue (y for YES/n for NO): y 1.Insert
Delete 3.Find 4.Display
Enter your choice: 4 The Tree is ....
2 6
Do u wish to continue (y for YES/n for NO): n

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void insert(int[],int);
void main()
{
int a[20],i,n; clrscr();
printf("\nEnter the size of an array:"); scanf("%d",&n);
for(i=0;i<n;i++){
printf("\nEnter the %d element in the array:",i+1); scanf("%d",&a[i]);
}
insert(a,n);
getch();
}

void insert(int a[],int n){ int i,j,temp; for(i=1;i<n;i++){ temp=a[i];
for(j=i-1;j>=0;j--){ if(a[j]>temp){
a[j+1]=a[j];
}
else
break;
}
a[j+1]=temp;
}
printf("\nData After Insertion sort\n"); for(i=0;i<n;i++)
printf("%d\t", a[i]);
}
```

OUTPUT:

Enter the size of an array:5
Enter the 1 element in the array:8 Enter the 2 element in the array:2 Enter the 3 element in the array:6 Enter the 4 element in the array:4 Enter the 5 element in the array:1 Data After Insertion sort
1 2 4 6 8

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void SelectionSort
(int arr[], int n)
{
int i, j;
for (i = 0; i < n; ++i)
{
for (j = i+1; j < n; ++j)
{
if (arr[i] > arr[j])
{
arr[i] = arr[i]+arr[j];
arr[j] = arr[i]-arr[j];
arr[i] = arr[i]-arr[j];
}
}
}
}
int main()
{
clrscr(); int n, i;
printf("\nEnter the number of data element to be sorted: "); scanf("%d",&n);
int arr[10];
for(i = 0; i <n ; i++)
{
printf("Enter %d element:",i+1); scanf("%d",&arr[i]);
}
SelectionSort(arr, n);
printf("\nSorted Data:\n");
for (i = 0; i < n; i++)
printf("%d \t ",arr[i]); getch();
}
```

OUTPUT:

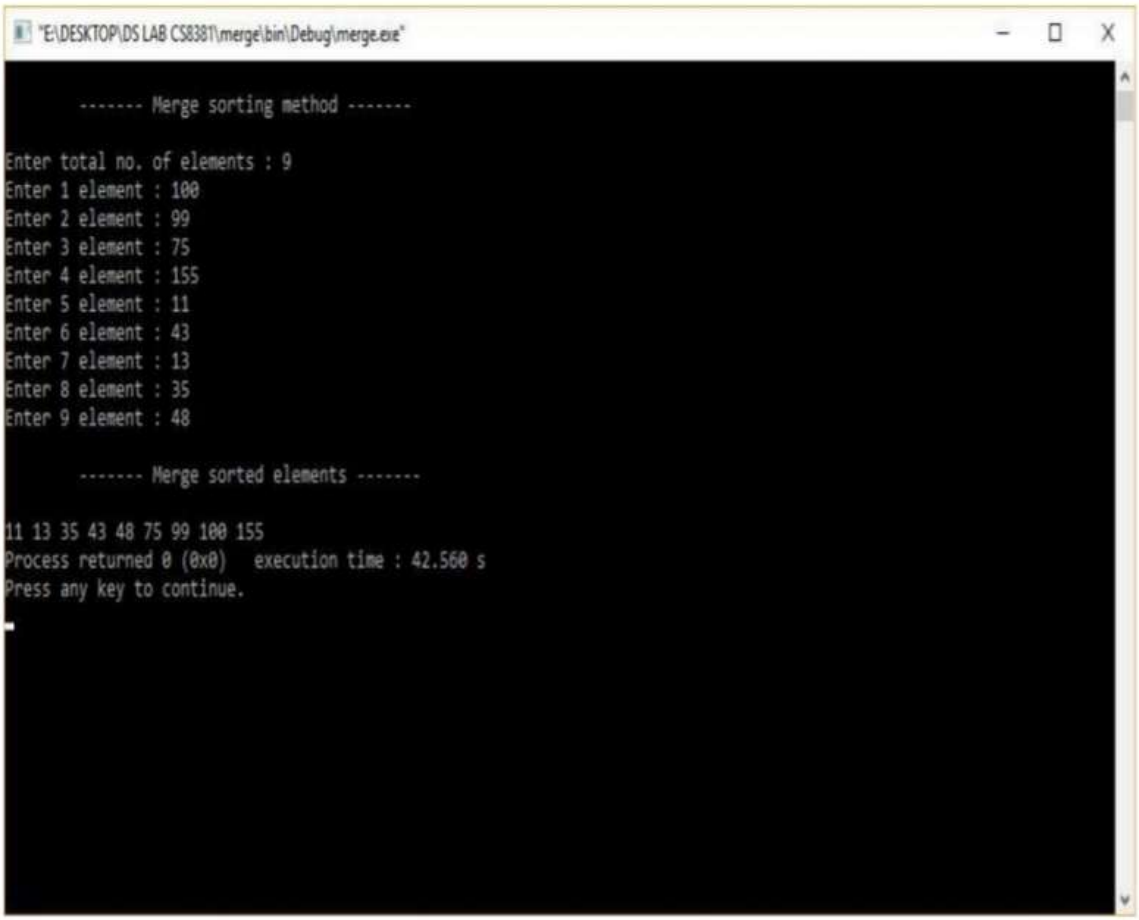
Enter the number of data element to be sorted: 5 Enter 1 element:10
Enter 2 element:2
Enter 3 element:8
Enter 4 element:4
Enter 5 element:6 Sorted Data:
2 4 6 8 10

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
void merge(int[], int, int, int);
void part(int[], int, int);
void main()
{
    int arr[30];
    int i, size;
    printf("\n\t------- Merge sorting method -------\n\n");
    printf("Enter total no. of elements : ");
    scanf("%d", &size);
    for (i = 0; i < size; i++)
    {
        printf("Enter %d element : ", i + 1);
        scanf("%d", &arr[i]);
    }
    part(arr, 0, size - 1);
    printf("\n\t------- Merge sorted elements -------\n\n");
    for (i = 0; i < size; i++)
    printf("%d ", arr[i]);
    getch();
}
void part(int arr[], int min, int max)
{
    int mid;
    if (min < max)
    {
        mid = (min + max) / 2;
        part(arr, min, mid);
        part(arr, mid + 1, max);
        merge(arr, min, mid, max);
    }
}
void merge(int arr[], int min, int mid, int max)
{
    int tmp[30];
    int i, j, k, m;
    j = min;
    m = mid + 1;
    for (i = min; j <= mid && m <= max; i++)
    {
        if (arr[j] <= arr[m])
        {
                tmp[i] = arr[j];
                j++;
        }


else
        {
                tmp[i] = arr[m];
                m++;
        }
    }
    if (j > mid)
```

```
    {
        for (k = m; k <= max; k++)
        {
            tmp[i] = arr[k];
            i++;
        }
    }
    else
    {
        for (k = j; k <= mid; k++)
        {
            tmp[i] = arr[k];
            i++;
        }
    }
    for (k = min; k <= max; k++)
        arr[k] = tmp[k];
}
```

OUTPUT:



PROGRAM

```
#include<stdio.h>
#include<conio.h> #include<stdlib.h> #define MAX 10
int create(int num)
{
int key; key=num%10; return key;
}
void display(int a[MAX])
{
int i;
printf("\nThe Hash Table is \n");
for(i=0;i<MAX;i++)
{
printf("\n %d %d",i,a[i]);
}
}
void linearprob(int a[MAX],int key,int num)
{
int flag=0,i,count=0; if(a[key]= = -1)
{
a[key]=num;
}
else
{
i=0;
while(i<MAX)
{
if(a[i]!=-1)
{
count++;
} i++;
}
if(count= =MAX)
{
printf("\nHash Table is Full"); display(a);
getch();
exit(1);
}
for(i=key+1;i<MAX;i++)
{
if(a[i]= = -1)
{
a[i]=num; flag=1; break;
}
}
for(i=0;i<key&&flag= = 0;i++)
{
```

```
if(a[i]= = -1)
{
a[i]=num; flag=1; break;
}
}
}
}
void main()
{
int a[max],num,key,i; char ans;
clrscr();
printf("\nCollision Handling by Linear Probing");
for(i=0;i<MAX;i++)
{
a[i]=-1;
}
do
{
printf("\nEnter the Number "); scanf("%d",&num); key=create(num); linearprob(a,key,num);
printf("\nDo u want to continue?(y for YES/n for NO): ");
ans=getche();
}while(ans= = 'y'); display(a);
getch();
}
```

OUTPUT:

```
Collision Handling by Linear Probing Enter the Number 131
Do u want to continue? (y for YES/n for NO): y Enter the Number 21
Do u want to continue? (y for YES/n for NO): y Enter the Number 3
Do u want to continue? (y for YES/n for NO): y Enter the Number 4
Do u want to continue? (y for YES/n for NO): y Enter the Number 5
Do u want to continue? (y for YES/n for NO): y Enter the Number 8
Do u want to continue? (y for YES/n for NO): y Enter the Number 9
Do u want to continue? (y for YES/n for NO): y Enter the Number 18
Do u want to continue? (y for YES/n for NO): n The Hash Table is ....
0 18
1 131
2 21
3 3
4 4
5 5
6 -1
```

7 -1
8 8
9 9

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void printArray(int arr[], int n)
{
printf("The Quadratic Probing Hash table"); for (int i = 0; i < n; i++)
{
printf("\n[%d] %d",i,arr[i]);
}
}
void hashing(int table[], int tsize, int arr[], int N)
{
for (int i = 0; i < N; i++)
{
int hv = arr[i] % tsize; if (table[hv] == -1) table[hv] = arr[i];
else
{
for (int j = 0; j < tsize; j++)
{
int t = (hv + j * j) % tsize; if (table[t] == -1)
{
table[t] = arr[i];
break;
}
}
}
}
printArray(table, N);
}
void main()
{
int arr[MAX],i,n,s=10; clrscr();
printf("\nEnter the size of n:"); scanf("%d",&n); for(i=0;i<n;i++)

{
printf("\nEnter the Number: "); scanf("%d",&arr[i]);
}
int hash_table[MAX]; for (int i = 0; i < s; i++)
{
hash_table[i] = -1;
}
hashing(hash_table, s, arr, n); getch();
}
```

OUTPUT:

Enter the size of n:7 Enter the Number: 50 Enter the Number: 700 Enter the Number: 76 Enter the Number: 85 Enter the Number: 92 Enter the Number: 73 Enter the Number: 101

The Quadratic Probing Hash table [0] 50
[1] 700
[2] 92
[3] 73
[4] -1
[5] 85
[6] 76