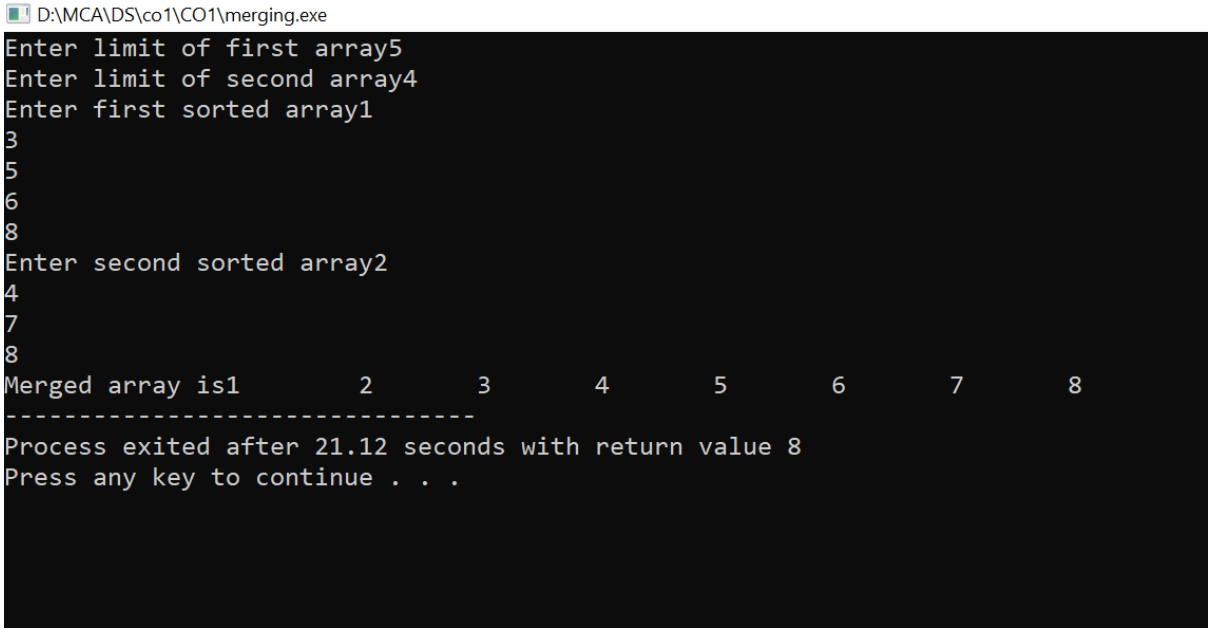
Program

```
#include<stdio.h>
void main()
{
	int i,j,k,a[20],b[20],c[20],m,n;
	printf("Enter limit of first array");
	scanf("%d",&m);
	printf("Enter limit of second array");
	scanf("%d",&n);
	printf("Enter first sorted array");
	for(i=0;i<m;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("Enter second sorted array");
	for(i=0;i<n;i++)
	{
		scanf("%d",&b[i]);
	}
	printf("Merged array is");
	i=0;
	j=0;
	k=0;
	while(i<m && j<n)
	{
		if(a[i]<b[j])
		{
			c[k]=a[i];
			i++;
```

```
        }
        else if(a[i]>b[j])
        {
            c[k]=b[j];
            j++;
        }
        else
        {
            c[k]=a[i];
            j++; i++;
        }
        k++;
    }
    while(i<m)
    {
        c[k]=a[i];
        k++;
        i++;
    }
    while(j<n)
    {
        c[k]=b[j];
        k++;
        j++;
    }
    for(i=0;i<k;i++)
    {
        printf("%d \t",c[i]);
    }
}
```

Output


D:\MCA\DS\co1\CO1\merging.exe
Enter limit of first array5
Enter limit of second array4
Enter first sorted array1
3
5
6
8
Enter second sorted array2
4
7
8
Merged array is1 2 3 4 5 6 7 8

Process exited after 21.12 seconds with return value 8
Press any key to continue . . .

Program

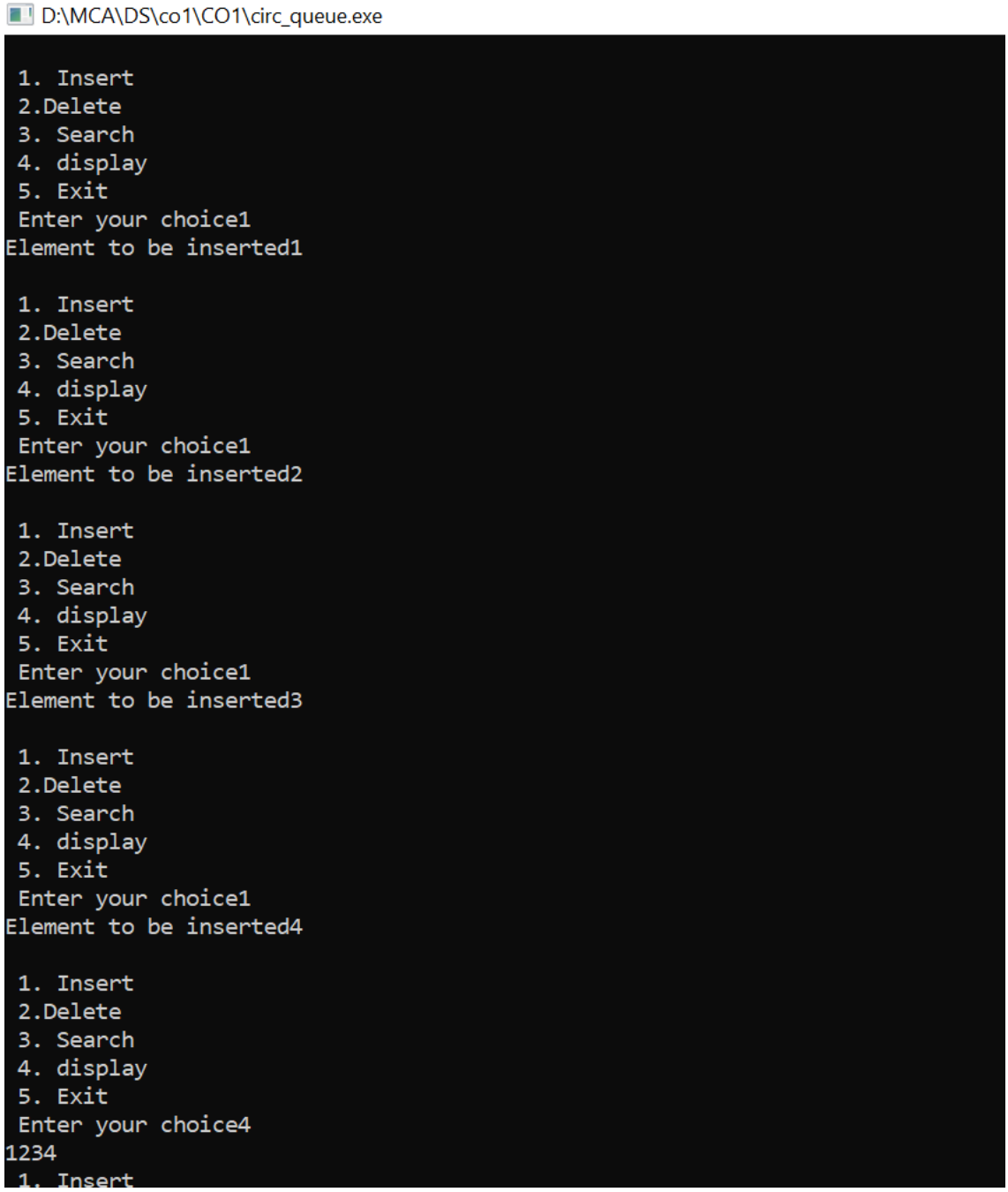
```
#include<stdio.h>
void insert(int *);
void del(int *);
void disp(int *);
void search(int *);
int front=-1,rear=-1;
int size=4;
void main()
{
	int q[20],opt;
	do
	{
		printf("\n 1. Insert \n 2.Delete \n 3. Search \n 4. display \n 5. Exit \n Enter your choice");
		scanf("%d",&opt);
		switch(opt)
		{
			case 1:
				insert(q);
				break;
			case 2:
				del(q);
				break;
			case 3:
				search(q);
				break;

			case 4:
```

```
                    disp(q);
                    break;
            }
      }
while(opt<5);
}
void insert(int *q)
{
        if(front==(rear+1)%size)
        {
                printf("Queue is full");
        return;
        }
        if(front==-1)
                front=0;
        rear=(rear+1)%size;
        printf("Element to be inserted");
        scanf("%d",&q[rear]);
}
void del(int *q)
{
        if(front==-1)
        {
                printf("Queue is empty");
                return;
        }
        printf("Deleted element is %d",q[front]);
        if(front==rear)
                front=rear=-1;
        else
```

```
                front=(front+1)%size;
}
void disp(int *q)
{
        int f;
        if(front==-1)
        {
                printf("Queue is empty");
                return;
        }
        f=front;
        while(1)
        {
                printf("%d",q[f]);
                if(f==rear)
                        break;
                f=(f+1)%size;
        }
}
void search(int *q)
{
        int f,item,c;
        printf("item to be search");
        scanf("%d",&item);
        if(front==-1)
        {
                printf("Queue is empty");
                return;
        }
        f=front;
```

```
        while(1)
        {
        if(f==item)
        {
                printf("item %d is found",item);
                break;
        }
        if(f==rear)
        {
                printf("item not found");
                break;
        }
        f=(f+1)%size;
        }
}
```

Output:


D:\MCA\DS\co1\CO1\circ_queue.exe
1. Insert
2.Delete
3. Search
4. display
5. Exit
Enter your choice1
Element to be inserted1
1. Insert
2.Delete
3. Search
4. display
5. Exit
Enter your choice1
Element to be inserted2
1. Insert
2.Delete
3. Search
4. display
5. Exit
Enter your choice1
Element to be inserted3
1. Insert
2.Delete
3. Search
4. display
5. Exit
Enter your choice1
Element to be inserted4
1. Insert
2.Delete
3. Search
4. display
5. Exit
Enter your choice4
1234
1. Insert

```
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice1
Queue is full
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice2
Deleted element is 1
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice4
234
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice1
Element to be inserted7

 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice3
item to be search3
item 3 is found
 1. Insert
 2.Delete
 3. Search
 4. display
```

```
item to be search3
item 3 is found
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice2
Deleted element is 2
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice4
347
 1. Insert
 2.Delete
 3. Search
 4. display
 5. Exit
 Enter your choice5

--------------------------------
Process exited after 143.8 seconds with return value 5
Press any key to continue . . .
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct Node *Top=NULL;
void push();
void display();
void pop();
void search();
struct Node
{
	int data;
	struct Node *next;
};
void main()
{
	int q[20],opt;
	do
	{
		printf("\n 1. push \n 2.pop \n 3. Search \n 4. display \n 5. Exit \n Enter your
	choice");
		scanf("%d",&opt);
		switch(opt)
		{
			case 1:
				push();
				break;
			case 2:
				pop();
				break;
			case 3:
```

```
                search();
                break;
            case 4:
                display();
                break;
        }
    }
while(opt<5);
}
void push()
{
    int x;
    struct Node *ne;
    printf("Read value");
    scanf("%d",&x);
    ne=(struct Node*)malloc(sizeof(struct Node));
    if(ne==NULL)
    {
        printf("stack overflow");
        return;
    }
    else
    {
        ne->data=x;
        ne->next=Top;
        Top=ne;
    }
}
void pop()
{
```

```
        struct Node *ptr=Top;
        int item;
        if(ptr==NULL)
        {
                printf("stack is empty");
                return;
        }
        else
        {
                item=ptr->data;
                printf("deleted element:%d",item);
                Top=ptr->next;
        }
}
void search()
{
        int item;
        struct Node *ptr=Top;
        printf("element to be search");
        scanf("%d",&item);
        if(Top==NULL)
        {
                printf("Stack is empty");
                return;
        }
        else
        {
                while(ptr!=NULL)
                {
                        if(ptr->data==item)
```

```
                {
                        printf("item found");
                        printf("%d  ",ptr->data);return;
                }
                ptr=ptr->next;
            }
            if(ptr==NULL)
            {
                printf("item not found");
                return;
            }
        }
    }
    void display()
    {
        struct Node *ptr=Top;
        if(Top==NULL)
        {
            printf("Stack is empty");
            return;
        }
        else
        {
            while(ptr!=NULL)
            {
                printf("%d  ",ptr->data);
                ptr=ptr->next;
            }
        }
}
```

Output

D:\MCA\DS\co1\CO1\stack_using_linkedlist.exe

```
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value1

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value20

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value40

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice4
40  20  1
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice2
deleted element:40
 1. push
 2.pop
```

```
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice4
20  1
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value90

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value80

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice4
80  90  20  1
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice3
element to be search80
item found80
 1. push
 2.pop
 3. Search
```

```
item found80
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value90

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice1
Read value80

 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice4
80  90  80  90  20  1
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice3
element to be search80
item found80
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice3
element to be search60
item not found
 1. push
```

```
element to be search60
item not found
 1. push
 2.pop
 3. Search
 4. display
 5. Exit
 Enter your choice5

--------------------------------
Process exited after 232.6 seconds with return value 5
Press any key to continue . . .
```

Program

```
#include<stdio.h>
#include<stdlib.h>
void insertf();
void insertl();
void display();
void insertp();
void deletef();
void deletel();
void search();
void deletep();
struct Node
{
        struct Node *left;
        struct Node *right;
        int data;
}*head=NULL;
void main()
{
        int opt;
        do
        {
                printf("\n 1. insert at front \n 2.insert last \n 3. insert position\n 4. delete front
                \n 5. delete last \n 6.delete from position \n 7. display \n 8. search \n 9. Exit \n
                Enter your choice");
                scanf("%d",&opt);
                switch(opt)
                {
                        case 1:
                                insertf();
```

```
                break;
            case 2:
                insertl();
                break;
            case 3:
                insertp();
                break;
            case 4:
                deletef();
                break;
            case 5:
                deletel();
                break;
            case 6:
                deletep();
                break;
            case 7:
                display();
                break;
            case 8:
                search();
                break;
            case 9:
                break;
            default:
                printf("invalid number");
        }
    }
while(opt!=9);
}
```

```
void insertf()
{
        int item;
        struct Node *ne;
        printf("Enter item");
        scanf("%d",&item);
        ne=(struct Node*)malloc(sizeof(struct Node));
        if(ne==NULL)
        {
                printf("insufficient memory");
                return;
        }
        ne->data=item;
        ne->left=NULL;
        ne->right=NULL;
        if(head==NULL)
        {
                head=ne;
                return;
        }
        ne->right=head;
        head->left=ne;
        head=ne;
}
void insertl()
{
        int item;
        struct Node *ptr,*ne;
        printf("Enter item");
        scanf("%d",&item);
```

```
        ne=(struct Node*)malloc(sizeof(struct Node));
        if(ne==NULL)
        {
                printf("insufficient memory");
                return;
        }
        ne->data=item;
        ne->left=NULL;
        ne->right=NULL;
        if(head==NULL)
        {
                head=ne;
                return;
        }
        ptr=head;
        while(ptr->right!=NULL)
        {
                ptr=ptr->right;
        }
        ptr->right=ne;
        ne->left=ptr;
}
void display()
{
        struct Node *ptr;
        ptr=head;
        if(head==NULL)
        {
                printf("list empty");
                return;
```

```
        }
        while(ptr!=NULL)
        {
                printf("%d ",ptr->data);
                ptr=ptr->right;
        }
}
void insertp()
{
        int item,key;
        struct Node *ptr,*ne;
        printf("Enter item");
        scanf("%d",&item);
        printf("key value");
        scanf("%d",&key);
        ne=(struct Node*)malloc(sizeof(struct Node));
        if(ne==NULL)
        {
                printf("insufficient memory");
                return;
        }
        ne->data=item;
        ne->left=NULL;
        ne->right=NULL;
        if(head==NULL)
        {
                head=ne;return;
        }
        ptr=head;
        while(ptr->data!=key && ptr->right!=NULL)
```

```
        {
                ptr=ptr->right;
        }
        if(ptr->right==NULL)
        {
                ptr->right=ne;
                ne->left=ptr;
        }
        else
        {
                ne->right=ptr->right;
                ptr->right->left=ne;
                ne->left=ptr;
                ptr->right=ne;
                return;
        }
}
void deletef()
{
        struct Node *ptr;
        ptr=head;
        if(head==NULL)
        {
                printf("list is empty");
                return;
        }
        head=head->right;
        if(head!=NULL)
        {
                head->left=NULL;
```

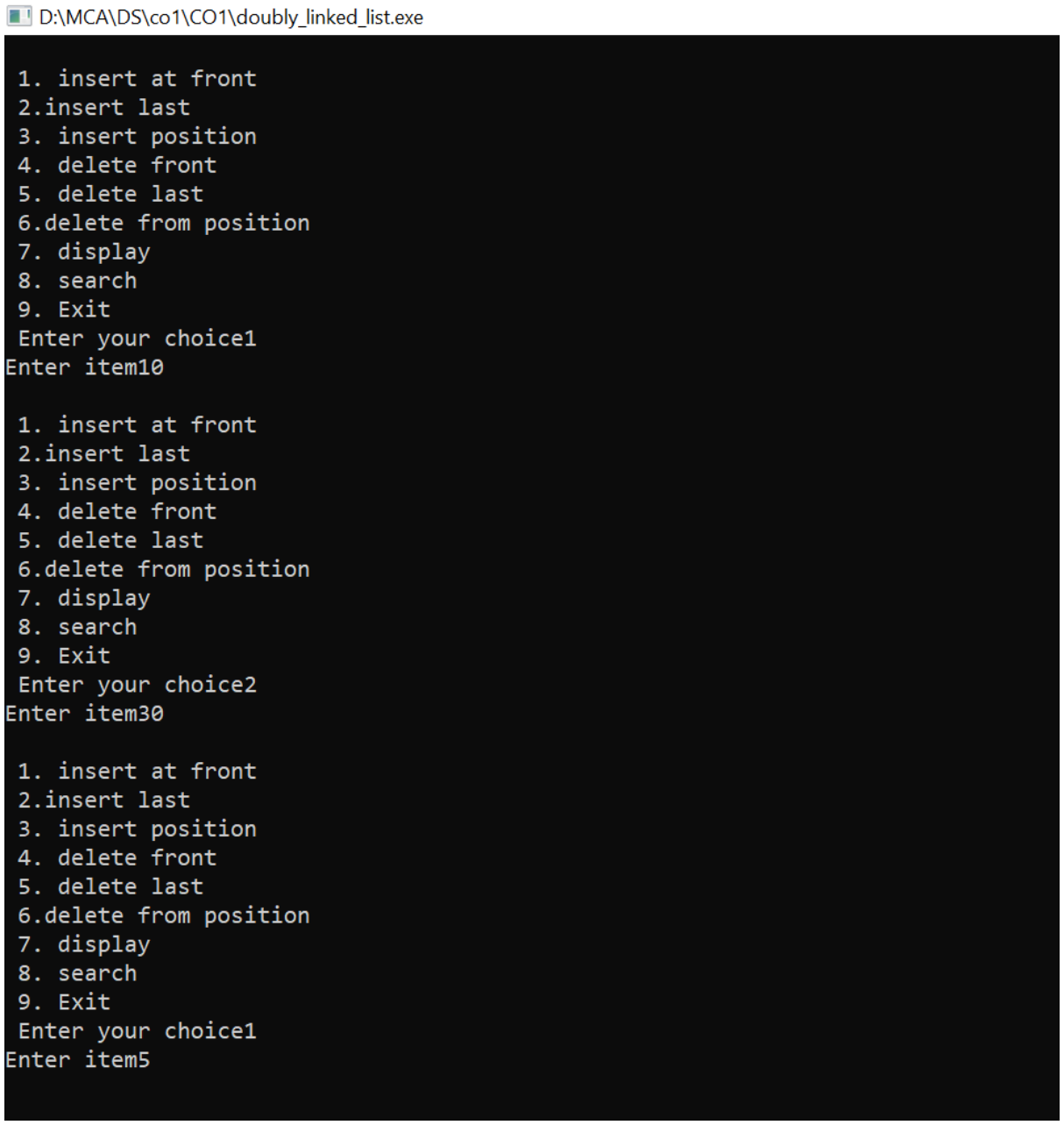
```
        }
        free(ptr);
}
void deletel()
{
        struct Node *ptr,*p;
        if(head==NULL)
        {
                printf("list is empty");
                return;
        }
        if(head->right==NULL)
        {
                free(head);
                head=NULL;
                return;
        }
        ptr=head;while(ptr->right!=NULL)
        {
                ptr=ptr->right;
        }
        p=ptr;
        ptr=ptr->left;
        ptr->right=NULL;
        free(p);
}
void deletep()
{
        struct Node *ptr,*next,*prev;
        int key;
```

```
printf("item to be deleted");
scanf("%d",&key);
if(head==NULL)
{
        printf("list is empty");
        return;
}
if(head->data==key)
{
        ptr=head;
        head=ptr->right;
        if(head!=NULL)
        {
                head->left=NULL;
        }
        free(ptr);
        return;
}
ptr=head;
while(ptr->data!=key && ptr->right!=NULL)
{
        ptr=ptr->right;
}
if(ptr->data==key)
{
        next=ptr->right;
        prev=ptr->left;
        if(next!=NULL)
        {
                prev->right=next;
```

```
                next->left=ptr->left;
            }
            else
            {
                prev->right=NULL;
            }
            free(ptr);return;
        }
        else
        {
            printf("Element not found");
            return;
        }
}
void search()
{
        struct Node *ptr;
        int item;
        ptr=head;
        printf("Element to be search");
        scanf("%d",&item);
        if(head==NULL)
        {
            printf("list empty");
            return;
        }
        while(ptr!=NULL)
        {
            if(ptr->data==item)
            {
```

```
                    printf("%d is found",ptr->data);
                    return;
                }
                ptr=ptr->right;
            }
            if(ptr==NULL)
            {
                printf("element not found");
                return;
            }
        }
}
```

Output


D:\MCA\DS\co1\CO1\doubly_linked_list.exe
1. insert at front
2.insert last
3. insert position
4. delete front
5. delete last
6.delete from position
7. display
8. search
9. Exit
Enter your choice1
Enter item10
1. insert at front
2.insert last
3. insert position
4. delete front
5. delete last
6.delete from position
7. display
8. search
9. Exit
Enter your choice2
Enter item30
1. insert at front
2.insert last
3. insert position
4. delete front
5. delete last
6.delete from position
7. display
8. search
9. Exit
Enter your choice1
Enter item5

D:\MCA\DS\co1\CO1\doubly_linked_list.exe

```
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice3
Enter item20
key value10

 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice7
5 10 20 30
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice4

 1. insert at front
```

D:\MCA\DS\co1\CO1\doubly_linked_list.exe

```
 Enter your choice4

 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice7
10 20 30
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice5

 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice7
10 20
 1. insert at front
 2.insert last
```

D:\MCA\DS\co1\CO1\doubly_linked_list.exe

```
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice1
Enter item3

 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice7
3 10 20
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice6
item to be deleted10

 1. insert at front
 2.insert last
```

D:\MCA\DS\co1\CO1\doubly_linked_list.exe

```
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice7
3 20
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice8
Element to be search24
element not found
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice8
Element to be search3
3 is found
 1. insert at front
 2.insert last
```

```
 9. Exit
 Enter your choice8
Element to be search3
3 is found
 1. insert at front
 2.insert last
 3. insert position
 4. delete front
 5. delete last
 6.delete from position
 7. display
 8. search
 9. Exit
 Enter your choice9

--------------------------------
Process exited after 151.9 seconds with return value 9
Press any key to continue . . .
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
	struct Node *left;
	struct Node *right;
	int data;
}*root=NULL;
void insertion()
{
	int n;
	struct Node *ne,*ptr,*ptr1;
	ne=(struct Node*)malloc(sizeof(struct Node));
	printf("Enter data");
	scanf("%d",&n);
	if(ne==NULL)
	{
		printf("insufficient memory");
		return;
	}
	ne->right=NULL;
	ne->left=NULL;
	ne->data=n;
	if(root==NULL)
	{
		root=ne;
		//break;
	}
```

```
ptr=root;
while(ptr!=NULL)
{
        if(ptr->data==n)
        {
                printf("data is present");
                return;
        }
        else if(ptr->data>n)
        {
                ptr1=ptr;
                ptr=ptr->left;
        }
        else
        {
                ptr1=ptr;
                ptr=ptr->right;
        }
}
if(ptr==NULL)
{
        if(n>ptr1->data)
        {
                ptr1->right=ne;
        }
        else
        {
                ptr1->left=ne;
        }
}
```

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

	}
	return;
}
void postorder(struct Node *root)
{
	struct Node *p=root;
	if(p!=NULL)
	{
```

```
                postorder(p->left);
                postorder(p->right);
                printf("%d  ",p->data);

        }
        return;
}
void search(struct Node *root)
{
        int x;
        struct Node *ptr=root;
        printf("Element to be search");
        scanf("%d",&x);
        while(ptr!=NULL)
        {
                if(ptr->data==x)
                {
                        printf("Data present");
                        break;
                }
                if(x>ptr->data)
                {
                        ptr=ptr->right;
                }
                else
                {
                        ptr=ptr->left;
                }
        }
        if(ptr==NULL)
```

```
                printf("Data is not present");
}
void deletion(struct Node *root,int x)
{
        struct Node *ptr=root,*parent,*p;
        if(root==NULL)
        {
                printf("\n Tree is empty");
                return;
        }
        parent=NULL;
        while(ptr!=NULL)
        {
                if(ptr->data==x)
                                break;
                parent=ptr;
                if(x>ptr->data)
                                ptr=ptr->right;
                else
                                ptr=ptr->left;
        }
        if(ptr==NULL)
        {
                printf("Item not found");
                return;
        }
        //case1
        if(ptr->right==NULL && ptr->left==NULL)
        {
                if(parent==NULL)
```

```
                        root=NULL;
        else if(parent->right==ptr)
                        ptr->right=NULL;
        else
                        ptr->left=NULL;
        printf("Element deleted");
        free(ptr);
        return;
}
//case3
int dat;
if(ptr->right!=NULL && ptr->left!=NULL)
// find inorder successor
{
        p=ptr->right;
        while(p->left!=NULL)
                        p=p->left;
        dat=p->data;
        deletion(root,p->data);
        ptr->data=dat;
}
//case 2
if(parent==NULL)
{
        if(ptr->right==NULL)
                        root=ptr->left;
        else
                        root=ptr->right;
}
else if(parent->right==ptr)
```

```
        {
                if(ptr->right==NULL)
                                parent->right=ptr->left;
                else
                                parent->right=ptr->right;
        }
        else
        if(ptr->left==NULL)
                        parent->left=ptr->right;
        else
                        parent->left=ptr->left;
        printf("Element deleted");
        free(ptr);
        return;
}
void main()
{
        int opt,x;
        do
        {
        printf("\n 1. insertion \n 2.deletion \n 3. inorder traverse\n 4. preorder traverse \n");
        printf("5. postorder traverse \n 6.search\n 7. Exit \n Enter your choice");
        scanf("%d",&opt);
        switch(opt)
        {
                case 1:
                        insertion();
                        break;
                case 2:
                        printf("Element to be delete");
```

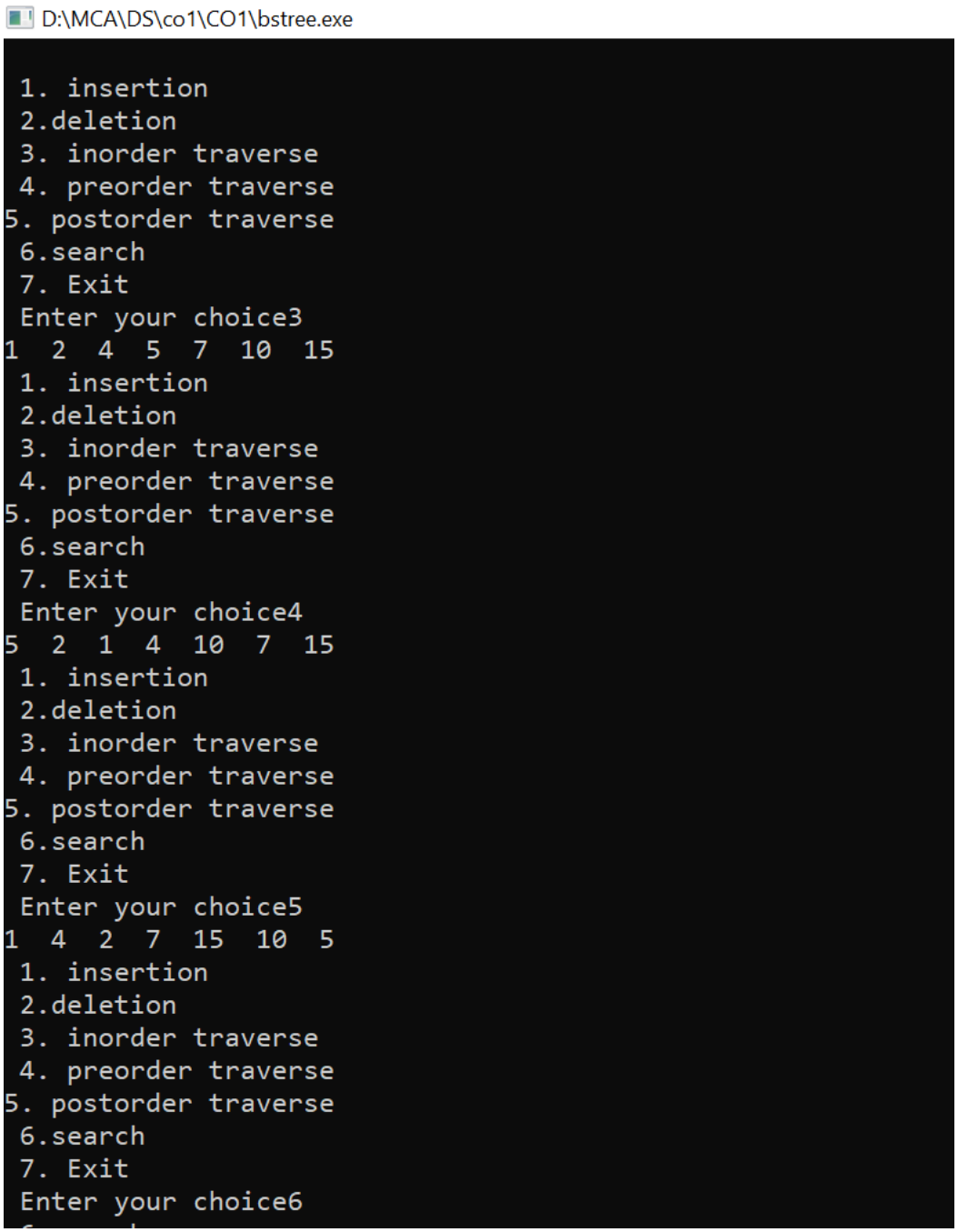
```
                scanf("%d",&x);
                deletion(root,x);
                break;
        case 3:
                //struct Node *p=root;
                inorder(root);
                break;
        case 4:
                preorder(root);
                break;
        case 5:
                postorder(root);
                break;
        case 6:
                search(root);
                break;
        case 7:
                break;
        default:
                printf("invalid choice");
    }
  }
   while(opt!=7);
}
```

Output

D:\MCA\DS\co1\CO1\bstree.exe

```
 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data5
data is present
 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data2

 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data10

 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
```

```
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data1

 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data4

 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data7

 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice1
Enter data15
```


D:\MCA\DS\co1\CO1\bstree.exe
1. insertion
2.deletion
3. inorder traverse
4. preorder traverse
5. postorder traverse
6.search
7. Exit
Enter your choice3
1 2 4 5 7 10 15
1. insertion
2.deletion
3. inorder traverse
4. preorder traverse
5. postorder traverse
6.search
7. Exit
Enter your choice4
5 2 1 4 10 7 15
1. insertion
2.deletion
3. inorder traverse
4. preorder traverse
5. postorder traverse
6.search
7. Exit
Enter your choice5
1 4 2 7 15 10 5
1. insertion
2.deletion
3. inorder traverse
4. preorder traverse
5. postorder traverse
6.search
7. Exit
Enter your choice6

```
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice6
Element to be search5
Data present
 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice2
Element to be delete15
Element deleted
 1. insertion
 2.deletion
 3. inorder traverse
 4. preorder traverse
5. postorder traverse
 6.search
 7. Exit
 Enter your choice3
1  2  4  5  7  10
--------------------------------
Process exited after 59.75 seconds with return value 3221225725
Press any key to continue . . .
```

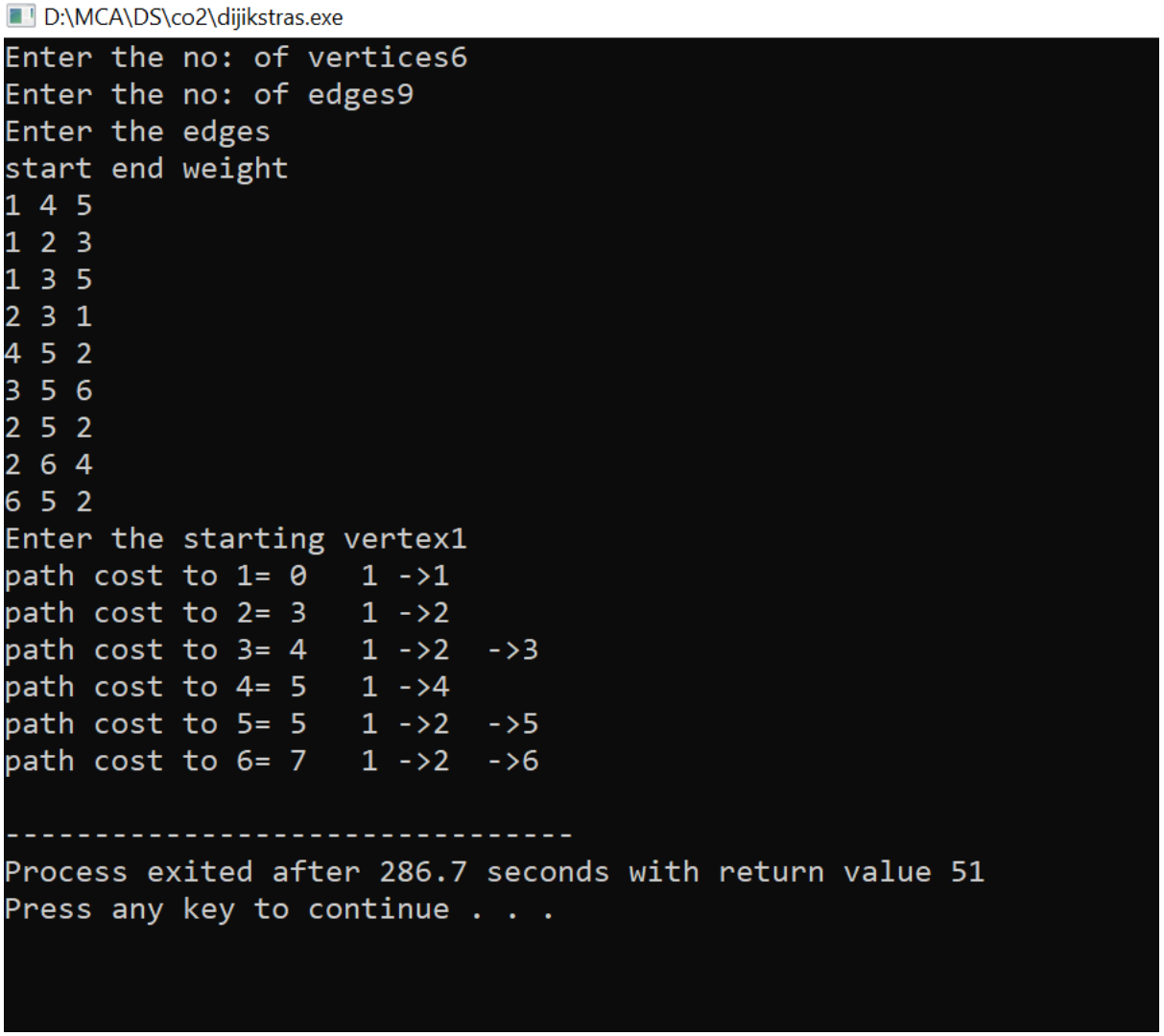
Program

```
#include<stdio.h>
#include<conio.h>
#define inf 999
void printpath(int,int);
int extractmin();
int v,adj[20][20],dist[20],visit[20],pred[20];
void main()
{
        int e,st,en,w,i,j,src,ver,k;
        printf("Enter the no: of vertices");
        scanf("%d",&v);
        printf("Enter the no: of edges");
        scanf("%d",&e);
        for(i=0;i<=v;i++)
        {
                for(j=0;j<=v;j++)
                adj[i][j]=inf;
                dist[i]=inf;
                visit[i]=0;
        }
        printf("Enter the edges\n");
        printf("start end weight\n");
        for(i=1;i<=e;i++)
        {
                scanf("%d%d%d",&st,&en,&w);
                adj[st][en]=w;
        }
        printf("Enter the starting vertex");
```

```
scanf("%d",&src);
dist[src]=0;
pred[src]=src;
for(k=1;k<=v;k++)
{
        ver=extractmin();
        visit[ver]=1;
        if (dist[ver]==inf) continue;
        for(i=1;i<=v;i++)
        {
                if (adj[ver][i]!=inf&& visit[i]==0 )
                        if (dist[i]>dist[ver]+adj[ver][i])
                        {
                                dist[i]=dist[ver]+adj[ver][i] ;
                                pred[i]=ver;
                        }
        }
}
for(i=1;i<=v;i++)
{
        if (dist[i]==inf)
                continue;
         printf("path cost to %d= %d   ",i,dist[i]);
         if( dist[i]!=inf)
         {
                 printpath(i,src);
                 printf("->%d",i);
                 printf("\n");
          }
}
```

```
   getch();
}
void printpath(int i,int src)
{
        if (pred[i]==src)
        {
                printf("%d ",src);return;
         }
        printpath(pred[i],src);
        printf("->%d  ",pred[i]);
}
int extractmin()
{
        int min=inf,i,ver;
         for(i=1;i<=v;i++)
         {
                if (visit[i]==0 && dist[i]<min)
                {
                        min=dist[i];
                        ver=i;
                }
         }
    return ver;
}
```

Output


D:\MCA\DS\co2\dijikstras.exe
Enter the no: of vertices6
Enter the no: of edges9
Enter the edges
start end weight
1 4 5
1 2 3
1 3 5
2 3 1
4 5 2
3 5 6
2 5 2
2 6 4
6 5 2
Enter the starting vertex1
path cost to 1= 0 1 ->1
path cost to 2= 3 1 ->2
path cost to 3= 4 1 ->2 ->3
path cost to 4= 5 1 ->4
path cost to 5= 5 1 ->2 ->5
path cost to 6= 7 1 ->2 ->6

Process exited after 286.7 seconds with return value 51
Press any key to continue . . .

Program

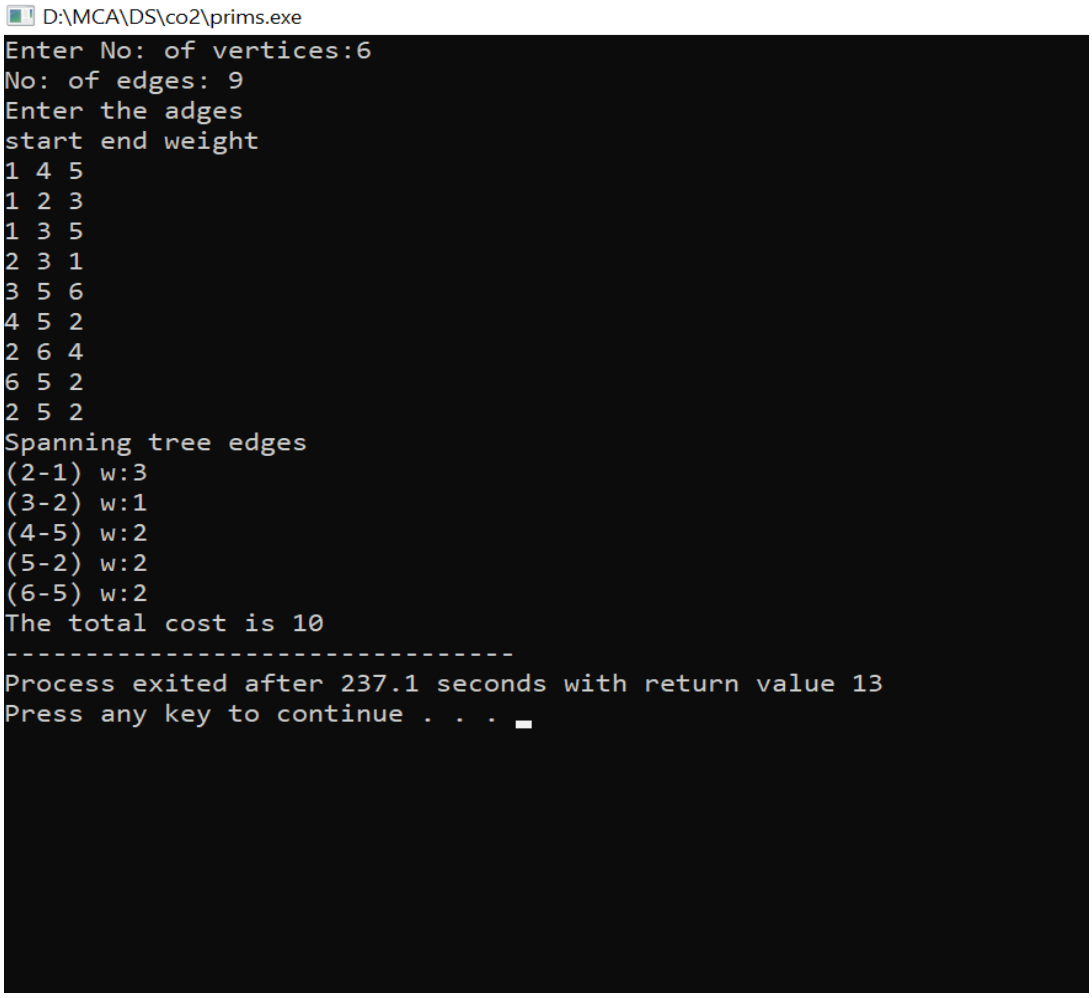
```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define inf 999
void addtoadjlist(int s,int en,int w);
int emptyQ();
int extractminQ();
struct node
{
        int vertex;
        int weight;
        struct node *next;
}*adj[20];
int v;
int p[20],key[20],q[20];
void main()
{
        int i,s,en,we,e,u,w,sum=0;
         struct node *ptr;
         printf("Enter No: of vertices:");
         scanf("%d",&v);
         for(i=1;i<=v;i++)
         {
                 p[i]=0;
                 key[i]=inf;
                 q[i]=1;
                 adj[i]=NULL;
          }
```

```
printf("No: of edges: ");
scanf("%d",&e);
printf("Enter the adges\n");
printf("start end weight");
for(i=1;i<=e;i++)
{
        scanf("%d%d%d",&s,&en,&we);
        addtoadjlist(s,en,we);
        addtoadjlist(en,s,we);
}
key[1]=0;
while(!emptyQ())
{
        u=extractminQ();
        ptr=adj[u];
        while(ptr!=NULL)
        {
                w=ptr->vertex;
                if (q[w]==1 && ptr->weight < key[w])
                {
                        key[w]=ptr->weight;
                        p[w]=u;
                }
                ptr=ptr->next;
        }
}
sum=0;
printf("Spanning tree edges\n");
for(i=2;i<=v;i++)
{
```

```
                printf("(%d-%d) w:%d \n",i,p[i],key[i]);
                sum=sum+key[i];
        }
        printf("The total cost is %d",sum);
        getch();
}
int emptyQ()
{
        int i,flag=1;
        for(i=1;i<=v;i++)
        {
                if (q[i]==1)
                {
                        flag=0;
                        break;
                }
        }
        return flag;
}
int extractminQ()
{
        int i,min=inf,ver;
        for(i=1;i<=v;i++)
        {
                if (key[i]<min && q[i]==1)
                {
                        ver=i;
                        min=key[i];
                }
        }
```

```
        q[ver]=0;
        return ver;
}
void addtoadjlist(int s,int en,int w)
{
        struct node *ne=(struct node *)malloc(sizeof(struct node));
        ne->vertex=en;
        ne->weight=w;
        ne->next=adj[s];
        adj[s]=ne;
 }
```

Output


D:\MCA\DS\co2\prims.exe
Enter No: of vertices:6
No: of edges: 9
Enter the adges
start end weight
1 4 5
1 2 3
1 3 5
2 3 1
3 5 6
4 5 2
2 6 4
6 5 2
2 5 2
Spanning tree edges
(2-1) w:3
(3-2) w:1
(4-5) w:2
(5-2) w:2
(6-5) w:2
The total cost is 10

Process exited after 237.1 seconds with return value 13
Press any key to continue . . .

Program

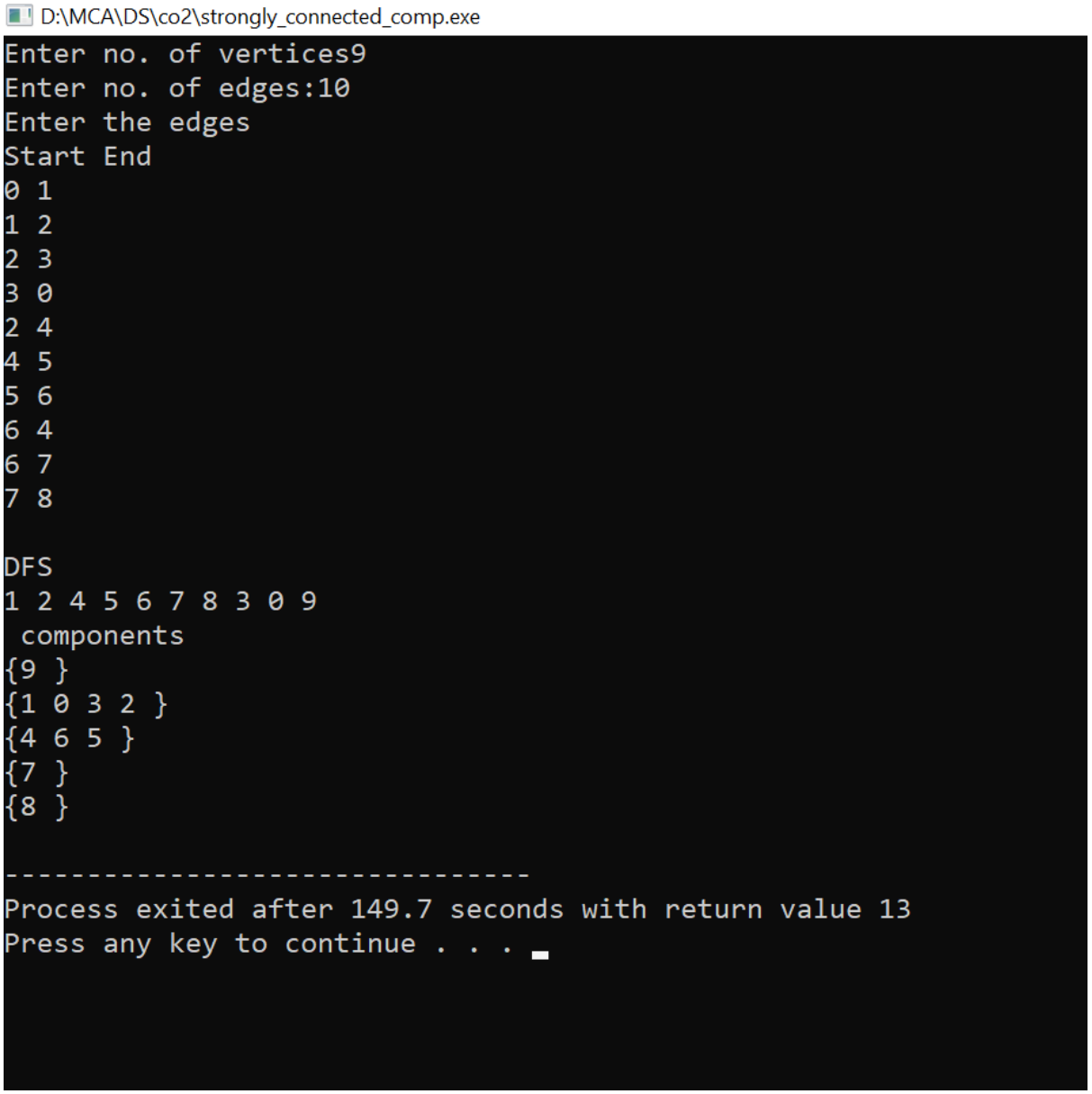
```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
        int vertex;
        struct node *next;
};
int v,e;
struct node *adj[20], *adj1[20];
int t=0,visited[20],ft[20];
void dfs();
void dfsvisit(int);
void dfs1();
void dfsvisit1(int);
void adjlistrep(struct node **adj,int s,int en)
{
        struct node *ne=(struct node*)malloc(sizeof(struct node));
        ne->vertex=en;
        ne->next=adj[s];
        adj[s]=ne;
}
void main()
{
        int s,i,en;
        struct node *ptr;
        printf("Enter no. of vertices");
        scanf("%d",&v);
```

```
	for(i=0;i<=v;i++)
		adj[i]=adj1[i]=NULL;
	printf("Enter no. of edges:");
	scanf("%d",&e);
	printf("Enter the edges\n");
	printf("Start End\n");
	for(i=0;i<e;i++)
	{
		scanf("%d%d",&s,&en);
		adjlistrep(adj,s,en);
		adjlistrep(adj1,en,s);
	}
	dfs();
	dfs1();
	getch();
}
void dfs()
{
	int i;
	for(i=0;i<=v;i++)
		visited[i]=0;
	printf("\nDFS\n");
	for(i=1;i<=v;i++)
	{
		if(visited[i]==0)
		{
			dfsvisit(i);
		}
	}
}
```

```
void dfsvisit(int u)
{
	int w;
	struct node *ptr;
	visited[u]=1;
	printf("%d ",u);
	ptr=adj[u];
	while(ptr!=NULL)
	{
		w=ptr->vertex;
		if(visited[w]==0)
			dfsvisit(w);
		ptr=ptr->next;
	}
	t++;
	ft[u]=t;
}
void dfs1()
{
	int i,max=0,ver;
	printf("\n components\n");
	for(i=0;i<=v;i++)
		visited[i]=0;
	while(1)
	{
		max=0;
		for(i=1;i<=v;i++)
		{
			if(visited[i]==0 && ft[i]>max)
			{
```

```
                    ver=i;
                    max=ft[i];
                }
            }
            if(max==0)
                break;
            printf("{");
            dfsvisit1(ver);
            printf("}\n");
        }
}
void dfsvisit1(int u)
{
        int w;
        struct node *ptr;
        visited[u]=1;
        printf("%d ",u);
        ptr=adj1[u];
        while(ptr!=NULL)
        {
            w=ptr->vertex;
            if(visited[w]==0)
                dfsvisit1(w);
            ptr=ptr->next;
        }
}
```

Output:


D:\MCA\DS\co2\strongly_connected_comp.exe
Enter no. of vertices9
Enter no. of edges:10
Enter the edges
Start End
0 1
1 2
2 3
3 0
2 4
4 5
5 6
6 4
6 7
7 8
DFS
1 2 4 5 6 7 8 3 0 9
components
{9 }
{1 0 3 2 }
{4 6 5 }
{7 }
{8 }

Process exited after 149.7 seconds with return value 13
Press any key to continue . . .

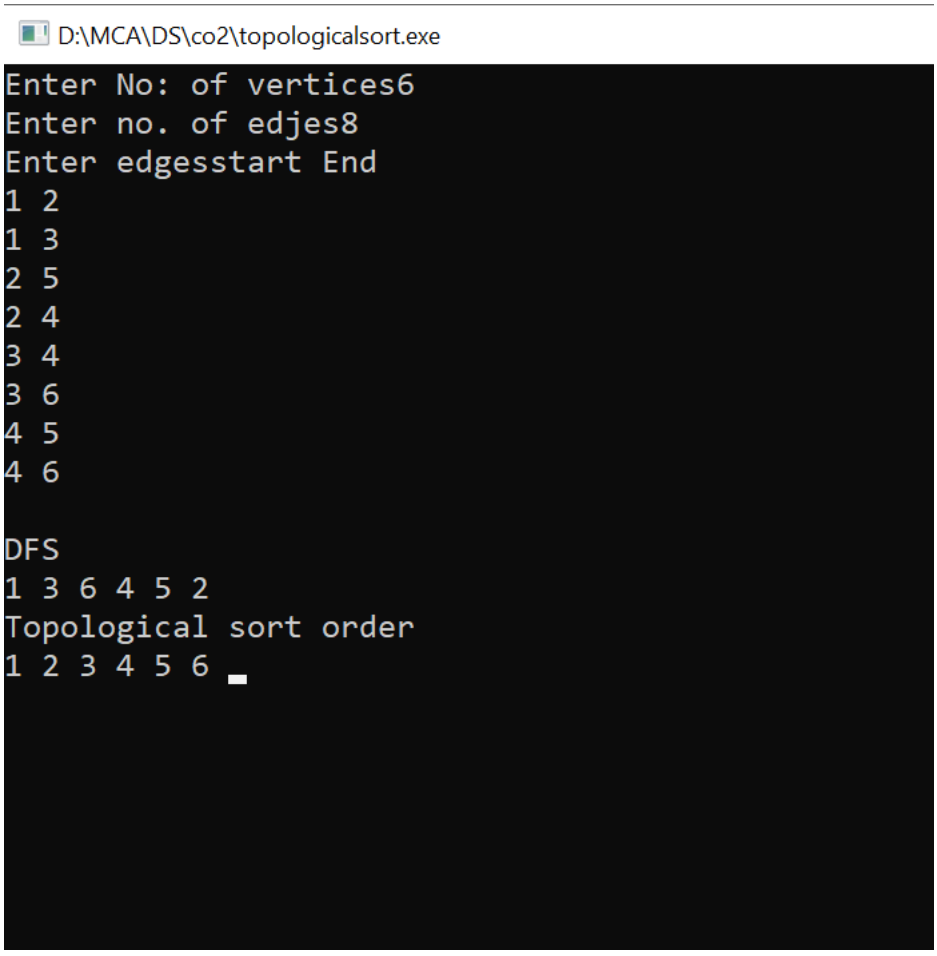
Program

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
        int vertex;
        struct node *next;
}*adj[20];
int v,e;
int visited[20],top[20];
int t=0;
void dfs();
void dfsvisit();
void main()
{
        int s,i,en;
        struct node *ne;
        printf("Enter No: of vertices");
        scanf("%d",&v);
        for(i=0;i<=v;i++)
                adj[i]=NULL;
        printf("Enter no. of edjes");
        scanf("%d",&e);
        printf("Enter edges");
        printf("start End\n");
        for(i=0;i<e;i++)
        {
                scanf("%d%d",&s,&en);
```

```
                ne=(struct node*)malloc(sizeof(struct node));
                ne->vertex=en;
                ne->next=adj[s];
                adj[s]=ne;
        }
        dfs();
        printf("\nTopological sort order\n");
        for(i=t-1;i>=0;i--)
                printf("%d ",top[i]);
        getch();
}
void dfs()
{
        int i;
        for(i=0;i<=v;i++)
                visited[i]=0;
        printf("\nDFS\n");
        for(i=1;i<=v;i++)
                if(visited[i]==0)
                        dfsvisit(i);
}
void dfsvisit(int u)
{
        int w;
        struct node *ptr;
        visited[u]=1;
        printf("%d ",u);
        ptr=adj[u];
        while(ptr!=NULL)
        {
```

```
            w=ptr->vertex;
            if(visited[w]==0)
                dfsvisit(w);
            ptr=ptr->next;
        }
        top[t]=u;
        t++;
}
```

Output


D:\MCA\DS\co2\topologicalsort.exe
Enter No: of vertices6
Enter no. of edjes8
Enter edgesstart End
1 2
1 3
2 5
2 4
3 4
3 6
4 5
4 6
DFS
1 3 6 4 5 2
Topological sort order
1 2 3 4 5 6

Program

```
#include<stdio.h>
#include<stdlib.h>
int red=1,black=0;
struct node
 {
        int data,color;
        struct node *right, *left;
 };
void doop(struct node *,struct node *,struct node *);
void RRRotation(struct node *);
void LLRotation(struct node *);
struct node *ROOT=NULL;
struct node *findParent(struct node *n) ;
//function to reserve memory for a node
struct node * getNode()
 {
        struct node *ne;
        ne=(struct node *) malloc(sizeof(struct node));
        if (ne==NULL)
        printf("No Memory");
        return ne;
 }
//function to find the parent node of a node
struct node* findParent(struct node *n)
 {
        struct node *ptr=ROOT,*parent=NULL;
        int x=n->data;
   while(ptr!=n)
```

```
    {
        parent=ptr;
        if(x>ptr->data)
                ptr=ptr->right;
        else
                ptr=ptr->left;
    }
  return parent;
 }
//function to insert a value in the Binary search tree
void insert()
 {
        int x;
        struct node *ne,*parent,*ptr,*pparent,*uncle;
        //Perform standard BST insertion and make the colour of newly inserted
        nodes as RED.
        printf("Enter the element to insert");
        scanf("%d",&x);
        ne=getNode();
        if(ne==NULL)
                return;
        ne->data=x;
        ne->left=ne->right=NULL;
        ne->color=red;
        //If x is the root, change the colour of x as BLACK and return
        if(ROOT==NULL)
        {
                ROOT=ne;
                ne->color=black;
                return;
        }
```

```
ptr=ROOT;
while(ptr!=NULL)
{
	if(ptr->data==x)
	{
		printf("Data already present");
		break;
	}
	parent=ptr;
	if(x>ptr->data)
		ptr=ptr->right;
	else
		ptr=ptr->left;
}
if(ptr!=NULL)
	return;
if(x>parent->data)
	parent->right=ne;
else
	parent->left=ne;

while(ne!=ROOT)
{
	//find uncle
	parent=findParent(ne);
	if(parent->color==black)
		break;
	if(parent->color==red)
	{
		pparent=findParent(parent);
```

```
if(pparent->right==parent)
        uncle=pparent->left;
else
        uncle=pparent->right;
//If x's uncle is BLACK, or NULL then call doop()
if(uncle==NULL)
{
                doop(ne,parent,pparent);
                break;
}
if(uncle->color==black)
{
                doop(ne,parent,pparent);
                break;
}
```

/* If x's uncle is RED (Grandparent must have been black from property 4)
(1)Change the colour of parent and uncle as BLACK.
(2) Colour of a grandparent as RED.
(3) Change x = x's grandparent, repeat steps 2 and 3 for new x. */

```
if(uncle->color==red)
{
        parent->color=uncle->color=black;
        if (pparent!=ROOT)
        {
                if(pparent->color==red)
                        pparent->color=black;
                else
                        pparent->color=red;
```

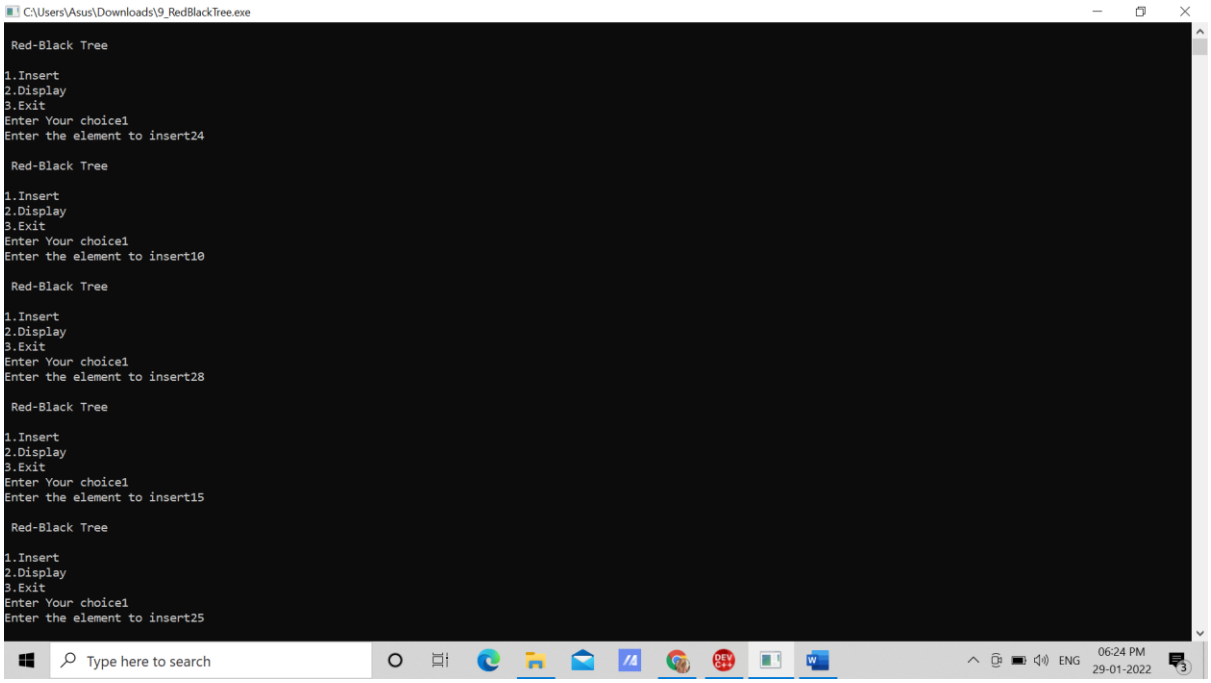
```
                                if(pparent->color==red)
                                        ne=pparent;
                        }
                        else
                                break;
                }
        }
}
//function for inorder traversal
void inorder(struct node *ptr)
{
        if(ptr!=NULL)
        {
                inorder(ptr->left);
                printf("%d(%c) ",ptr->data,ptr->color==0?'b':'r');
                inorder(ptr->right);
        }
 }
void doop(struct node *ne,struct node *parent,struct node *pparent)
{
   /*(i) Left Left Case (p is left child of g and x is left child of p)
    (ii) Left Right Case (p is left child of g and x is the right child of p)
    (iii) Right Right Case (Mirror of case i)
     (iv) Right Left Case (Mirror of case ii)*/
  if(ne==parent->left && parent==pparent->left)
  {
        struct node *left=pparent->left;
        LLRotation(pparent);
        parent->color=parent->color==1?0:1;
```

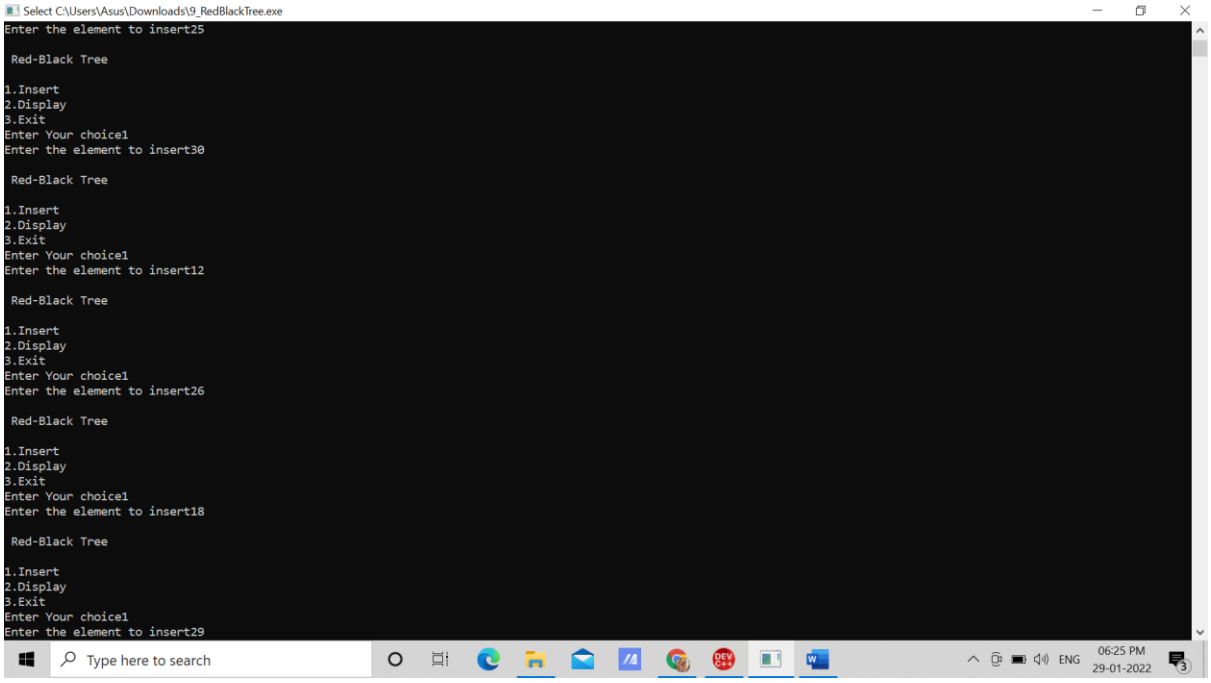
```
        pparent->color=pparent->color==1?0:1;
        if(pparent==ROOT)
                ROOT=left;
 }
else if (parent==pparent->left && ne==parent->right)
{
        struct node *left=pparent->left;
        RRRotation(parent);
        LLRotation(pparent);
        ne->color=ne->color==1?0:1;
        pparent->color=pparent->color==1?0:1;
        if(pparent==ROOT)
                ROOT=left;
  }
else if( ne==parent->right && parent==pparent->right)
{
        struct node *right=pparent->right;
        RRRotation(pparent);
        parent->color=parent->color==0?1:0;
        pparent->color=pparent->color==0?1:0;
        if(pparent==ROOT)
                ROOT=right;
 }
else if (parent==pparent->right && ne==parent->left)
{
        struct node *right=pparent->right;
        LLRotation(parent);
        RRRotation(pparent);
        pparent->color=pparent->color==1?0:1;
        ne->color=ne->color==1?0:1;
```

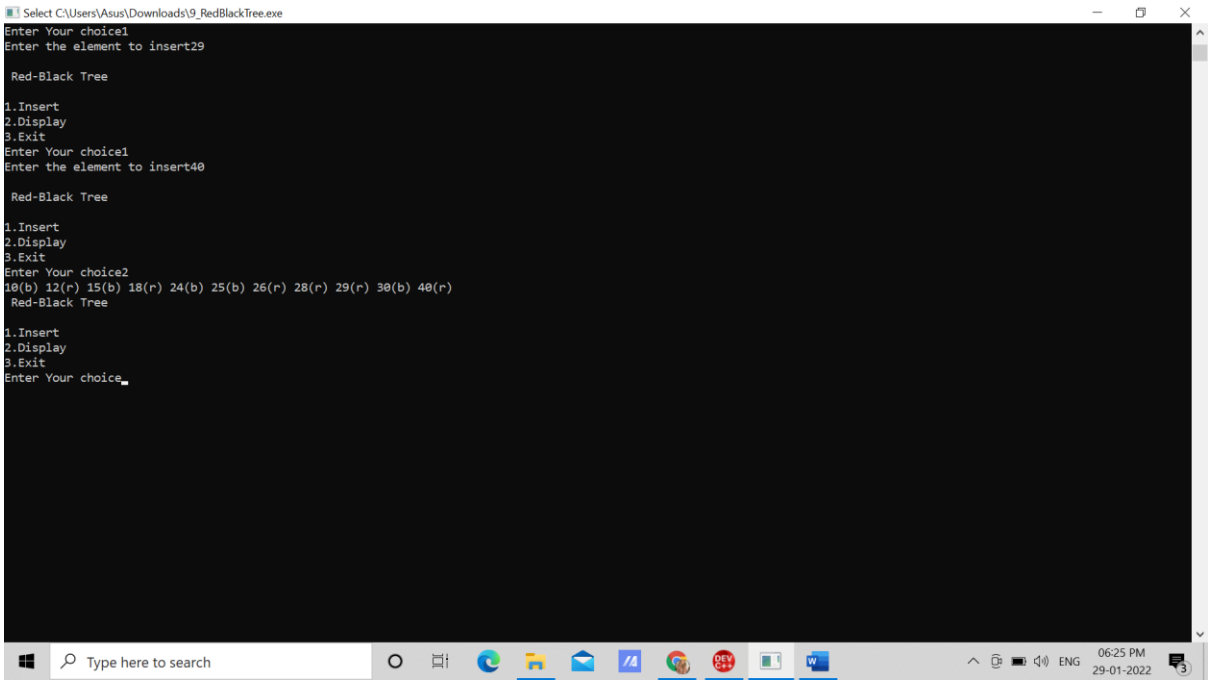
```
        if(pparent==ROOT)
                ROOT=right;
  }
}
void LLRotation(struct node *y) // function for Right Rotation
{
        struct node *p=findParent(y);
        struct node *x=y->left;
        struct node *T2= x->right;
        x->right=y;
        y->left=T2;
        if(p!=NULL)
                if(p->right==y)
                        p->right=x;
                else
                        p->left=x;
 }

void RRRotation(struct node *x) // function for left rotation
{
        struct node *p=findParent(x);
        struct node *y=x->right;
        struct node *T2=y->left;
        y->left=x;x->right=T2;
        if(p!=NULL)
                if(p->right==x)
                        p->right=y;
                else
                        p->left=y;
 }
```

```
void main()
{
	int opt;
	do
	{
		printf("\n Red-Black Tree\n");
		printf("\n1.Insert \n2.Display \n3.Exit\nEnter Your choice");
		scanf("%d",&opt);
		switch(opt)
		{
			case 1:
				insert();
				break;
			case 2:
				inorder(ROOT);
				break;
		}
	}
while(opt!=3);
}
```

Output:


C:\Users\Asus\Downloads\9_RedBlackTree.exe
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert24
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert10
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert28
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert15
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert25
Type here to search
ENG
06:24 PM
29-01-2022


Select C:\Users\Asus\Downloads\9_RedBlackTree.exe
Enter the element to insert25
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert30
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert12
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert26
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert18
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert29
Type here to search
ENG
06:25 PM
29-01-2022


Select C:\Users\Asus\Downloads\9_RedBlackTree.exe
Enter Your choice1
Enter the element to insert29
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice1
Enter the element to insert40
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice2
10(b) 12(r) 15(b) 18(r) 24(b) 25(b) 26(r) 28(r) 29(r) 30(b) 40(r)
Red-Black Tree
1.Insert
2.Display
3.Exit
Enter Your choice
Type here to search
ENG
06:25 PM
29-01-2022

Program

```
#include<stdio.h>
#include<stdlib.h>
struct edge
{
        int start;
        int end;
        int weight;

} *adj[20];
struct node
{
        int data;
        struct node *next;

} *first[20];
int n=0;
int find(int x)
{
        int flag=0,i;
        struct node* p;
        for(i=0;i<n;i++)
        {
                p=first[i];
                while(p!=NULL)
                {
                        if(p->data==x)
                        {
                                flag=1;
```

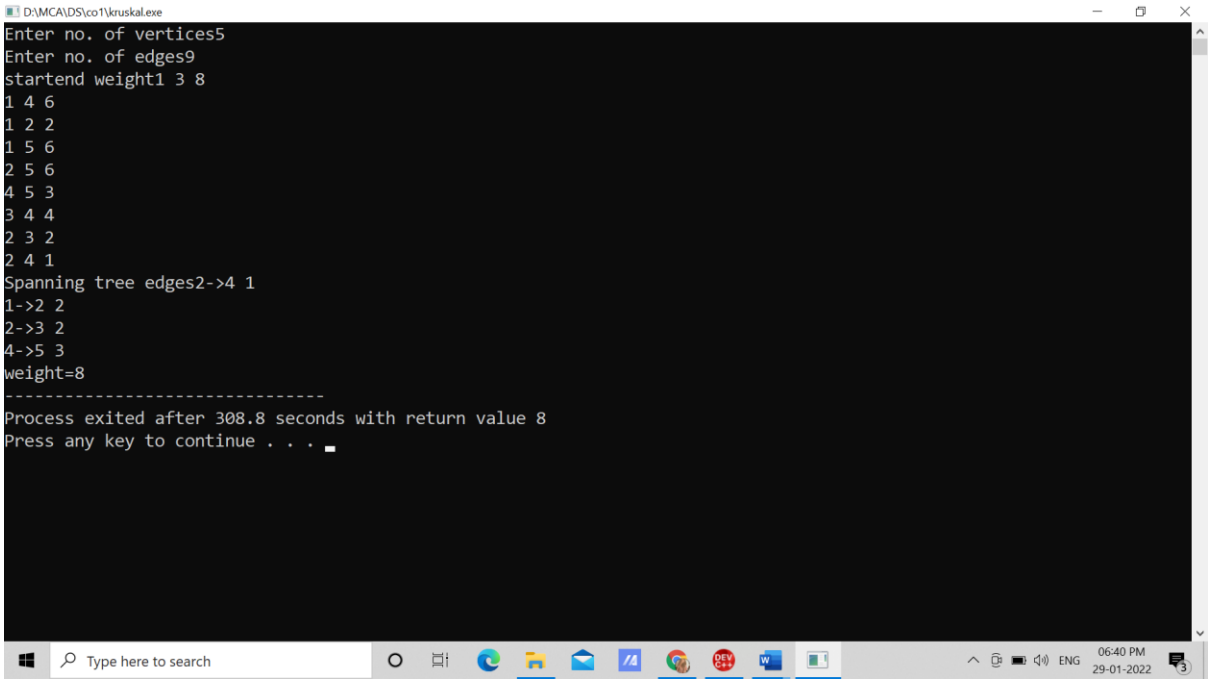
```
                        break;
                    }
                    p=p->next;
                }
                if(flag==1)
                    break;
            }
            if(flag==1)
                return(i);
            else
                return -1;
}
void Union(int x,int y)
{
            int i,j;
            struct node* p;
            i=find(x);
            j=find(y);
            if(i==j)
                printf("Both are in the same set");
            else
            {
                p=first[i];
                while(p->next != NULL)
                {
                    p=p->next;
                }
                p->next=first[j];
                first[j]=NULL;
            }
```

```
}

void makeset(int i)
{
	int pos;
	pos=find(i);
	if(pos==-1)
	{
		first[n]=(struct node*)malloc (sizeof(struct node));
		first[n]->data=i;
		first[n]->next=NULL;
		n++;
	}
	else
	{
		printf("\n the number exist in another set");
	}
}
void main()
{
	int s,en,w,i,k,c,count,e,u,v;
	struct edge A[20],adj[20];

	printf("Enter no. of vertices");
	scanf("%d",&v);
	for(i=1;i<=v;i++)
	{
		makeset(i);
	}
	printf("Enter no. of edges");
```

```
scanf("%d",&e);
printf("start");
printf("end weight");
c=-1;
for(i=1;i<=e;i++)
{
        scanf("%d%d%d",&s,&en,&w);
        for(k=c;k>=0;k--)
        {
                if(adj[k].weight>w)
                        adj[k+1]=adj[k];
                else
                        break;
        }
        adj[k+1].start=s;
        adj[k+1].end=en;
        adj[k+1].weight=w;
        c++;
}
count=0;
for(i=0;i<c;i++)
{
        u=adj[i].start;
        v=adj[i].end;
        if(find(u)!=find(v))
        {
                A[count].start=u;
                A[count].end=v;
                A[count].weight=adj[i].weight;
                count++;
```

```
                    Union(u,v);
                }
        }
        printf("Spanning tree edges");
        int sum=0;
        for(i=0;i<count;i++)
        {
                printf("%d->%d %d \n",A[i].start,A[i].end,A[i].weight);
                sum=sum+A[i].weight;
        }
        printf("weight=%d",sum);

}
```

Output:


D:\MCA\DS\co1\kruskal.exe
Enter no. of vertices5
Enter no. of edges9
startend weight1 3 8
1 4 6
1 2 2
1 5 6
2 5 6
4 5 3
3 4 4
2 3 2
2 4 1
Spanning tree edges2->4 1
1->2 2
2->3 2
4->5 3
weight=8

Process exited after 308.8 seconds with return value 8
Press any key to continue . . .
Type here to search
ENG
06:40 PM
29-01-2022

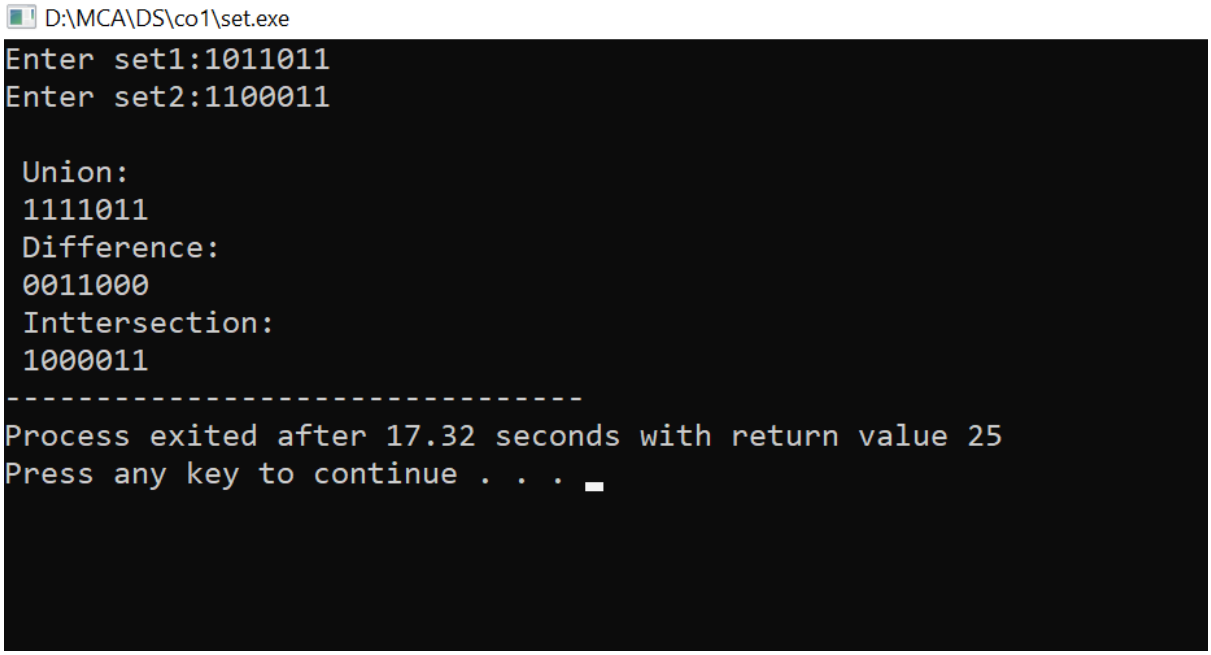
Program

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void setunion(char *s1,char *s2,char *s3)
{
	int i,l;
	l=strlen(s1);
	for(i=0;i<l;i++)
	{
		if(s1[i]=='0' && s2[i]=='0')
		{
			s3[i]=0;
		}
		else
		{
			s3[i]=1;
		}
	}
	s3[i]='\0';
}
void setintersection(char *s1,char *s2,char *s3)
{
	int l,i;
	l=strlen(s1);
	for(i=0;i<l;i++)
	{
		if(s1[i]=='1' && s2[i]=='1')
		{
```

```
                s3[i]=1;
            }
            else
            {
                s3[i]=0;
            }
        }
        s3[i]='\0';
}


void setdifference(char *s1,char *s2,char *s3)
{
        int i,l;
        l=strlen(s1);
        for(i=0;i<l;i++)
        {
            if(s1[i]=='1' && s2[i]=='0')
            {
                s3[i]=1;
            }
            else
            {
                s3[i]=0;
            }
        }
        s3[i]='\0';
}
```

```
void main()
{
        char s1[20],s2[20],s3[20];
        printf("Enter set1:");
        scanf("%s",s1);
        printf("Enter set2:");
        scanf("%s",s2);
        setunion(s1,s2,s3);
        printf("\n Union:\n %s",s3);
        setdifference(s1,s2,s3);
        printf("\n Difference:\n%s",s3);
        setintersection(s1,s2,s3);
        printf("\n Inttersection:\n%s",s3);

}
```

Output


D:\MCA\DS\co1\set.exe
Enter set1:1011011
Enter set2:1100011
Union:
1111011
Difference:
0011000
Inttersection:
1000011
Process exited after 17.32 seconds with return value 25
Press any key to continue . . .

Program

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
struct node
{
        int vertex;
        struct node  *next;
};
int v,e;
struct node **adj;
int que[30],visited[30];
int f=-1,r=-1;

void enq(int x)
{
        if (f==-1 && r==-1)
                f=0;
        r=(r+1)%v;
        que[r]=x;
}
int dequ()
{
        int data;
        data=que[f];
        if (f==r)
                f=r=-1;
        else
                f=(f+1)%v;
```

```
        return data;
}
void bfs()
{
        struct node *ptr;
        int ver,i,w;
        for(i=0;i<=v;i++)
                visited[i]=0;
        enq(1);
        visited[1]=1;
        printf("%d ",1);
        while(!(f==-1))
        {
                ver=dequ();
                ptr=adj[ver];
                while(ptr!=NULL)
                {
                        w=ptr->vertex;
                        if (visited[w]==0)
                        {
                                enq(w);
                                printf("%d ",w);
                                visited[w]=1;
                        }
                        ptr=ptr->next;
                }
        }
    }
```

```
void main()
{
	int s,i,en;
	struct node *ne;
	printf("Enter No: of vertices");
	scanf("%d",&v);
	adj= (struct node **)malloc((v+1)*sizeof(struct node *));
	for(i=0;i<=v;i++)
		adj[i]=NULL;
	printf("enter No: of Edjes");
	scanf("%d",&e);
	printf("Enter the edges\n");
	printf("start End\n");
	for(i=0;i<e;i++)
	{
		scanf("%d%d",&s,&en);
		ne=(struct node*)malloc(sizeof(struct node));
		ne->vertex=en;
		ne->next=adj[s];
		adj[s]= ne;
	}
	printf("\nbfs\n");
	bfs();
	getch();
}
```

Output

```
Enter No: of vertices8
enter No: of Edjes8
Enter the edges
start End
1 2
1 3
1 4
2 7
3 6
4 5
7 8
6 8

bfs
1 4 3 2 5 6 7 8
```

Program

```
#include<stdlib.h>
#include<stdio.h>
struct node
{
	int data;
	struct node *next;
};
struct node *first[20];
void makeset();
void unionset();
int find(int);
void display();
int n=0;
void main()
{
	int opt,x,i;
	do
	{
	printf("\nMenu");
	printf("\n1.Makeset\n2.Union\n3.Find\n4.Display\n5.Exit");
	printf("\nSelect the option : ");
	scanf("%d",&opt);
	switch(opt)
	{
		case 1:
			makeset();
			break;
		case 2:
```

```
                unionset();
                break;
        case 3:
                printf("Enter the value for x : ");
                scanf("%d",&x);
                i=find(x);
                if(i==-1)
                        printf("Element not found");
                else
                        printf("Element = %d ",first[i]->data);
                break;
        case 4:
                display();
                break;
    }
}while(opt!=5);
}
void makeset()
{
    int x,pos;
    printf("\nEnter the element : ");
    scanf("%d",&x);
    pos=find(x);
    if (pos==-1)
    {
        first[n]=(struct node *)malloc(sizeof(struct node *));
        first[n]->data=x;
        first[n]->next=NULL;
        n++;
    }
```

```
        else
                printf("Element already exists");
}
int find(int x)
{
        int i,m,flag=0;
        struct node *p;
        for(i=0;i<n;i++)
        {
                p=first[i];
                while(p!=NULL)
                {
                if(p->data==x)
                {
                        flag=1;
                        break;
                }
                p=p->next;
                }
                if (flag==1)
                        break;
        }
        if(flag==1)
                return i;
        else
                return -1;
}
void unionset()
{
        int a,b,i,j;
```

```
        struct node *p;
        printf("\nEnter the first element : ");
        scanf("%d",&a);
        printf("\nEnter the second element : ");
        scanf("%d",&b);
        i=find(a);
        j=find(b);
        if (i==-1 || j ==-1)
        {
        printf("Element not found");
        return;
        }
        if (i==j)
        printf("Both are in the same set");
        else
        {
                p=first[i];
                while(p->next!=NULL)
                p=p->next;
                p->next=first[j];
                first[j]=NULL;
        }
}
void display()
{
        int i;
        struct node *p;
        for(i=0;i<n;i++)
        {
        p=first[i];
```

```
        if(p==NULL)
                continue;
        printf("{");
        while(p!=NULL)
        {
                printf("%d ",p->data);
                p=p->next;
        }
        printf("}\n");
        }
}
```

Output

```
Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 1

Enter the element :
1

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 1

Enter the element : 2

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 1

Enter the element : 3

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 1

Enter the element : 4
```

```
Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 1

Enter the element : 5

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 1

Enter the element : 6

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 4
{1 }
{2 }
{3 }
{4 }
{5 }
{6 }

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
```

```
4.Display
5.Exit
Select the option : 2

Enter the first element : 2

Enter the second element : 1

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 4
{2 1 }
{3 }
{4 }
{5 }
{6 }

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 2

Enter the first element : 5

Enter the second element : 6

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 4
{2 1 }
```

```
{2 1 }
{3 }
{4 }
{5 6 }

Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 3
Enter the value for x : 5
Element = 5
Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 3
Enter the value for x : 10
Element not found
Menu
1.Makeset
2.Union
3.Find
4.Display
5.Exit
Select the option : 5

--------------------------------
Process exited after 461.8 seconds with return value 5
Press any key to continue . . .
```