1. Merge two sorted arrays and store it in a third array.

AIM : Program to sort two arrays and store it in a third array.

ALGORITHM :

STEP 1: START

STEP 2: Declare two arrays a and b

STEP 3: Read the elements of the arrays

STEP 4: Traverse both the arrays

STEP 5: Find the minimum element among the current two elements of both the arrays and update the merge array with the least value and increment its index to next position.

STEP 6: Repeat procedure until both arrays are exhausted and return merge array.

STEP 7: STOP

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
void merge (int [], int, int [], int, int []);
void main ()
{
 int a [100], b [100], m, n, c, sort [200];
 clrscr();
 printf("Enter no of elements in 1st array:");
 scanf("%d", &m);
```

```
 printf("\nEnter the elements:", m);
 for (c=0; c<m;c++)
 {
  scanf("%d", &a[c]);
 }
 printf("\nEnter the no of elements in 2nd
array:");  scanf("%d", &n);
 printf("\nEnter the elements", n);
 for (c = 0; c < n; c++)
 {
  scanf("%d", &b[c]);
 }
 merge (a, m, b, n, sort);
 printf("\nSorted array:");
 for (c = 0; c < m + n; c++) {
 printf("%d\t", sort[c]);
 }
 getch();
}
void merge (int a [], int m, int b [], int n, int sorted
 []) { int i, j, k;
 j = k = 0;
 for (i = 0; i < m + n;)
 {
  if (j < m && k < n)
 {
  if (a[j] < b[k])
```

```
{
 sorted[i] = a[j];
  j++;
}
Else
{
sorted[i] = b[k];
 k++;
}
i++; }
else if (j == m) {
for (; i < m + n;) {
sorted[i] = b[k];
k++;
i++; }
}
else {
for (; i < m + n;) {
sorted[i] = a[j];
j++;
i++;
}
}
} }
```

RESULT : The required output is obtained.

OUTPUT

```
Enter no of elements in 1st array: 3

Enter the elements:1 3 5

Enter the no of elements in 2nd array: 3

Enter the elements: 4 6 9

Sorted array: 1 3       4       5       6       9
```

2. IMPLEMENT STACK AS AN ARRAY

AIM : Program to implement stack as an array

ALGORITHM :

STEP 1: START

STEP 2: Declare all the functions used in stack implementation.

STEP 3: Create a one dimensional array with fixed size (int stack[100])

STEP 4: Define a integer variable 'top' and initialize with '-1'. (int top = -1)

STEP 5: In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

STEP 6: For **push**() operations, check whether stack is full. (top == n-1)

STEP 7: If it is full, then display **"**Overflow**"** and terminate the function.

STEP 8: If it is not full, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

STEP 10 : For **pop**() operation: Check whether stack is EMPTY. (top == -1)

STEP 11: If it is empty, then display **"**Underflow**"** and terminate the function.

STEP 12: If it is not empty, then delete stack[top] and decrement top value by one (top--).

STEP 13: **display**() the elements using for loop and print stack[i]).

STEP 14: Display the top element, **topele**() by placing the topmost element of stack in val and print val.

STEP 15: STOP.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int stack[100],i,j,ch,n,top=-1;
void push();
void pop();
void display();
void topele();
void main()
{
clrscr();
printf("Enter the no of elements:");
scanf("%d",&n);
printf("\nOperations on stack");
 while(ch!=5)
 {
printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.TOP ELEMENT\n5.EXIT");
printf("\nEnter the choice:");
scanf("%d",&ch);
  switch(ch)
   {
    case 1:
     {
push();
```

```
      break;
     }
    case 2:
     {
pop();
      break;
     }
    case 3:
     {
display();
      break;
     }
     case 4:
     {
topele();
      break;
     }
     case 5:
     {
printf("\nExiting");
      break;
     }
default:printf("Invalid Choice");
   }
 };
getch();
```

```
}
void push()
{
 int val;
 if(top==n-1)
printf("\nOverflow");
 else
  {
printf("\nEnter the element:");
scanf("%d",&val);
   top++;
   stack[top]=val;
   }
 }
void pop()
{
 if(top==-1)
printf("\nUnderflow");
 else
  {
   int val;
val=stack[top];
   top--;
printf("\nElement deleted");
  }
}
```

```
void display()
{
printf("\nThe elements are:");
 for(i=top;i>=0;i--)
  {
printf("%d\t",stack[i]);
  }
  if(top==-1)
printf("\nStack is empty");
}
void topele()
{
 int val;
 if(top==-1)
printf("\nStack empty");
 else
  {
val=stack[top];
printf("\nThe topmost element is: %d",val);
  }
}
```

RESULT: The required output is obtained.

OUTPUT

```
Enter the no of elements:3

Operations on stack
1.PUSH
2.POP
3.DISPLAY
4.TOP ELEMENT
5.EXIT
Enter the choice:1

Enter the element:5

1.PUSH
2.POP
3.DISPLAY
4.TOP ELEMENT
5.EXIT
Enter the choice:1
```

```
Enter the element:9

1.PUSH
2.POP
3.DISPLAY
4.TOP ELEMENT
5.EXIT
Enter the choice:3

The elements are:9       5
1.PUSH
2.POP
3.DISPLAY
4.TOP ELEMENT
5.EXIT
Enter the choice:2
```

```
Element deleted
1.PUSH
2.POP
3.DISPLAY
4.TOP ELEMENT
5.EXIT
Enter the choice:4

The topmost element is: 5
1.PUSH
2.POP
3.DISPLAY
4.TOP ELEMENT
5.EXIT
```

3. IMPLEMENT QUEUE AS AN ARRAY

AIM : PROGRAM TO IMPLEMENT QUEUE AS AN ARRAY

ALGORITHM :

STEP 1: START

STEP 2: Declare all the functions used in queue implementation.

STEP 3: Initialize an array, queue_array[MAX], rear=-1,front=-1

STEP 4: Using switch case, enter your choice

STEP 5: In **insert** operation, if (rear == MAX - 1) print "Queue Overflow"

STEP 6: Else read add_item and store the item in queue_array[rear]

STEP 7:for **delete()** operation, if (front == - 1 || front > rear) print "Queue Underflow" , else, print the deleted item.

STEP 8: display() the array elements by printing queue_array[i]);

STEP 9: STOP

SOURCE CODE :

```
#include <stdio.h>
#define MAX 50
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
```

```
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
      printf("2.Delete element from queue \n");
      printf("3.Display all elements of queue \n");
      printf("4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(1);
         default:
         printf("Wrong choice \n");
```

```
void insert()
{
   int add_item;
   if (rear == MAX - 1)
   printf("Queue Overflow \n");
   else
   {
      if (front == - 1)
      front = 0;
      printf("Inset the element in queue : ");
      scanf("%d", &add_item);
      rear = rear + 1;
      queue_array[rear] = add_item;
   }
}
void delete()
{
   if (front == - 1 || front > rear)
   {
      printf("Queue Underflow \n");
      return ;
   }
   else
   {
      printf("Element deleted from queue is : %d\n", queue_array[front]);
      front = front + 1;
```

```
    }
}
void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

RESULT : The required output is obtained.

OUTPUT

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 4
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 6
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
```

```
Enter your choice : 3
Queue is :
4 6
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2
Element deleted from queue is : 4
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
```

4. IMPLEMENT CIRCULAR QUEUE OPERATIONS-ADD, DELETE, SEARCH

AIM : Program to implement circular queue operations

ALGORITHM :

STEP 1: START

STEP 2: Input the number of elements

STEP 3: Input your choice

STEP 4: For **insert** operation, if(front==0 && rear==(maxsize-1)), print "Overflow"

STEP 5: Otherwise, rear=(rear+1) % maxsize, q[rear]=item.

STEP 6: if(front==-1), front=0 and print that element is inserted.

STEP 7: For **search** operation, input the element which you want to search and check if(item==q[i])

STEP 8: If the condition is true, print that the location in which the element was found at.

STEP 9: Else if(c==0) print that element couldn't be found. Otherwise print the number of times the element was found.

STEP 10: For **delete** operation, Print underflow if(front==-1), else, item=q[front], if(front==rear), then front==-1 and rear==-1,else front=(front+1) % maxsize.

STEP 11: Print the deleted item.

STEP 12: For **displaying**,if(rear==-1) print that the queue is empty,else, print the elements using for loop, that is, print q[i].

STEP 13: If the queue is not empty, print the front element - item=q[front] and print item

STEP 14: If the queue is not empty, print the rear element - item=q[rear] and print item

STEP 15: STOP

SOURCE CODE :

```
#include<stdio.h>
#define maxsize 25
void cqinsert();
void cqsearch();
void cqdelete();
void cqdisplay();
void cqfele();
void cqrele();
int front=-1,rear=-1;
int q[maxsize];
void main()
{
  int ch,n;
  printf("Enter the no of elements:");
  scanf("%d",&n);
  printf("\nOperations on circular queue:");
  while(ch!=7)
  {
   printf("\n1.INSERT\n2.SEARCH\n3.DELETE\n4.DISPLAY \n5.FRONT
ELEMENT\n6.REAR ELEMENT\n7.Exit");
   printf("\nEnter the choice:");
   scanf("%d",&ch);
   switch(ch)
    {
    case 1:{
```

```
            cqinsert();
            break;
            }
        case 2:{
            cqsearch();
            break;
            }
case 3:{
            cqdelete();
            break;
            }
case 4:{
            cqdisplay();
            break;
            }
case 5:{
            cqfele();
            break;
            }
case 6:{
            cqrele();
            break;
            }
case 7:{
           printf("\nExiting");
           break;
```

```
        }
default:printf("\nInvalid choice");
}
};
getch();
}
void cqinsert()
{
 int item;
 printf("\nEnter the element:");
 scanf("%d",&item);
 if(front==0 && rear==(maxsize-1))
  {
  printf("\nOverflow");
  return;
  }
  else
  {
  rear=(rear+1) % maxsize;
  q[rear]=item;
  }
  if(front==-1)
   front=0;
 printf("\nElement inserted");
 }
 void cqsearch()
```

```
{
   int item,i,c=0;
   printf("\n Enter the element you want to search: ");
   scanf("%d",&item);
   for(i=front;i<=rear;i++)
   {
      if(item==q[i])
       {
         printf("\nThe element was found at:%d ",i+1);
         c++;
       }
   }
      if(c==0)
      {
         printf("\nThe element does not exist in the queue. ");
      }
      else
      {
         printf("\nThe item is present %d times",c);
      }
}
void cqdelete()
{
 int item;
 if(front==-1)
 {
```

```
printf("\nUnderflow");
return;
}
else
{
item=q[front];
if(front==rear)
{
front=-1;
rear=-1;
}
else
 front=(front+1) % maxsize;
 }
printf("Element deleted is %d",item);
}
void cqdisplay()
{
 int i;
 if(rear==-1)
     printf("\nEmpty queue");
 else
     {
      printf("\nThe elements are:");
      for(i=front;i<=rear;i++)
      {
```

```
  printf("%d\t",q[i]);
  }
 }}
void cqfele()
{
int item;
if(front==-1)
 printf("\nEmpty queue");
else
{
item=q[front];
printf("\nThe front element is %d",item);
}}
void cqrele()
{
int item;
if(rear==-1)
printf("\nEmpty queue");
else
{
item=q[rear];
printf("The rear element is %d",item);
}}
```

RESULT : The required output is obtained.

OUTPUT

```
Enter the no of elements:3

Operations on circular queue:
1.INSERT
2.SEARCH
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:1

Enter the element:6

Element inserted
1.INSERT
2.SEARCH
3.DELETE
```

```
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:1

Enter the element:9

Element inserted
1.INSERT
2.SEARCH
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:1
```

```
Enter the element:3

Element inserted
1.INSERT
2.SEARCH
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:2

 Enter the element you want to search: 3

The element was found at:3
The item is present 1 times
1.INSERT
2.SEARCH
3.DELETE
```

```
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:4

The elements are:6      9       3
1.INSERT
2.SEARCH
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:3
Element deleted is 6
1.INSERT
2.SEARCH
```

```
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:5

The front element is 9
1.INSERT
2.SEARCH
3.DELETE
4.DISPLAY
5.FRONT ELEMENT
6.REAR ELEMENT
7.Exit
Enter the choice:6
The rear element is 3
1.INSERT
2.SEARCH
```

5. IMPLEMENT SINGLY LINKED LIST OPERATIONS AS A STACK-PUSH, POP, LINEAR SEARCH.

AIM: Program to implement singly linked list operations as a stack.

ALGORITHM:

STEP 1: START

STEP 2: Create struct node and initialize val, pointers next and head

STEP 3: Choose the option which you want to perform

STEP 4: For **push** operation, initialize a pointer ptr and the if(ptr == NULL) , print "Not able to push the element"

STEP 5: Else if(head==NULL) , ptr->val = val, ptr -> next = NULL, head=ptr

STEP 6: Else, ptr->val = val, ptr->next = head, head=ptr;

STEP 7: For **pop** operation, store head in ptr and point head to next, and then remove ptr.

STEP 8: For **search** operation, read the item which you want to search, then with the help of a while loop, traverse through the list, ptr = ptr->next; and print when the item is found.

STEP 9: **Display** the elements in the stack by traversing through the list.

STEP 10: STOP

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void search();
```

```
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;
void main ()
{
   int choice=0;
   while(choice != 5)
   {
      printf("\n\nChose one from the below options\n");
      printf("\n1.Push\n2.Pop\n3.Search\n4.Show\n5.Exit");
      printf("\n Enter your choice \n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
         case 2:
         {
            pop();
```

```
            break;
        }
        case 3:
        {
            search();
            break;
        }
        case 4:
        {
            display();
            break;
        }
        case 5:
        {
            printf("Exiting");
            break;
        }
        default:
        {
            printf("Please Enter valid choice ");
        }
    };
}
}
void push ()
{
```

```
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("Not able to push the element");
    }
    else
    {
        printf("Enter the value: ");
        scanf("%d",&val);
        if(head==NULL)
        {
            ptr->val = val;
            ptr -> next = NULL;
            head=ptr;
        }
        else
        {
            ptr->val = val;
            ptr->next = head;
            head=ptr;
        }
        printf("Item pushed.");
    }
}
void pop()
```

```
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow");
    }
    else
    {
        item = head->val;
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped.");
    }
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
```

```
    {
        printf("\nEnter item which you want to search: \n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->val == item)
            {
                printf("Item found at location: %d ",i+1);
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==0)
        {
            printf("Item not found.\n");
        }
    }
}
void display()
{
    int i;
    struct node *ptr;
    ptr=head;
    if(ptr == NULL)
    {
```

```
        printf("Stack is empty\n");
    }
    else
    {
        printf("Printing Stack elements: \n");
        while(ptr!=NULL)
        {
            printf("%d\n",ptr->val);
            ptr = ptr->next;
        }
    }
}
```

RESULT: The required output is obtained.

OUTPUT

```
Chose one from the below options

1.Push
2.Pop
3.Search
4.Show
5.Exit
 Enter your choice
1
Enter the value: 4
Item pushed.

Chose one from the below options

1.Push
2.Pop
3.Search
4.Show
5.Exit
```

```
5.Exit
 Enter your choice
1
Enter the value: 7
Item pushed.

Chose one from the below options

1.Push
2.Pop
3.Search
4.Show
5.Exit
 Enter your choice
3

Enter item which you want to search:
4
Item found at location: 2
```

```
Chose one from the below options

1.Push
2.Pop
3.Search
4.Show
5.Exit
 Enter your choice
2
Item popped.

Chose one from the below options

1.Push
2.Pop
3.Search
4.Show
5.Exit
 Enter your choice
```

```

1.Push
2.Pop
3.Search
4.Show
5.Exit
 Enter your choice
4
Printing Stack elements:
4
```

6. DOUBLY LINKED LIST-CREATE, INSERT, DELETE

AIM : Program to create a doubly linked list and perform the operations**.**

ALGORITHM :

STEP 1: START

STEP 2: Create DNode and initialize the pointers left,right,first=NULL, temp, current.

STEP 3: For **insertion** in the **first** place, if first is null, temp->right=NULL, (*first)=temp, else, temp->right=(*first);

(*first)->left=temp;

STEP 4: For **insertion** at the **last** place, traverse through the list with the help of the current pointer. Traversing will stop when the current's right becomes NULL and the point it to temp's left and now temp's right will be NULL

STEP 5: For **insertion at any place,** read the position in which you want to enter the element and traverse to that location and insert temp at that location.

STEP 6: For **deletion** at the **first** place, current=first, then first=first->link,

first->left=NULL and free(current)

STEP 7: For **deletion** at the **last** place, traverse through the list till current's right becomes null and free(current) when the last position is reaches

STEP 8: For **deletion at any place**, enter the position of the node which you want to delete and then traverse through the list and once the node is found, remove the connections of that node from other nodes and free(current)

STEP 9: Display the elements of the node by traversing through the list, first=first->right

STEP 10: For performing **search** operation, traverse through the list, ptr=ptr->right and when the element is found, print it.

STEP 11: STOP.

SOURCE CODE :

```
#include<stdio.h>
#include<stdlib.h>
struct DNode
{
   struct DNode *left;
   int data;
   struct DNode *right;
}*first=NULL,*temp,*current;
int n,pos,item,ch,i;
typedef struct DNode DNode;
void insertDlistf();
void insertDlistl();
void insertDlistatany();
void deleteDlistf();
void deleteDlistl();
void deleteDlistatany();
void traverseDlist();
void searchList();
void main()
{int r,p;
   printf("Enter the number of nodes:\n");
   scanf("%d",&n);
   while(n!=8)
   {
```

```
printf("\n1.Insertf\n2.Insertl\n3.Insertan\n4.Deletef\n5.Deletel\n6.Deletean\n
7.Traverse\n8.Search\n9.Exit");
      printf("\nEnter your choice:\n");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:{
         printf("Enter the item to be inserted: ");
         scanf("%d",&r);
         insertDlistf(&first,r);
         break;
      }
      case 2:
      {printf("Enter item:\n");
      scanf("%d",&r);
      insertDlistl(&first,r);
         break;
      }
      case 3:
      {
         printf("Enter the item:\n");
         scanf("%d",&r);
         printf("Enter the position:\n");
         scanf("%d",&p);
         insertDlistatany(&first,p,r);
         break;
```

```
}
case 4:
{
   deleteDlistf(&first);
   break;
}case 5:
{
   deleteDlistl(&first);
   break;
}
case 6:
{
   printf("Enter the position to be deleted:\n");
   scanf("%d",&p);
   deleteDlistatany(&first,p);
   break;
}

case 7:
{
   traverseDlist(first);
   break;
}
case 8:
{
   searchList(first);
```

```
            break;
        }
        case 9:
        {
            printf("Exit\n");
            break;
        }
       default:
        {
           printf("Enter a valid choice\n");
            break;
        }
    }}
}

void insertDlistf( DNode **first,int item)
{
temp=(DNode*)malloc(sizeof(DNode));
temp->data=item;
temp->left=NULL;
if(*first==NULL)
temp->right=NULL;
else
{
    temp->right=(*first);
    (*first)->left=temp;
```

```
}
(*first)=temp;
}
void insertDlistl(DNode **first,int pos,int item)
{
   temp=(DNode*)malloc(sizeof(DNode));
   temp->data=item;
   temp->right=NULL;
   if(*first==NULL)
   {
      temp->left=NULL;
      (*first)=temp;
   }
   else
   {
      current=(*first);
      while(current->right!=NULL)
      current=current->right;
      temp->left=current;
      current->right=temp;
   }
}
void insertDlistatany(DNode **first,int pos,int item)
{
   temp=(DNode*)malloc(sizeof(DNode));
temp->data=item;
```

```
if(pos==1)
{
   if(*first!=NULL)
   (*first)->left=temp;
   temp->right=(*first);
   temp->left=NULL;
   (*first)=temp;
}
else
{
   i=2;
   current=(*first);
   while((i<pos)&&(current->right!=NULL))
   {
      i++;
      current=current->right;}
      temp->left=current;
      temp->right=current->right;
      if(current->right!=NULL)
      temp->right->left=temp;
      current->right=temp;
   }
}
void deleteDlistf(DNode **first)
{
   if(*first==NULL)
```

```
    {
        printf("\nList is empty\n");
    }
    current=(*first);
    (*first)=(*first)->right;
    if(*first!=NULL)
    (*first)->left=NULL;
    item=current->data;
    free(current);
    printf("\nDeleted Item=%d",item);
}
void deleteDlistl(DNode **first)
{
    if(*first==NULL)
    {
        printf("List is empty\n");
        return;
    }
    current=*first;
    while(current->right!=NULL)
    current=current->right;
    if(current->left!=NULL)
    current->left->right=current->right;
    else
    (*first)=NULL;
    item=current->data;
```

```
    free(current);
    printf("Deleted item= %d",item);
}
void deleteDlistatany(DNode **first,int pos)
{i=1;
    if(*first==NULL)
    {
        printf("List is empty\n");
        return;
    }
    current=(*first);
    while(current!=NULL)
    {
        if(i==pos)
        {
        item=current->data;
        if(current->left==NULL)
        {
            current->right->left=NULL;
            (*first)=current->right;
            free(current);
        }
        else if(current->right==NULL)
        {
            current->left->right=current->right;
            free(current);
```

```
        }
        else
        {
            current->left->right=current->right;
            current->right->left=current->left;
            free(current);
        }
        printf("Deleted item=%d",item);
        return;
    }
    i++;
    current=current->right;}
printf("Such a node doesn't exist");
}
void traverseDlist(DNode *first)
{
    if(first==NULL)
    printf("\nList is empty\n");
    else
    {
        while(first!=NULL)
        {
            printf("%d ",first->data);
            first=first->right;
        }
        printf(" Last\n");
```

```
    }
  }
void searchList(DNode *first)
{
  struct DNode *ptr;
  int item,i=0,flag;
  ptr = first;
  if(ptr == NULL)
  {
    printf("\nEmpty List\n");
  }
  else
  {
    printf("\nEnter item which you want to search: \n");
    scanf("%d",&item);
    while (ptr!=NULL)
    {
      if(ptr->data == item)
      {
        printf("\nItem found at location: %d ",i+1);
        flag=1;
        break;
      }
       i++;
      ptr = ptr -> right;
    }
```

```
        if(flag==0)
        {
            printf("\nItem not found\n");
        }
    }
}
```

RESULT: The required output is obtained.

OUTPUT

```
Enter the number of nodes:
5

1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
1
Enter the item to be inserted: 8

1.Insertf
2.Insertl
3.Insertan
```

```
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
1
Enter the item to be inserted: 9

1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
```

```
8.Search
9.Exit
Enter your choice:
2
Enter item:
1

1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
3
```

```
Enter your choice:
3
Enter the item:
3
Enter the position:
2

1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
4
```

```
Deleted Item=9
1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
5
Deleted item= 1
1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
```

```
7.Traverse
8.Search
9.Exit
Enter your choice:
6
Enter the position to be deleted:
2
Deleted item=8
1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
1
```

```
Enter the item to be inserted: 5

1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
8

Enter item which you want to search:
5

Item found at location: 1
1.Insertf
```

```
1.Insertf
2.Insertl
3.Insertan
4.Deletef
5.Deletel
6.Deletean
7.Traverse
8.Search
9.Exit
Enter your choice:
7
5 3  Last
```

7. BINARY SEARCH TREE-INSERTION, DELETION, SEARCH

AIM : Program to perform binary search tree operations.

ALGORITHM :

Step 1: Include all the header files which are used in the program.

Step 2: Declare all the functions used in tree implementation.

Step 3: Create a structure tnode with three member variables (data,*lchild,*rchild)

Step 4: Inside the main function define the functions to implement the tree operations.

Step 5: Perform inorder,postorder,preorder traversals by calling their corresponding functions(Inorder(),Postorder(),Preorder()).

1.**Tinsert()** will insert the new value into a binary search tree:

It checks whether root is null, which means tree is empty. New node will become root node of tree.

If tree is not empty, it will compare value of new node with root node. If value of new node is greater than root, new node will be inserted to right subtree. Else, it will be inserted in left subtree.

2.**Tdelete()** will delete a particular node from the tree:

If value of node to be deleted is less than root node, search node in left subtree. Else, search in right subtree.

If node is found and it has no children, then set the node to null.

If node has one child then, child node will take position of node.

If node has two children then, find a minimum value node from its right subtree. This minimum value node will replace the current node.

3.**Tsearch()** will search a particular node from the tree.

SOURCE CODE :

```
#include<stdio.h>
struct tnode
{
 struct tnode *lchild;
 int data;
 struct tnode *rchild;
};
typedef struct tnode tnode;
tnode *getnode();
main()
 {
 int a[100],i,n,item;
 tnode *root;

 root=NULL;
 printf("Enter the no of elements:");
 scanf("%d",&n);
 printf("\nEnter the elements:");
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 for(i=0;i<n;i++)
```

```
 Tinsert(&root,a[i]);
printf("\nBinary tree:\n");
Tdisplay(root,1);
printf("\nInorder Traversal\n");
Inorder(root);
printf("\nPreorder Traversal\n");
Preorder(root);
printf("\nPostorder Traversal\n");
Postorder(root);
printf("\nEnter the element to delete:");
scanf("%d",&item);
Tdelete(&root,item);
printf("\nBinary tree after deletion\n");
Tdisplay(root,1);
printf("\nEnter the element to search:");
scanf("%d",&item);
Tsearch(item);
getch();
}
Tinsert(tnode **rt,int item)
{
 tnode *current=(*rt),*temp;
 if((*rt)==NULL)
 {
  (*rt)=getnode();
  (*rt)->data=item;
```

```
 (*rt)->lchild=NULL;
 (*rt)->rchild=NULL;
 return;
}
while(current!=NULL)
 {
 if(item<current->data)
 {
 if(current->lchild!=NULL)
   current=current->lchild;
 else
  {
  temp=getnode();
  current->lchild=temp;
  temp->data=item;
  temp->rchild=NULL;
  temp->lchild=NULL;
  return;
  }
  }
 else
 {
  if(item>current->data)
  {
   if(current->rchild!=NULL)
     current=current->rchild;
```

```
    else
    {
     temp=getnode();
     current->rchild=temp;
     temp->data=item;
     temp->rchild=NULL;
     temp->lchild=NULL;
     return;
     }
    }
    else
    {
    printf("\nWrong data");
    exit(0);
    }
    }
    }
    }
Inorder(tnode *rt)
{
 if(rt!=NULL)
 {
 Inorder(rt->lchild);
 printf("\t%d\t",rt->data);
 Inorder(rt->rchild);
 }
```

```
 else
  return;
}
Preorder(tnode *rt)
{
 if(rt!=NULL)
 {
 printf("\t%d\t",rt->data);
 Preorder(rt->lchild);
 Preorder(rt->rchild);
 }
 else
  return;
}
Postorder(tnode *rt)
{
 if(rt!=NULL)
 {
  Postorder(rt->lchild);
  Postorder(rt->rchild);
  printf("\t%d\t",rt->data);
 }
 else
  return;
}
Tdisplay(tnode *rt,int level)
```

```
{
 int i;
 if((rt)!=NULL)
 {
  Tdisplay((rt)->rchild,level+1);
  printf("\n");
  for(i=0;i<level;i++)
   printf("  ");
  printf("%d",(rt)->data);
  Tdisplay((rt)->lchild,level+1);
  }
}
Tdelete(tnode **rt,int item)
{
 tnode *current;
 if(*rt==NULL)
  {
  printf("\nError");
  getch();
  return;
  }
 if(item<(*rt)->data)
   Tdelete(&((*rt)->lchild),item);
 else
 {
  if(item>(*rt)->data)
```

```
  Tdelete(&((*rt)->rchild),item);
else
 {
 current=(*rt);
 if(current->rchild==NULL)
 {
 (*rt)=(*rt)->lchild;
 free(current);
 }
else
{
if(current->lchild==NULL)
{
(*rt)=(*rt)->rchild;
free(current);
}
else
{
current=(*rt)->rchild;
 while(current->lchild!=NULL)
  current=current->lchild;
 current->lchild=(*rt)->lchild;
 current=(*rt);
 (*rt)=(*rt)->rchild;
 free(current);
 }
```

```
   }
  }
  }
}
Tsearch(int item)
{
 tnode *temp,*root;
if(root==NULL)
 {
  printf("\nTree is empty");
  return;
 }
 if(item==root->data)
  {
  printf("\nElement found at root");
  return;
  }
 else if(item<root->data)
 {
  printf("\nElement found at left sub tree");
  return;
  }
 else
 {
  printf("\nElement found at right sub tree");
  return;
```

```
  }
}
tnode *getnode()
{
 tnode *p;
 p=(tnode *)malloc(sizeof(tnode));
 return(p);
}
freenode(tnode *p)
 {
 free(p);
 }
```

RESULT : The required output is obtained.

OUTPUT

```
Enter the no of elements:5

Enter the elements:50
25
60
30
77

Binary tree:

   77
  60
 50
   30
  25
Inorder Traversal
        25              30              50              60              77
Preorder Traversal
        50              25              30              60              77
```

```
Postorder Traversal
        30              25              77              60              50
Enter the element to delete:30

Binary tree after deletion

   77
  60
 50
  25
Enter the element to search:25
```

```
Enter the element to search:25

Element found at left sub tree_
```

8. SET DATA STRUCTURE AND SET OPERATIONS.- UNION, INTERSECTION, DIFFERENCE.

AIM : Program to perform set data structure and set operations.

ALGORITHM :

STEP 1: Start

STEP 2: Enter your choice

STEP 3: Enter the size of the first set and it's elements and the size of the second set and it's elements.

STEP 4: For union operation, if(a[i]!=b[i]), then, c[i]=1, otherwise, c[i]=a[i];

STEP 5: Print c[i] and call output(c)

STEP 6: For intersection operation, Check if (a[i]==b[i]), then c[i]=a[i], otherwise c[i]=0.

STEP 7: Print c[i] and perform output(c)

STEP 8: For difference operation, check if (a[i]==1 && b[i]==0), c[i]=1, otherwise, c[i]=0.

STEP 9: Print c[i] and perform output (c)

STEP 8: STOP

SOURCE CODE :

```
//Universal Set is {1,2,3,4,5,6,7,8,9}
#include <stdio.h>
void input();
void output();
void setunion();
void intersection();
```

```
void difference();
int a[]={0,0,0,0,0,0,0,0,0,0},b[]={0,0,0,0,0,0,0,0,0,0};
int c[]={0,0,0,0,0,0,0,0,0,0};
int main()
 {
  int ch,wish;
  do
   {
printf("\n___MENU_\n");
printf("1.INPUT\n2.UNION\n3.INTERSECTION\n4.DIFFERENCE\n");
printf("ENTER CHOICE : ");
scanf("%d",&ch);
  switch(ch)
   {
     case 1:input();
            break;
     case 2:setunion();
            break;
     case 3:intersection();
            break;
     case 4:difference();
            break;
   }
printf("\nDO YOU WANT TO CONTINUE?(1/0) : ");
scanf("%d",&wish);
}while(wish==1);
```

```
 }
void input()
{
  int n,x,i;
printf("\nENTER SIZE OF FIRST SET : ");
scanf("%d",&n);
printf("\nENTER ELEMENTS : \t");
  for(i=1;i<=n;i++)
  {
scanf("%d",&x);
      a[x]=1;
  }
printf("\nENTER SIZE OF SECOND SET : ");
scanf("%d",&n);
printf("\nENTER ELEMENTS : \t");
  for(i=1;i<=n;i++)
  {
scanf("%d",&x);
      b[x]=1;
  }
printf("\nFIRST SET : \t");
  for(i=1;i<=9;i++)
   {
printf("%d",a[i]);
   }
printf("\nSECOND SET : \t");
```

```
  for(i=1;i<=9;i++)
  {
printf("%d",b[i]);
  }
}
void output(int c[])
{
   int i;
printf("\nSET IS : \t");
   for(i=1;i<=9;i++)
   {
      if (c[i]!=0)
printf("%d\t",i);
   }
}
void setunion()
{
 int i,c[10];
 for(i=1;i<=9;i++)
   {
   if(a[i]!=b[i])
       c[i]=1;
   else
     c[i]=a[i];
   }
 for(i=1;i<=9;i++)
```

```
  {
printf("%d",c[i]);
  }
  output(c);
}
void intersection()
{
 int i,c[10];
 for(i=1;i<=9;i++)
  {
  if (a[i]==b[i])
        c[i]=a[i];
  else
        c[i]=0;
  }
 for(i=1;i<=9;i++)
  {
printf("%d",c[i]);
  }
  output(c);
}
void difference()
{
 int i,c[10];
 for(i=1;i<=9;i++)
  {
```

```
   if (a[i]==1 && b[i]==0)
          c[i]=1;
    else
      c[i]=0;
    }
  for(i=1;i<=9;i++)
   {
printf("%d",c[i]);
   }
   output(c);
}
```

RESULT : The required output is obtained.

OUTPUT

```
___MENU_
1.INPUT
2.UNION
3.INTERSECTION
4.DIFFERENCE
ENTER CHOICE : 1

ENTER SIZE OF FIRST SET : 3

ENTER ELEMENTS :         7
1
3

ENTER SIZE OF SECOND SET : 3

ENTER ELEMENTS :         5
1
3
```

```
FIRST SET :     101000100
SECOND SET :    101010000
DO YOU WANT TO CONTINUE?(1/0) : 1

___MENU_
1.INPUT
2.UNION
3.INTERSECTION
4.DIFFERENCE
ENTER CHOICE : 2
101010100
SET IS :        1       3       5       7
DO YOU WANT TO CONTINUE?(1/0) : 1

___MENU_
1.INPUT
2.UNION
3.INTERSECTION
4.DIFFERENCE
```

```
1.INPUT
2.UNION
3.INTERSECTION
4.DIFFERENCE
ENTER CHOICE : 3
101000000
SET IS :         1       3
DO YOU WANT TO CONTINUE?(1/0) : 1

___MENU_
1.INPUT
2.UNION
3.INTERSECTION
4.DIFFERENCE
ENTER CHOICE : 4
000000100
SET IS :         7
```

9. DISJOINT SET- CREATE, UNION, FIND.

AIM : PROGRAM TO CREATE DISJOINT SET AND PERFORM THE OPERATIONS.

ALGORITHM:

Step 1: START and Include all the header files which are used in the program.

Step 2: Create a structure DisjSet with three member functions (parent,rank,n) to store parent array,rank array,number of sets.Create a variable for the structure 'dis'.

Step 3: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on disjoint set.

makeSet()

Step 1: Declare 'i' and 'x'.

Step 2: Initialise i with '0' and incrementing by '1'(i++) until i<dis.n and read x.

Step 3: Assign dis.parent[i]=i and dis.rank[i]=0.

displaySet()

Step 1: Declare 'i'

Step 2: Iterate two loops for printing parent array and rank array initialise i with '0' and incrementing by '1'(i++) until i<dis.n.

Step 3: Print two arrays(dis.parent[i] and dis.rank[i]).

find(intx)

Step 1: Check the 'x' is not the parent of itself it means that find the representative(dis.parent[x]!=x)

Step 2: If it is true then dis.parent[x]=find(dis.parent[x]), so we recursively call find() on its parent and move I's node directly under the representative of this set.

Step 3: Return dis.parent[x].

union(int x,int y)

Step 1: Find current sets of x and y(int xset=find(x) and int yset=find(y))

Step 2: Check if it is in the same set,if it is true then exit

Step 3: Put smaller ranked item under bigger ranked item if ranks are different and vice versa.

Step 4: If ranks are same, then increment rank.
STEP 17: STOP

SOURCE CODE :

```
#include <stdio.h>
struct DisjSeto
{
   int parent[10];
   int rank[10]; //rank[i] is the height of the tree representing the set
   int n;
}dis;
   // Creates n single item sets
  void makeSet()
   {
         int i,x;
         for ( i = 0; i < dis.n; i++) {
        scanf("%d",&x);
```

```
        dis.parent[i] = i;
        dis.rank[i]=0;
    }
}
//Displays Disjoint set
void displaySet()
{
 int i;
 printf("\nParent Array\n");

    for ( i = 0; i < dis.n; i++) {
        printf("%d ",dis.parent[i]); }
        printf("\nRank Array\n");
         for (i = 0; i < dis.n; i++)
         {
             printf("%d ",dis.rank[i]);
         }
        printf("\n");
}
// Finds set of given item x
int find(int x)
{
    // Finds the representative of the set
    // that x is an element of
    if (dis.parent[x] != x) {
```

```
        // if x is not the parent of itself


        dis.parent[x] = find(dis.parent[x]);
    }
    return dis.parent[x];
}
void Union(int x, int y)
{
    // Find current sets of x and y
    int xset = find(x);
    int yset = find(y);
     if (xset == yset)
        return;
    if (dis.rank[xset] < dis.rank[yset]) {
        dis.parent[xset] = yset;
        dis.rank[xset]=-1;
    }
    else if (dis.rank[xset] > dis.rank[yset]) {
        dis.parent[yset] = xset;
        dis.rank[yset]=-1;
    }
    else {
        dis.parent[yset] = xset;
        dis.rank[xset] = dis.rank[xset] + 1;
        dis.rank[yset]=-1;
    }
```

```
    }

int main()
{
    int n,x,y,ch,wish;

    printf("Enter the number of elements: ");
    scanf("%d",&dis.n);
    printf("Enter the elements :");
    makeSet();
 do
 {
 printf("\n___MENU_\n");
 printf("1. Union \n2.Find\n3.Display\n");
 printf("\nEnter choice\n");
 scanf("%d",&ch);

 switch(ch)
  {
    case 1: printf("Enter elements to perform union");
                    scanf("%d %d",&x,&y);
                    Union(x, y);
                   break;
    case 2: printf("Enter elements to check if connected components");
                    scanf("%d %d",&x,&y);
                    if (find(x) == find(y))
```

```
                printf("Connected components\n") ;
                else
                    printf("Not connected components \n") ;
            break;
     case 3: displaySet();
            break;
   }
   printf("\nDo you wish to continue ?(1/0)\n");
   scanf("%d",&wish);
   }while(wish==1);
 return 0;
}
```

RESULT : The required output is obtained.

OUTPUT

```
Enter the number of elements: 6
Enter the elements :1 2 3 4 5 6

___MENU_
1. Union
2.Find
3.Display

Enter choice
1
Enter elements to perform union2 3

Do you wish to continue ?(1/0)
1

___MENU_
1. Union
2.Find
3.Display
```

```
2
Enter elements to check if connected components2 3
Connected components

Do you wish to continue ?(1/0)
1

___MENU_
1. Union
2.Find
3.Display

Enter choice
3

Parent Array
0 1 2 2 4 5
Rank Array
0 0 1 -1 0 0
```

10. RED BLACK TREE OPERATIONS

AIM : Program to perform red black tree operations.

ALGORITHM :

Step 1:START and Include all the header files which are used in the program.
Step 2: Create a structure rbtNode with essential member variables(data,color,*left,*right,*parent).Create a variable to access structure *root=NULL.
Step 3: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on red black tree implementation.

Insert()

Step 1: Check whether tree is Empty.
Step 2: If tree is Empty then insert the newNode as Root node with color Black and exit from the operation.
Step 3: If tree is not Empty then insert the newNode as a leaf node with Red color.
Step 4: If the parent of newNode is Black then exit from the operation.
Step 5: If the parent of newNode is Red then check the color of parent node's sibling of newNode.
Step 6: If it is Black or NULL node then make a suitable Rotation and Recolor it.
Step 7: If it is Red colored node then perform Recolor and Recheck it. Repeat the same until tree becomes Red Black Tree.

leftRotate()

Step 1: If the parent of newly inserting node is red node then perform this function by calling.

rightRotate()
Step 1: If the parent of newly inserting node is black or null node then perform this function by calling.

inorder()
Step 1: To display the tree nodes as left,root,right

color()
Step 1: When adjacent red color appears at the time of inserting then recolor it using this function.
STEP 15: STOP

SOURCE CODE :

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

{
   int data;
   char color;
   struct rbtNode *left, *right, *parent;
};
struct rbtNode *root=NULL;
void leftRotate(struct rbtNode *x)
{
   struct rbtNode *y;
   y= x->right;
   x->right = y->left;


   if (y->left != NULL)
            y->left->parent = x;

   y->parent = x->parent;
   if (x->parent == NULL)// x is root
            root = y;

   else if((x->parent->left != NULL) && (x->data==x->parent->left->data))
            x->parent->left = y;
   else
            x->parent->right = y;
   y->left = x;
   x->parent = y;
   return;
}
void rightRotate(struct rbtNode *y)
```

```
{
   struct rbtNode *x   ;
   x= y->left;
   y->left = x->right;
   if (x->right != NULL)
          x->right->parent = y;
   x->parent =y->parent;
   if (y->parent == NULL)
          root= x;
   else if((y->parent->left!=NULL)&&(y->data==y->parent->left->data))
          y->parent->left = x;
   else
          y->parent->right = x;
   x->right = y;
   y->parent = x;
 return;
}
void color(struct rbtNode *z)
{
   struct rbtNode *y=NULL;
   while((z->parent!=NULL)&&(z->parent->color=='R'))
   {
   if((z->parent->parent->left!=NULL)&&(z->parent->data==z->parent->parent-
>left->data))
          {
             if(z->parent->parent->right!=NULL)
                    y=z->parent->parent->right;
             if((y!=NULL)&&(y->color=='R'))
             {
                    z->parent->color='B';
                    y->color='B';
                    z->parent->parent->color = 'R';
                    if(z->parent->parent!=NULL)
                       z = z->parent->parent;
             }
             else
             {
          if((z->parent->right!=NULL)&&(z->data==z->parent->right->data))
                    {
                       z=z->parent;
```

```
                    leftRotate(z);
                }
                z->parent->color='B';
                        z->parent->parent->color='R';
                rightRotate(z->parent->parent) ;
            }
        }
        else
        {
                if(z->parent->parent->left!=NULL)
                y=z->parent->parent->left;
            if((y!=NULL)&&(y->color=='R'))
            {
                z->parent->color='B';
                y->color='B';
                z->parent->parent->color='R';
                if(z->parent->parent!=NULL)
                    z=z->parent->parent;
            }
            else
            {
              if((z->parent->left!=NULL)&&(z->data==z->parent->left->data))
                {
                    z=z->parent;
                    rightRotate(z);
                }
                z->parent->color='B';
                z->parent->parent->color='R';
                leftRotate(z->parent->parent);
            }

        }
    }
    root->color='B';
}
void insert(int val)
{
    struct rbtNode *x,*y,*z;
    z=(struct rbtNode *)malloc(sizeof(struct rbtNode));
    z->data=val;
```

```
    z->left=NULL;
    z->right=NULL;
    z->color='R';

    x=root;

    if(root==NULL)
    {
            root=z;
            root->color='B';
            return;
    }

    while(x!=NULL)
    {
            y=x;
            if(z->data<x->data)
               x=x->left;
            else
               x=x->right;
    }
    z->parent=y;
    if(y==NULL)
               root=z;
    else if(z->data<y->data)
               y->left=z;
    else
               y->right=z;

    color(z);

}
void inorder(struct rbtNode *root)

{
    struct rbtNode *temp=root;
    if (temp != NULL)
    {
            inorder(temp->left);
```

```
            printf("%d-%c   ", temp->data,temp->color);

        inorder(temp->right);
    }
    return;
}
void displayTree(struct rbtNnode *root,int level)
{
  int i;
  struct rbtNode *temp=root;

  if(temp!=NULL)
  {
        displayTree(temp->right,level+1);
        printf("\n\n");
        for(i=0;i<level;i++)
                printf("    ");
        printf("%d%c",temp->data,temp->color);
        displayTree(temp->left,level+1);
  }
}
void main()
{
   int ch,val;
   clrscr();
   printf("*******RED BLACK TREE INSERTION PROGRAM********")  ;
   while(1)
   {
        printf("\n1.Insert\n2.Display\n3.Exit\nEnter choice   :   ");
        scanf("%d",&ch);
        switch(ch)
        {
          case 1 :
          printf("Enter element : ");
          scanf("%d",&val);
          insert(val);
          break;
          case 2:
          displayTree(root,1);
          printf("\n\nINORDER TRAVERSAL :     ");
```

```
            inorder(root);break;
            case 3:exit(0);
        }
    }

}
```

RESULT : The required output is obtained.

OUTPUT

```
*******RED BLACK TREE INSERTION PROGRAM********
1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 4

1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 10

1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 3
```

```
1.Insert
2.Display
3.Exit
Enter choice    :    1
Enter element : 9

1.Insert
2.Display
3.Exit
Enter choice    :    2


         10B

              9R

     4B

         3B
```

```
INORDER TRAVERSAL :     3-B   4-B   9-R   10-B
```

11. BFS TRAVERSAL PROGRAM

AIM : Program to perform bfs traversal.

ALGORITHM :

STEP 1: START

STEP 2: Read the number of vertices

STEP 3: Read the graph data in matrix form

STEP 4: Read the starting vertex

STEP 5: In bfs(), Check if(a[v][i] && !visited[i]) and then q[++r]=i

STEP 6: if(f<=r), perform visited[q[f]]=1, bfs(q[f++]);

STEP 7: Print

STEP 8: STOP

SOURCE CODE :

```
#include<stdio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
        for (i=1;i<=n;i++)
          if(a[v][i] && !visited[i])
           q[++r]=i;
        if(f<=r) {
                visited[q[f]]=1;
                bfs(q[f++]);
        }
```

```
}
void main() {
        int v;
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++) {
                q[i]=0;
                visited[i]=0;
        }
        printf("\n Enter graph data in matrix form:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
        printf("\n Enter the starting vertex:");
        scanf("%d",&v);
        bfs(v);
        printf("\n The node which are reachable are:\n");
        for (i=1;i<=n;i++)
          if(visited[i])
           printf("%d\t",i); else
           printf("\n Bfs is not possible");
        getch();
}
```

RESULT : The required output is obtained.

OUTPUT

```
 Enter the number of vertices:4

 Enter graph data in matrix form:
0 1 0 1
1 1 0 0
0 0 1 1
1 0 1 1

 Enter the starting vertex:1

 The node which are reachable are:
1       2       3       4
```

12. DFS TRAVERSAL PROGRAM

AIM : Program to perform dfs traversal.

ALGORITHM :

STEP 1: START

STEP 2: Read the adjacency matrix

STEP 3: Initialize reach[v]=1

STEP 4: In dfs(), if(a[v][i] && !reach[i]), then print v and i, dfs(i), and then print v.

STEP 5: Check if the graph is connected or not.

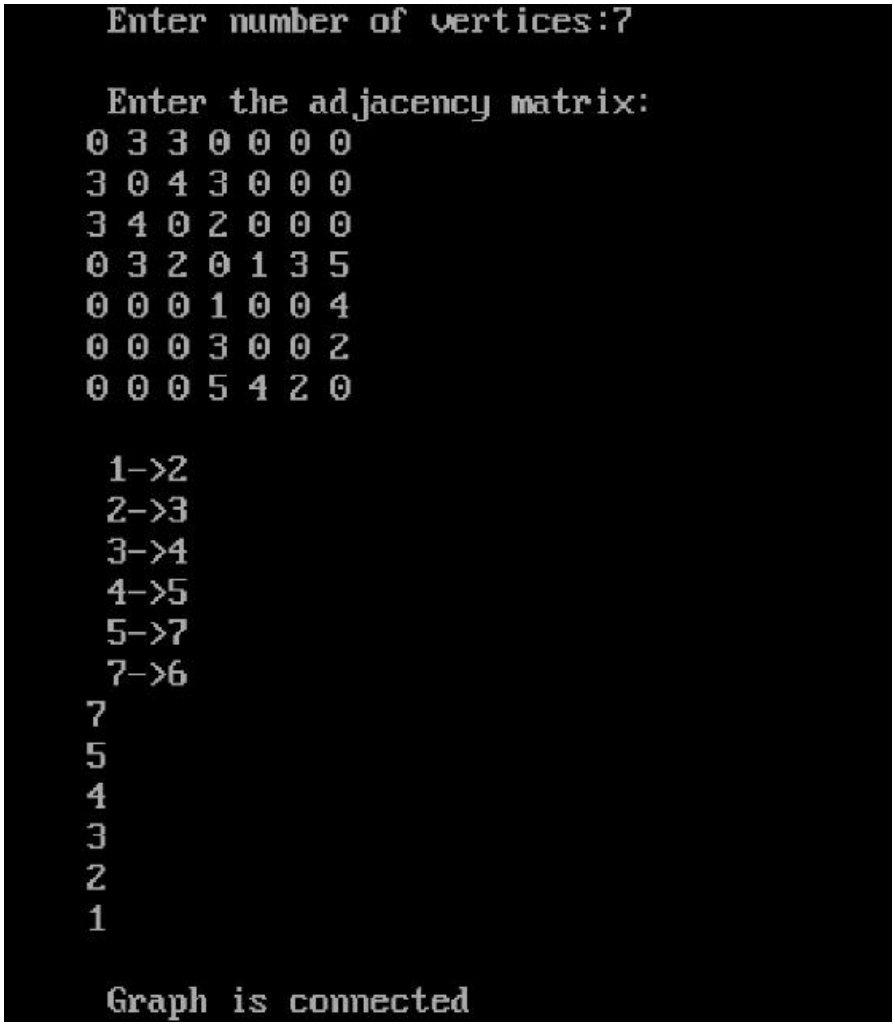
SOURCE CODE :

```
#include<stdio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
  int i;
  reach[v]=1;
  for (i=1;i<=n;i++)
  if(a[v][i] && !reach[i])
  {
     printf("\n %d->%d",v,i);
     dfs(i);
     printf("\n%d",v);
  } }
```

```
void main()
 {
   int i,j,count=0;
   printf("\n Enter number of vertices:");
   scanf("%d",&n);
   for (i=1;i<=n;i++)
   {
        reach[i]=0;
        for (j=1;j<=n;j++)
        a[i][j]=0; }
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        dfs(1);
        printf("\n");
        for (i=1;i<=n;i++)
         {
           if(reach[i])
           count++;}
           if(count==n)
           printf("\n Graph is connected");
           else
           printf("\n Graph is not connected");
           getch();}
```

RESULT : The desired output is obtained.

OUTPUT


Enter number of vertices:7
Enter the adjacency matrix:
0 3 3 0 0 0 0
3 0 4 3 0 0 0
3 4 0 2 0 0 0
0 3 2 0 1 3 5
0 0 0 1 0 0 4
0 0 0 3 0 0 2
0 0 0 5 4 2 0
1->2
2->3
3->4
4->5
5->7
7->6
7
5
4
3
2
1
Graph is connected

13. PRIM'S ALGORITHM PROGRAM.

AIM : Program to perform prim's algorithm.

ALGORITHM :

STEP 1: START

STEP 2: Enter the number of nodes

STEP 3: Enter the adjacency matrix

STEP 4: Initialize the cost[i][j] as 999

STEP 5: Assign min as 999

STEP 6: While the number of edges is less than the number of vertices, perform the following functions.

STEP 7: Using for loops check if(cost[i][j]< min), and if(visited[i]!=0) and if the conditions are true, a=u=i, b=v=j.

STEP 8: if(visited[u]==0 || visited[v]==0), print a, b, min and increment mincost.

STEP 9: Then cost[a][b]=cost[b][a]=999 and print the minimum cost

STEP 10: STOP

SOURCE CODE :

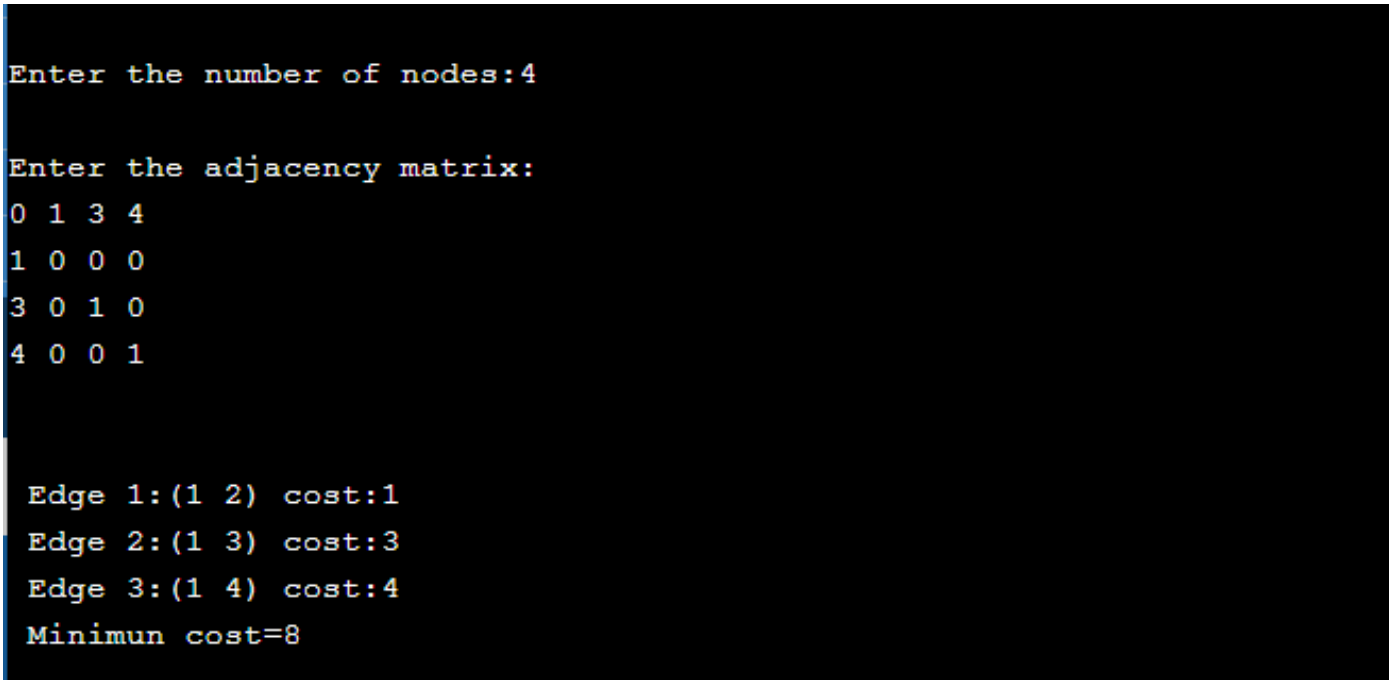
```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
        printf("\nEnter the number of nodes:");
        scanf("%d",&n);
```

```
printf("\nEnter the adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0)
                cost[i][j]=999;}
visited[1]=1;
printf("\n");
while(ne < n)
{
        for(i=1,min=999;i<=n;i++)
        for(j=1;j<=n;j++)
        if(cost[i][j]< min)
        if(visited[i]!=0)
        {
                min=cost[i][j];
                a=u=i;
                b=v=j;}
        if(visited[u]==0 || visited[v]==0)
        {
                printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;}
        cost[a][b]=cost[b][a]=999;}
printf("\n Minimum cost=%d",mincost);
```

```
    getch();
}
```

RESULT : The desired output is obtained.

OUTPUT


Enter the number of nodes:4
Enter the adjacency matrix:
0 1 3 4
1 0 0 0
3 0 1 0
4 0 0 1
Edge 1:(1 2) cost:1
Edge 2:(1 3) cost:3
Edge 3:(1 4) cost:4
Minimun cost=8

14. KRUSKAL'S ALGORITHM PROGRAM.

AIM : Program to perform kruskal's algorithm.

ALGORITHM :

STEP 1: START

STEP 2: Read the number of vertices

STEP 3: Enter the cost adjacency matrix

STEP 4: Assign cost[i][j]=999

STEP 5: while the number of edges less than the number of vertices, and min=999, check if the cost[i][j] is less than min and perform min=cost[i][j],

a=u=i, b=v=j.

STEP 6: u=find(u) , v=find(v), and if(uni(u,v)) print a,b,min and increment the minimum cost

STEP 7: cost[a][b]=cost[b][a]=999 and print the minimum cost.

STEP 8: STOP

SOURCE CODE :

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
```

```
{

        printf("\n\tImplementation of Kruskal's algorithm\n");
        printf("\nEnter the no. of vertices:");
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
        }
        printf("The edges of Minimum Cost Spanning Tree are\n");
        while(ne < n)
        {
          for(i=1,min=999;i<=n;i++)
                {
                  for(j=1;j <= n;j++)
                        {
                           if(cost[i][j] < min)
                                {
                                   min=cost[i][j];
                                   a=u=i;
```

```
                                b=v=j;
                            }
                        }
                    }
                    u=find(u);
                    v=find(v);
                    if(uni(u,v))
                    {
                      printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                      mincost +=min;
                    }
             cost[a][b]=cost[b][a]=999;
          }
    printf("\n\tMinimum cost = %d\n",mincost);
    getch();
}
int find(int i)
{
          while(parent[i])
          i=parent[i];
          return i;
}
int uni(int i,int j)
{
          if(i!=j)
          {
```

```
                parent[j]=i;
                return 1;
        }
        return 0;
}
```

RESULT : The desired output is obtained.

OUTPUT

```
        Implementation of Kruskal's algorithm

Enter the no. of vertices:4

Enter the cost adjacency matrix:
0 1 4 5
1 3 6 1
4 6 0 0
5 1 0 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) =1
2 edge (2,4) =1
3 edge (1,3) =4

        Minimum cost = 6
```

15. DIJKSTRA'S ALGORITHM PROGRAM.

AIM : Program to perform dijkstra's algorithm

ALGORITHM :

Step 1:START and Include all the header files which are used in the program.

Step 2: Define INFINITY as 999 and MAX 10

Step 3: Read the number of vertices and adjacency weighted matrix.

Step 4: Read the starting node,u.

Step 5: Initialize visited array with false which shows that currently, the tree is empty.

Step 6: Initialize cost array with infinity which shows that it is impossible to reach any node from the start node via a valid path in the tree.

Step 7: The cost to reach the start node will always be zero, hence cost[start]=0.

Step 8: Now at every iteration we choose a node to add in the tree, hence we need n iterations to add n nodes in the tree:

8(a): Choose a node that has a minimum cost and is also currently nonvisited i.e., not present in the tree.

8(b): Update the cost of non-visited nodes which are adjacent to the newly added node with the minimum of the previous and new path.

8(c): Mark the node as visited.

STEP 9: STOP

SOURCE CODE :

```
#include<stdio.h>
#define INFINITY 9999
```

```
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u;
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]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
getch();
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
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];
```

```
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
```

```
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node %d = %d",i,distance[i]);
printf("\nPath = %d",i);
j=i;
do
{
j=pred[j];
printf(" <- %d",j);
}while(j!=startnode);
}
}
```

RESULT : The desired output is obtained.

OUTPUT

```
Enter no. of vertices:4

Enter the adjacency matrix:
0 1 3 0
1 0 1 2
3 1 0 1
0 2 1 0

Enter the starting node:1

Distance of node 0 = 1
Path = 0 <- 1
Distance of node 2 = 1
Path = 2 <- 1
Distance of node 3 = 2
Path = 3 <- 1
```