



ADITYA COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Affiliated to JNTUA, Anantapuramu & Approved by AICTE, New Delhi)

Valasapalle (P), Punganur Road, Madanapalle-517 325, Chittoor District, Andhra Pradesh

ADVANCED DATA STRUCTURES AND ALGORITHMS LABORATORY (20A05301P)

Branch -CSE, AI&DS

Year & Sem- II-I

OBJECTIVES AND OUTCOMES

Objectives: To make the student learn a object oriented way of solving problems. To make the student write ADA for all data structures.

Outcomes: At the end of the course the students are able to: For a given Search problem Linear Search and Binary Search student will able to implement it. For a given problem of Stacks, Queues and linked list student will able to implement it and analyze the same to determine the time and computation complexity. Student will able to write program for Selection Sort, Bubble Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort and compare their performance in term of Space and Time complexity.

Vision: To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission: To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers. Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement.

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS:

1. Intel based desktop PC of 166MHz or faster processor with at least 64 MB RAM and 100 MB free disk space.
2. Dev and turbo C++ compilers .

LIST OF EXPERIMENTS

1. Write a Program to implement Stack using Arrays.
2. Write a Program to implement Stack using Linked List.
3. Write a Program to implement Single Linked List.
4. Write a Program to implement Operations of Binary Search Tree.
5. Write a Program to implement Towers of Hanoi.
6. Write a Program to implement Splay Tree.
7. Write a Program to implement Quick Sort.
8. Write a Program to implement Merge Sort.
9. Write a Program to find solution for knapsack using Greedy's Method.
10. Write a Program to find minimum cost spanning tree using prim's algorithm.
11. Write a Program to find minimum cost spanning tree using kruskal's algorithm.
12. Write a Program to find Single source shortest path for given graph.
13. Write a Program to find Solution for Job Sequence with deadlines problem.
14. Write a Program to find Solution of 0/1 knapsack problem using Dynamic Programming.
15. Write a Program to implement N- Queen's problem using back tracking.
16. Write a Program to perform Binary Search for given set of integer values.

USEFUL TEXT BOOKS / REFERENCES :

1. Y Daniel Liang, "Introduction to Programming using Python", Pearson.
2. Benjamin Baka, David Julian, "Python Data Structures and Algorithms", Packt Publishers, 2017.
3. Rance D. Necaise, "Data Structures and Algorithms using Python", Wiley Student Edition.

EXPERIMENTS

1.AIM OF THE EXPERIMENT: To Write a program to implement stack using Arrays.

DESCRIPTION : Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). In programming terms, putting an item on top of the stack is called push and removing an item is called pop.

SOURCE CODE:

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
//clrscr();
top=-1;
printf("\n Enter the size of STACK[MAX=100]:");
scanf("%d",&n);
printf("\n\t STACK OPERATIONS USING ARRAY");
printf("\n\t-----");
printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
do
{
printf("\n Enter the Choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
{
push();
```

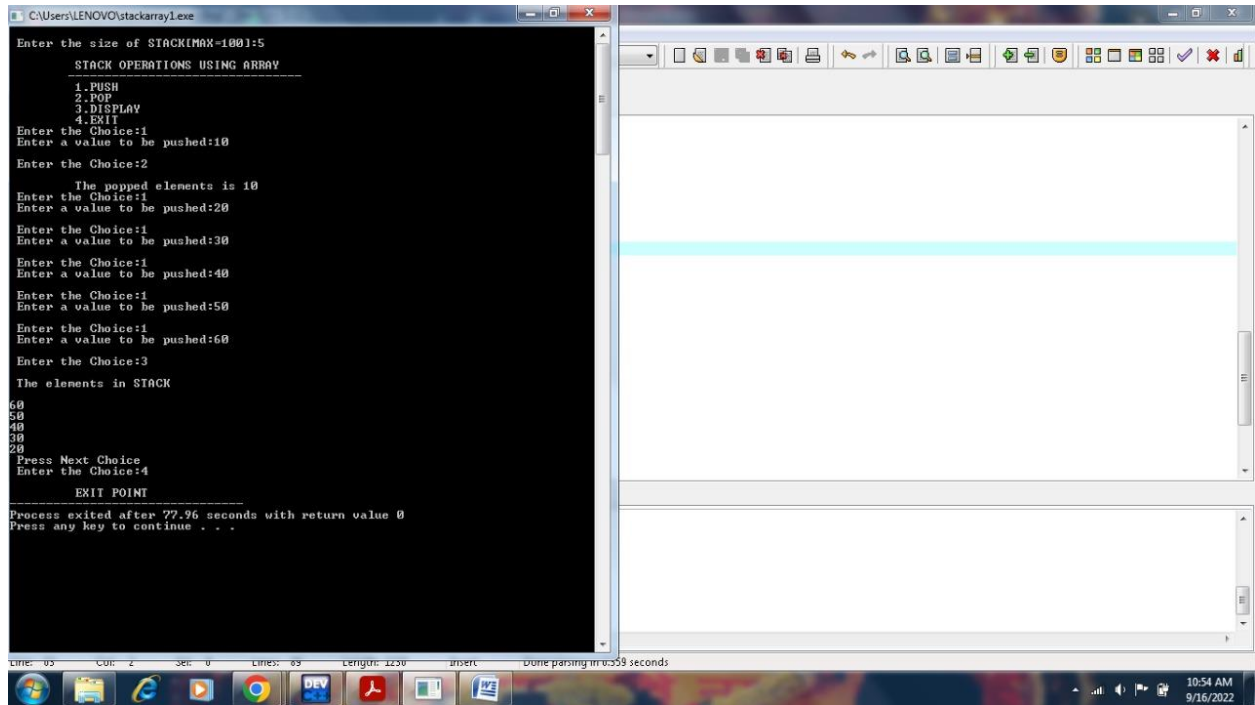
```
break;
}
case 2:
{
pop();
break;
}
case 3:
{
display();
break;
}
case 4:
{
printf("\n\t EXIT POINT ");
break;
}
default:
{
printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
}
}
}
while(choice!=4);
return 0;
}

void push()
{
if(top>=n-1)
{
printf("\n\tSTACK is over flow");
```

```
}  
else  
{  
printf(" Enter a value to be pushed:");  
scanf("%d",&x);  
top++;  
stack[top]=x;  
}  
}  
void pop()  
{  
if(top<=-1)  
{  
printf("\n\t Stack is under flow");  
}  
else  
{  
printf("\n\t The popped elements is %d",stack[top]);  
top--;  
}  
}  
void display()  
{  
if(top>=0)  
{  
printf("\n The elements in STACK \n");  
for(i=top; i>=0; i--)  
printf("\n%d",stack[i]);  
printf("\n Press Next Choice");  
}  
else
```

```
{  
printf("\n The STACK is empty");  
}  
}
```

OUTPUT :



```
C:\Users\LENOVO\stackarray1.exe  
Enter the size of STACK(MAX=1001):5  
STACK OPERATIONS USING ARRAY  
1.PUSH  
2.POP  
3.DISPLAY  
4.EXIT  
Enter the Choice:1  
Enter a value to be pushed:10  
Enter the Choice:2  
The popped elements is 10  
Enter the Choice:1  
Enter a value to be pushed:20  
Enter the Choice:1  
Enter a value to be pushed:30  
Enter the Choice:1  
Enter a value to be pushed:40  
Enter the Choice:1  
Enter a value to be pushed:50  
Enter the Choice:1  
Enter a value to be pushed:60  
Enter the Choice:3  
The elements in STACK  
60  
50  
40  
30  
20  
Press Next Choice  
Enter the Choice:4  
EXIT POINT  
Process exited after 77.96 seconds with return value 0  
Press any key to continue . . .  
time: 02  loc: 2  sel: 0  time: 05  Length: 2230  prntc  Done parsing in 0.359 seconds  
10:54 AM 9/16/2022
```

RESULT: Hence The above Program stack using arrays is executed successfully.

2.AIM OF THE EXPERIMENT: To Write a program to implement Stack using Linked Lists.

DESCRIPTION : In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its previous node in the list. The next field of the first element must be always NULL.

SOURCE CODE:

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

```
void push();
void pop();
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;
main ( )
{
int choice=0;
printf("\n*****Stack operations using linked list*****\n");
printf("\n-----\n");
while(choice != 4)
{
printf("\n\nChose one from the below options...\n");
printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
printf("\n Enter your choice \n");
scanf("%d",&choice);
switch(choice)
{
case 1:
{
push();
break;
}
case 2:
{
```

```
pop();
break;
}
case 3:
{
display();
break;
}
case 4:
{
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
```

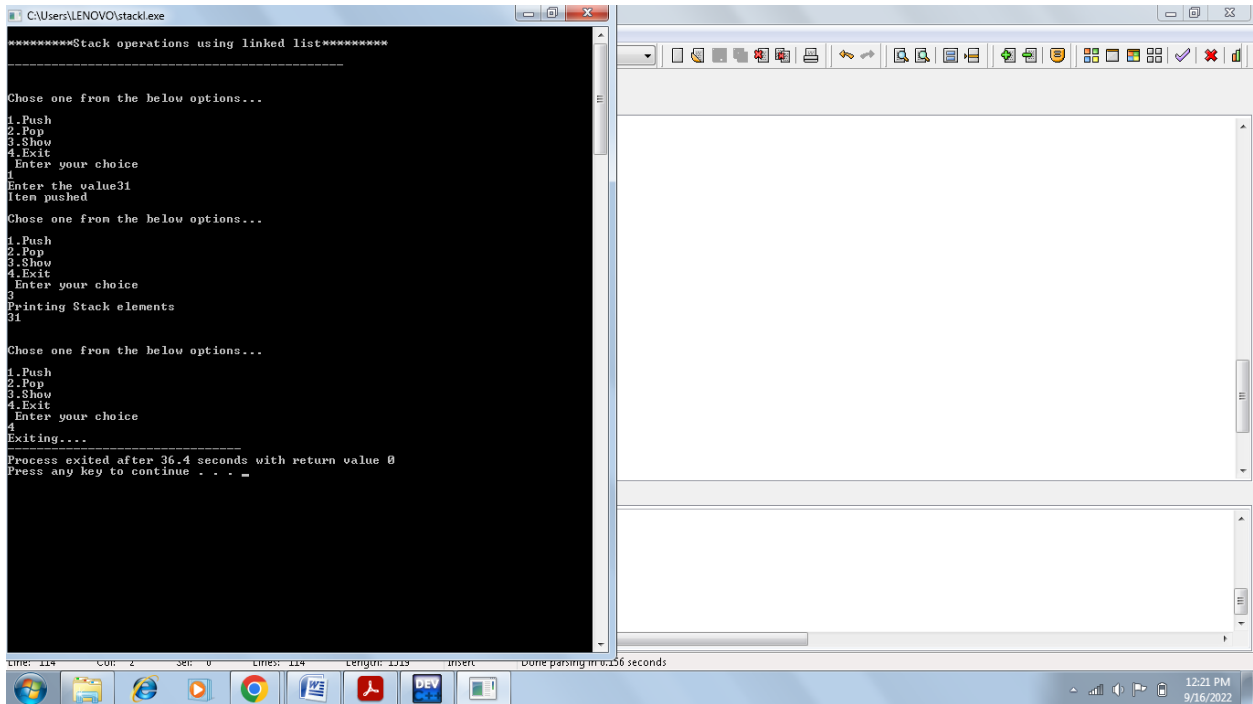


```
{
printf("Enter the value");
scanf("%d",&val);
if(head==NULL)
{
ptr->val = val;
ptr -> next = NULL;
head=ptr;
}
else
{
ptr->val = val;
ptr->next = head;
head=ptr;
}
printf("Item pushed");
}
}

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 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;
}
}
}
```

OUTPUT:



```
*****Stack operations using linked list*****

Chose one from the below options...
1. Push
2. Pop
3. Show
4. Exit
Enter your choice
1
Enter the value31
Item pushed

Chose one from the below options...
1. Push
2. Pop
3. Show
4. Exit
Enter your choice
3
Printing Stack elements
31

Chose one from the below options...
1. Push
2. Pop
3. Show
4. Exit
Enter your choice
4
Exiting...

Process exited after 36.4 seconds with return value 0
Press any key to continue . . .
```

RESULT: Hence The above Program stack using Linked List is executed successfully.

3 AIM OF THE EXPERIMENT: To Write a program to implement Single Linked List.

DESCRIPTION : A singly linked list is a type of linked list that is unidirectional, that is, it can be traversed in only one direction from head to the last node (tail). Each element in a linked list is called a node. A single node contains data and a pointer to the *next* node which helps in maintaining the structure of the list. The first node is called the head it points to the first node of the list and helps us access every other element in the list. The last node, also sometimes called the tail, points to NULL which helps us in determining when the list ends

SOURCE CODE:

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

struct node
```

```
{
int data;
struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
main ()
{
int choice =0;
while(choice != 9)
{
printf("\n\n*****Main Menu*****\n");
printf("\nChoose one option from the following list ...\n");
printf("\n===== \n");
printf("\n1.Insert in begining");
printf("\n2.Insert at last");
printf("\n3.Insert at any random location");
printf("\n4.Delete from Beginning");
printf("\n5.Delete from last");
printf("\n6.Delete node after specified location");
printf("\n7.Search for an element");
printf("\n8.Show");
```

```
printf("\n9.Exit");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
{
case 1:
begininsert();
break;
case 2:
lastinsert();
break;
case 3:
randominsert();
break;
case 4:
begin_delete();
break;
case 5:
last_delete();
break;
case 6:
random_delete();
break;
case 7:
search();
break;
case 8:
display();
break;
```

```
case 9:
exit(0);
break;
default:
printf("Please enter valid choice..");
}
}
}
void beginsert()
{
struct node *ptr;
int item;
ptr = (struct node *) malloc(sizeof(struct node *));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
else
{
printf("\nEnter value\n");
scanf("%d",&item);
ptr->data = item;
ptr->next = head;
head = ptr;
printf("\nNode inserted");
}
}
void lastinsert()
{
```

```
struct node *ptr,*temp;
int item;
ptr = (struct node*)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
else
{
printf("\nEnter value?\n");
scanf("%d",&item);
ptr->data = item;
if(head == NULL)
{
ptr -> next = NULL;
head = ptr;
printf("\nNode inserted");
}
else
{
temp = head;
while (temp -> next != NULL)
{
temp = temp -> next;
}
temp->next = ptr;
ptr->next = NULL;
printf("\nNode inserted");
}
```

```
}  
}  
void randominsert()  
{  
    int i,loc,item;  
    struct node *ptr, *temp;  
    ptr = (struct node *) malloc (sizeof(struct node));  
    if(ptr == NULL)  
    {  
        printf("\nOVERFLOW");  
    }  
    else  
    {  
        printf("\nEnter element value");  
        scanf("%d",&item);  
        ptr->data = item;  
        printf("\nEnter the location after which you want to insert ");  
        scanf("\n%d",&loc);  
        temp=head;  
        for(i=0;i<loc;i++)  
        {  
            temp = temp->next;  
            if(temp == NULL)  
            {  
                printf("\ncan't insert\n");  
                return;  
            }  
        }  
        ptr ->next = temp ->next;
```



```
temp ->next = ptr;
printf("\nNode inserted");
}
}
void begin_delete()
{
struct node *ptr;
if(head == NULL)
{
printf("\nList is empty\n");
}
else
{
ptr = head;
head = ptr->next;
free(ptr);
printf("\nNode deleted from the beginning ...\n");
}
}
void last_delete()
{
struct node *ptr,*ptr1;
if(head == NULL)
{
printf("\nlist is empty");
}
else if(head -> next == NULL)
{
head = NULL;
```

```
free(head);
printf("\nOnly node of the list deleted ...\n");
}
else
{
ptr = head;
while(ptr->next != NULL)
{
ptr1 = ptr;
ptr = ptr ->next;
}
ptr1->next = NULL;
free(ptr);
printf("\nDeleted Node from the last ...\n");
}
}

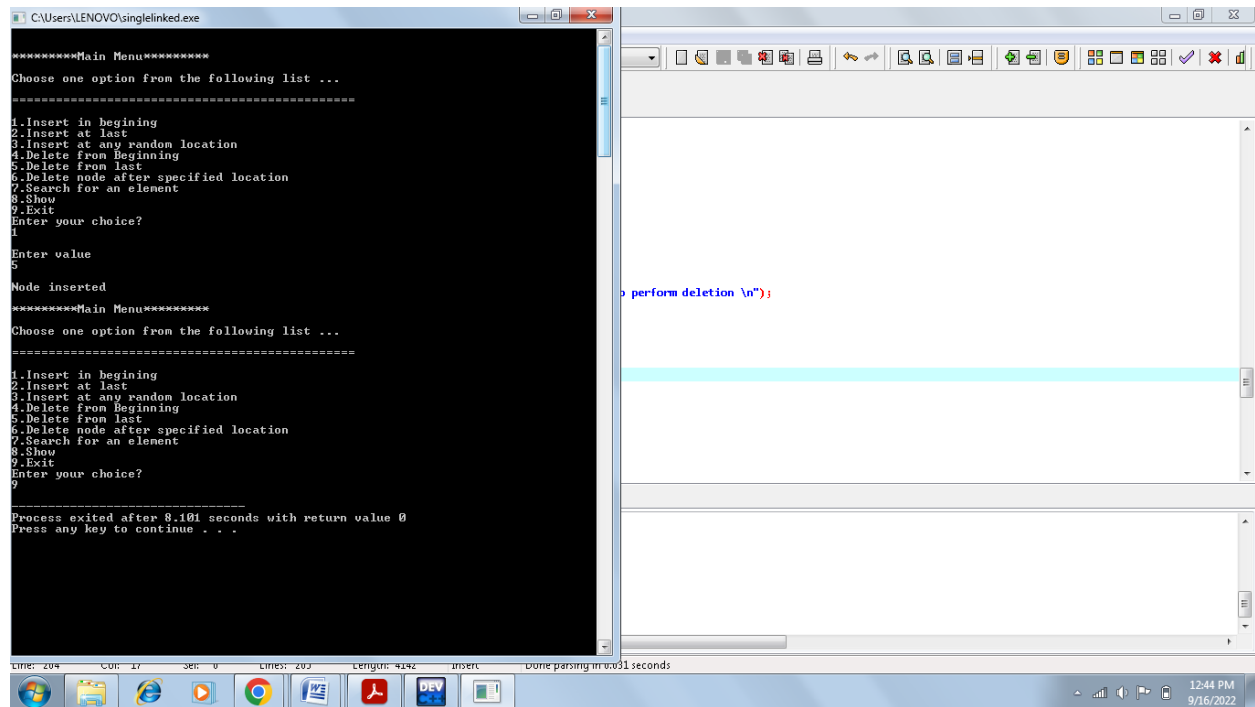
void random_delete()
{
struct node *ptr,*ptr1;
int loc,i;
printf("\n Enter the location of the node after which you want to perform deletion \n");
scanf("%d",&loc);
ptr=head;
for(i=0;i<loc;i++)
{
ptr1 = ptr;
ptr = ptr->next;
if(ptr == NULL)
{
```

```
printf("\nCan't delete");
return;
}
}
ptr1 ->next = ptr ->next;
free(ptr);
printf("\nDeleted node %d ",loc+1);
}
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->data == item)
{
printf("item found at location %d ",i+1);
flag=0;
}
else
```

```
{
flag=1;
}
i++;
ptr = ptr -> next;
}
if(flag==1)
{
printf("Item not found\n");
}
}
}
void display()
{
struct node *ptr;
ptr = head;
if(ptr == NULL)
{
printf("Nothing to print");
}
else
{
printf("\nprinting values . . . . \n");
while (ptr!=NULL)
{
printf("\n%d",ptr->data);
ptr = ptr -> next;
}
}
```

```
}
```

OUTPUT:



```
C:\Users\LENOVO\singlelinked.exe

*****Main Menu*****
Choose one option from the following list ...
=====
1.Insert in beginning
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit
Enter your choice?
1
Enter value
5
Node inserted
*****Main Menu*****
Choose one option from the following list ...
=====
1.Insert in beginning
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit
Enter your choice?
9
-----
Process exited after 8.101 seconds with return value 0
Press any key to continue . . .
```

RESULT: Hence The above Program Single Linked List is executed successfully.

4 AIM OF THE EXPERIMENT: To Write a program to implement Operations of Binary Search Tree

DESCRIPTION : Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree

SOURCE CODE:

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
struct node
```

```
{
int info;
struct node * lchild;
struct node * rchild;
}*root; ///////

void find(int item, struct node **par, struct node **loc)
{
struct node *ptr, *ptrsave;
if (root == NULL) ///////
{
*loc = NULL;
*par = NULL;
return;
} ///////
if (item == root->info)
{
*loc = root;
*par = NULL;
return;
} ///////
if (item < root->info)
ptr = root->lchild;
else
ptr = root->rchild;
ptrsave = root; ///////
while (ptr != NULL)
{
if (item == ptr->info)
{ *loc = ptr;
```

```
*par = ptrsave;
return;
} /////

ptrsave = ptr;
if (item < ptr->info)
ptr = ptr->lchild;
else
ptr = ptr->rchild;
} /////

*loc = NULL;
*par = ptrsave;
}
/////

void insert(int item)
{
struct node *tmp, *parent, *location;
find(item, &parent, &location);
if (location != NULL)
{
printf("Item already present");
return;
} /////

tmp = (struct node *) malloc(sizeof(struct node));
tmp->info = item;
tmp->lchild = NULL;
tmp->rchild = NULL;
/////

if (parent == NULL)
root = tmp;
```

```
else
if (item < parent->info)
parent->lchild = tmp;
else
parent->rchild = tmp;
} //////
void case_a(struct node *par, struct node *loc)
{
if (par == NULL)
root = NULL;
else
if (loc == par->lchild)
par->lchild = NULL;
else
par->rchild = NULL;
}
void case_b(struct node *par, struct node *loc) //////
{
struct node * child;
//intializing child
if (loc->lchild != NULL)
child = loc->lchild;
else
child = loc->rchild;
if (par == NULL)
root = child;
else
if (loc == par->lchild)
par->lchild = child;
```



```
else
par->rchild = child; ///////
}
void case_c(struct node *par, struct node *loc)
{
struct node *ptr, *ptrsave, *suc, *parsuc;
ptrsave = loc;
ptr = loc->rchild;
while (ptr->lchild != NULL)
{
ptrsave = ptr;
ptr = ptr->lchild;
} ///////
suc = ptr;
parsuc = ptrsave;
if (suc->lchild == NULL && suc->rchild == NULL)
case_a(parsuc, suc);
else
case_b(parsuc, suc);
if (par == NULL) ///////
root = suc;
else
if (loc == par->lchild)
par->lchild = suc;
else
par->rchild = suc;
suc->lchild = loc->lchild;
suc->rchild = loc->rchild;
} ///////
```

```
int del(int item)
{
    struct node *parent, *location;
    if (root == NULL)
    {
        printf("Tree empty");
        return 0;
    }
    find(item, &parent, &location);
    if (location == NULL)
    {
        printf("Item not present in tree");
        return 0;
    } ///////
    if (location->lchild == NULL && location->rchild == NULL)
        case_a(parent, location);
    if (location->lchild != NULL && location->rchild == NULL)
        case_b(parent, location);
    if (location->lchild == NULL && location->rchild != NULL)
        case_b(parent, location);
    if (location->lchild != NULL && location->rchild != NULL)
        case_c(parent, location);
    free(location);
}
/////

int preorder(struct node *ptr)
{
    if (root == NULL)
    {
```

```
printf("Tree is empty");
return 0;
}
if (ptr != NULL)
{
printf("%d ", ptr->info);
preorder(ptr->lchild);
preorder(ptr->rchild);
} //////////
}

void inorder(struct node *ptr)
{
if (root == NULL)
{
printf("Tree is empty");
return;
} //////////
if (ptr != NULL)
{
inorder(ptr->lchild);
printf("%d ", ptr->info);
inorder(ptr->rchild);
} //////////////////////
}

void postorder(struct node *ptr)
{
if (root == NULL)
{
printf("Tree is empty");
```

```
return;
} ////////////
if (ptr != NULL)
{
postorder(ptr->lchild);
postorder(ptr->rchild);
printf("%d ", ptr->info);
} ////////////
}

void display(struct node *ptr, int level)
{
int i;
if (ptr != NULL)
{
display(ptr->rchild, level + 1);
printf("\n");
for (i = 0; i < level; i++)
printf(" ");
printf("%d", ptr->info);
display(ptr->lchild, level + 1);
}
} ////////////

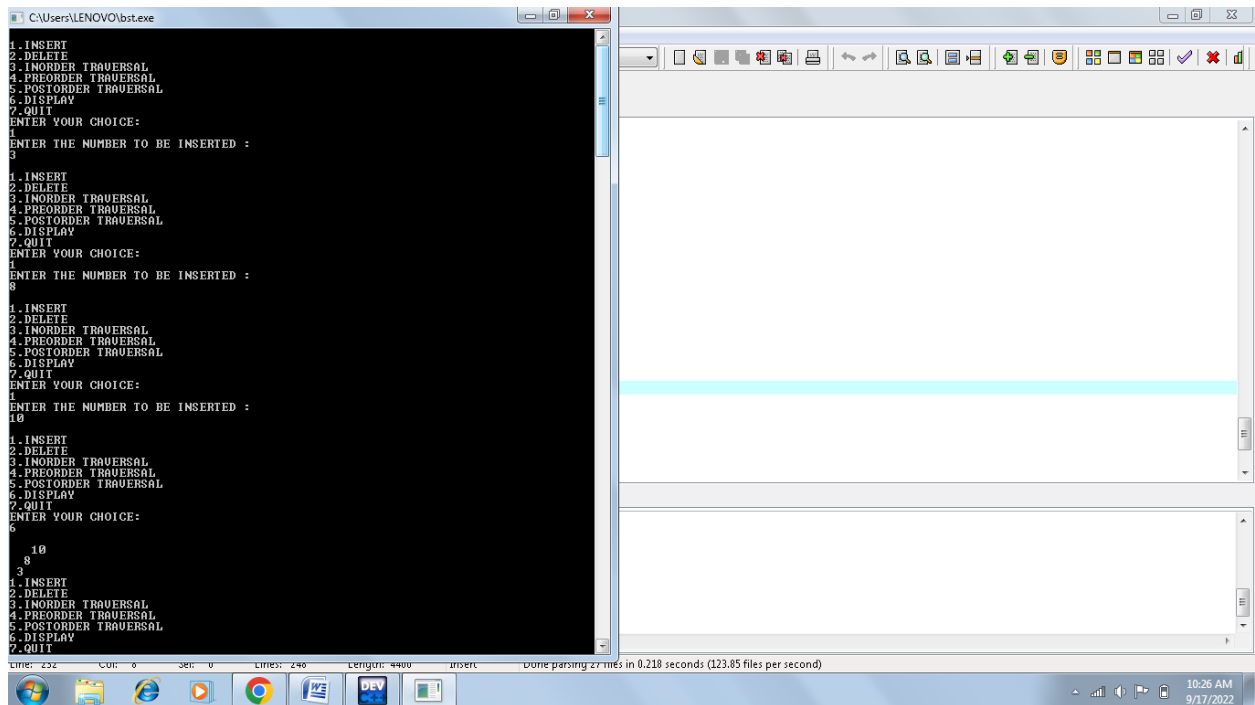
int main()
{
int choice, num;
root = NULL;
while (1)
{
printf("\n");
```

```
printf("1.INSERT\n");
printf("2.DELETE\n");
printf("3.INORDER TRAVERSAL\n");
printf("4.PREORDER TRAVERSAL\n");
printf("5.POSTORDER TRAVERSAL\n");
printf("6.DISPLAY\n");
printf("7.QUIT\n"); ////////////
printf("ENTER YOUR CHOICE: \n");
scanf("%d", &choice);
switch (choice)
{
case 1:
printf("ENTER THE NUMBER TO BE INSERTED : \n");
scanf("%d", &num);
insert(num);
break; ////////////
case 2:
printf("ENTER THE NUMBER TO BE DELETED : \n");
scanf("%d", &num);
del(num);
break;
case 3:
inorder(root);
break;
case 4:
preorder(root);
break;
case 5:
postorder(root);
```

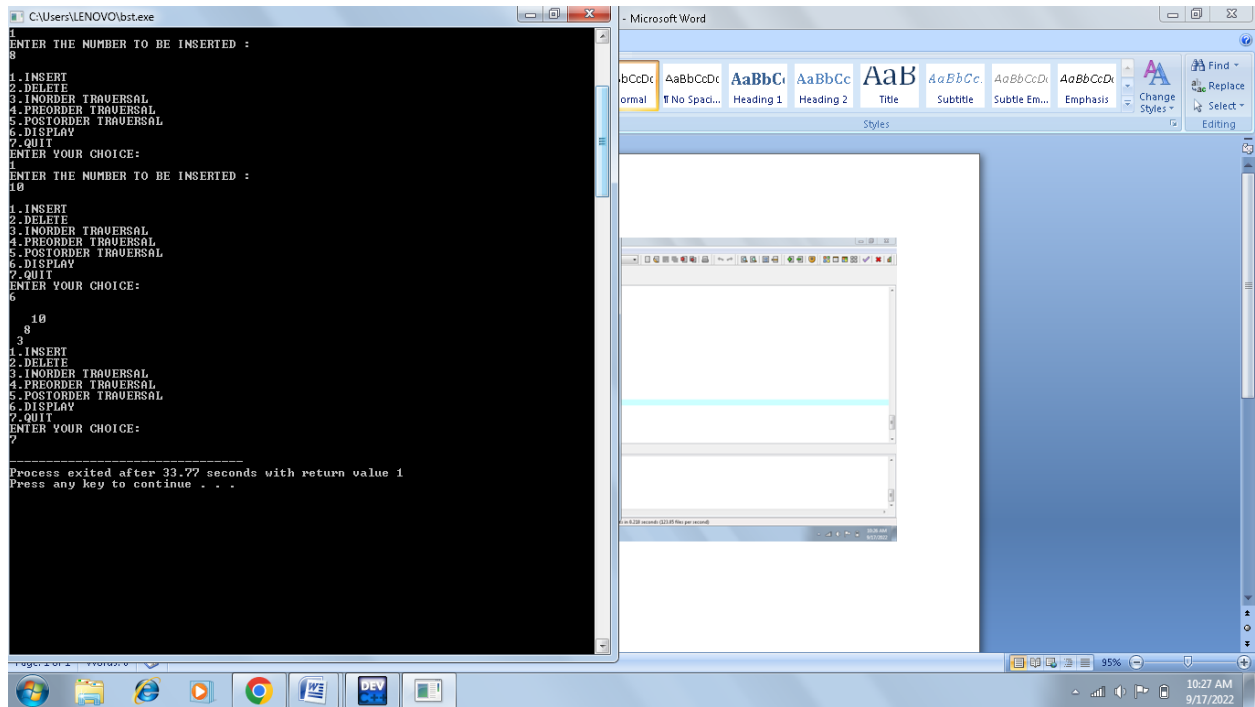
```
break; ///////////////////////////////////

case 6:
display(root, 1);
break;
case 7:
exit(1);
break;
default:
printf("Wrong choice\n");
}
}
}
```

OUTPUT:



```
C:\Users\LENOVO\bst.exe
1. INSERT
2. DELETE
3. INORDER TRAVERSAL
4. PREORDER TRAVERSAL
5. POSTORDER TRAVERSAL
6. DISPLAY
7. QUIT
ENTER YOUR CHOICE:
1
ENTER THE NUMBER TO BE INSERTED :
3
1. INSERT
2. DELETE
3. INORDER TRAVERSAL
4. PREORDER TRAVERSAL
5. POSTORDER TRAVERSAL
6. DISPLAY
7. QUIT
ENTER YOUR CHOICE:
1
ENTER THE NUMBER TO BE INSERTED :
8
1. INSERT
2. DELETE
3. INORDER TRAVERSAL
4. PREORDER TRAVERSAL
5. POSTORDER TRAVERSAL
6. DISPLAY
7. QUIT
ENTER YOUR CHOICE:
1
ENTER THE NUMBER TO BE INSERTED :
10
1. INSERT
2. DELETE
3. INORDER TRAVERSAL
4. PREORDER TRAVERSAL
5. POSTORDER TRAVERSAL
6. DISPLAY
7. QUIT
ENTER YOUR CHOICE:
6
10
8
3
1. INSERT
2. DELETE
3. INORDER TRAVERSAL
4. PREORDER TRAVERSAL
5. POSTORDER TRAVERSAL
6. DISPLAY
7. QUIT
ENTER YOUR CHOICE:
1
ENTER THE NUMBER TO BE INSERTED :
3
1. INSERT
2. DELETE
3. INORDER TRAVERSAL
4. PREORDER TRAVERSAL
5. POSTORDER TRAVERSAL
6. DISPLAY
7. QUIT
ENTER YOUR CHOICE:
6
10
8
3
```



RESULT: Hence The above Program Binary Search Tree is executed successfully.

5 AIM OF THE EXPERIMENT: To Write a program to implement Towers of Hanoi

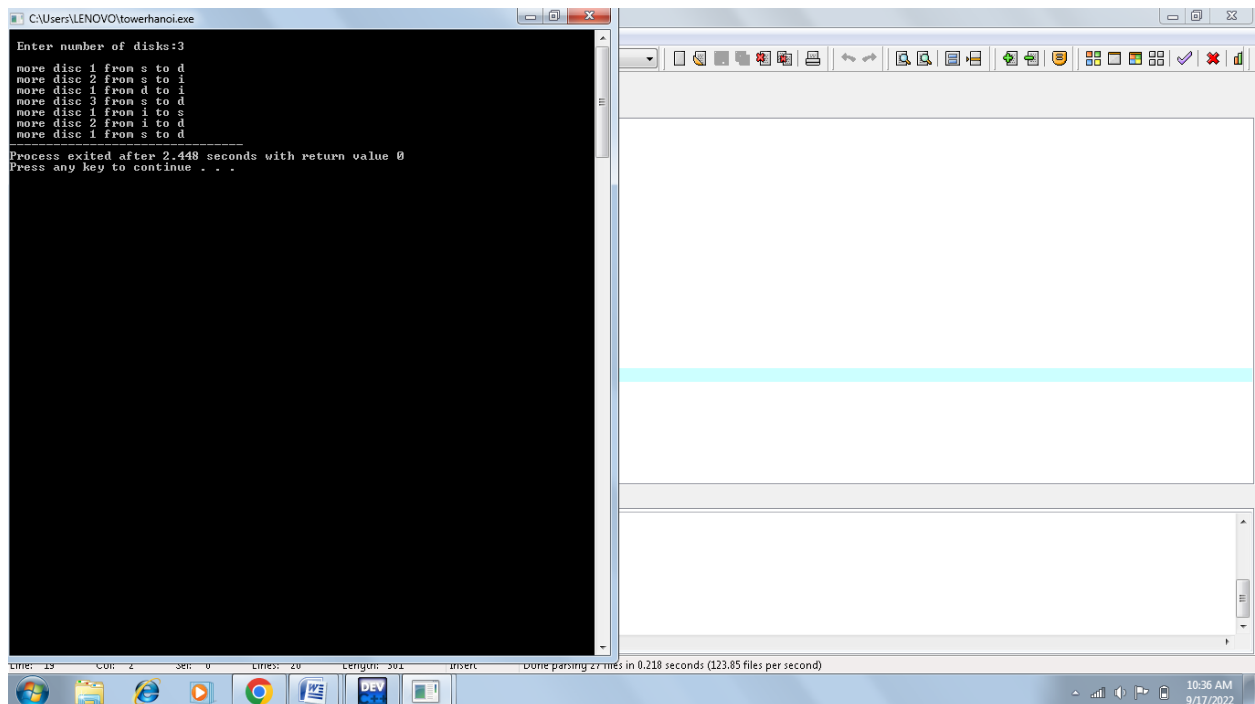
DESCRIPTION : Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the rules: 1) Only one disk can be moved at a time. 2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack. 3) No disk may be placed on top of a smaller disk.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
void transfer(int,char,char,char);
int main()
{
    int n;
```

```
printf("\n Enter number of disks:");  
  
scanf("%d",&n);  
  
transfer(n,'s','i','d');  
  
}  
  
void transfer(int n, char s, char i, char d)  
{  
    if(n>0)  
    {  
        transfer(n-1,s,d,i);  
  
        printf("\n more disc %d from %c to %c", n,s,d);  
  
        transfer(n-1,i,s,d);  
    }  
}
```

OUTPUT:



```
C:\Users\LENOVO\towerhanoi.exe  
Enter number of disks:3  
more disc 1 from s to d  
more disc 2 from s to i  
more disc 1 from d to i  
more disc 3 from s to d  
more disc 1 from i to s  
more disc 2 from i to d  
more disc 1 from s to d  
-----  
Process exited after 2.448 seconds with return value 0  
Press any key to continue . . .
```

RESULT: Hence The above Program Towers of Hanoi is executed successfully.

6 AIM OF THE EXPERIMENT: To Write a program to implement Splay Tree.

DESCRIPTION : Splay tree is also self-balancing BST. The main idea of splay tree is to bring the recently accessed item to root of the tree, this makes the recently searched item to be accessible in $O(1)$ time if accessed again. The idea is to use locality of reference.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
// An splay tree node
struct node
{
    int key;
    struct node *left, *right;
};
/* Helper function that allocates a new node with the given key and
   NULL left and right pointers. */
struct node* newNode(int key) {
    struct node* node = (struct node*) malloc(sizeof(struct node));
    node->key = key;
    node->left = node->right = NULL;
    return (node);
}
// A utility function to right rotate subtree rooted with y
// See the diagram given above.
struct node *rightRotate(struct node *x) {
    struct node *y = x->left;
    x->left = y->right;
    y->right = x;
    return y;
}
```

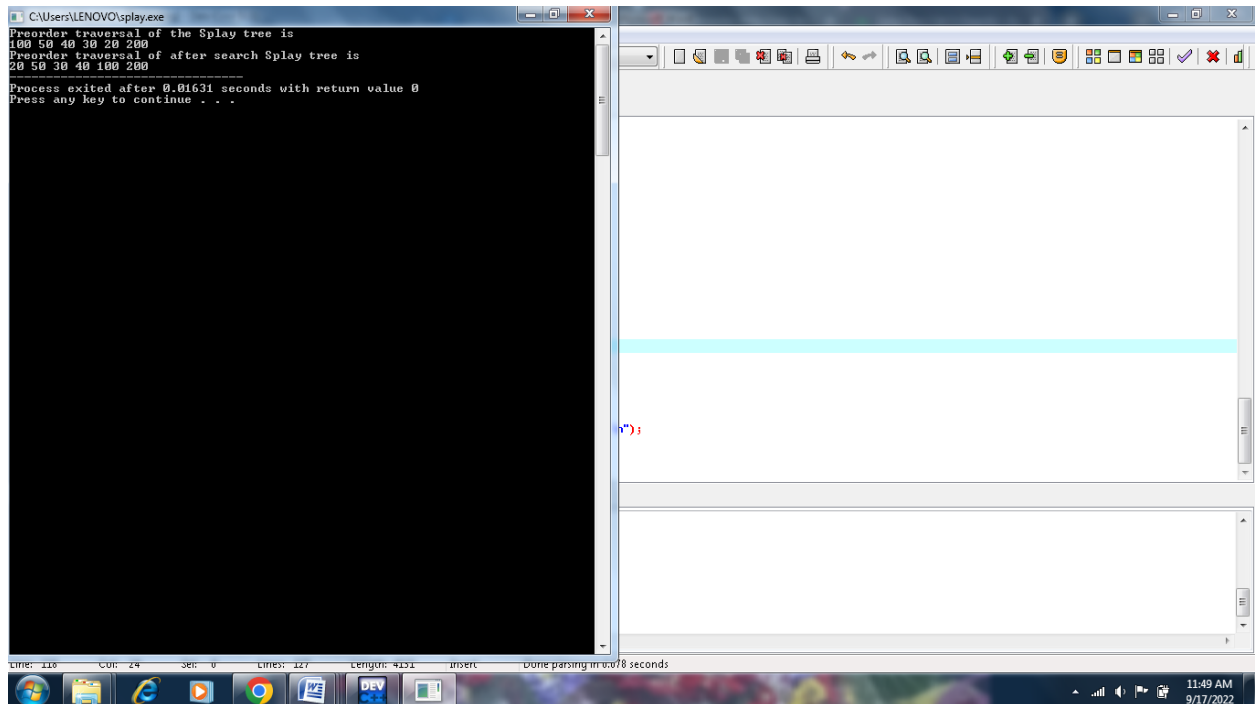
```
// A utility function to left rotate subtree rooted with x
// See the diagram given above.
struct node *leftRotate(struct node *x) {
    struct node *y = x->right;
    x->right = y->left;
    y->left = x;
    return y;
}
// This function brings the key at root if key is present in tree.
// If key is not present, then it brings the last accessed item at
// root. This function modifies the tree and returns the new root
struct node *splay(struct node *root, int key) {
    // Base cases: root is NULL or key is present at root
    if (root == NULL || root->key == key)
        return root;
    // Key lies in left subtree
    if (root->key > key) {
        // Key is not in tree, we are done
        if (root->left == NULL)
            return root;
        // Zig-Zig (Left Left)
        if (root->left->key > key) {
            // First recursively bring the key as root of left-left
            root->left->left = splay(root->left->left, key);
            // Do first rotation for root, second rotation is done after else
            root = rightRotate(root);
        } else if (root->left->key < key) // Zig-Zag (Left Right)
        {
            // First recursively bring the key as root of left-right
```

```
    root->left->right = splay(root->left->right, key);
    // Do first rotation for root->left
    if (root->left->right != NULL)
        root->left = leftRotate(root->left);
}
// Do second rotation for root
return (root->left == NULL) ? root : rightRotate(root);
} else // Key lies in right subtree
{
    // Key is not in tree, we are done
    if (root->right == NULL)
        return root;
    // Zag-Zig (Right Left)
    if (root->right->key > key) {
        // Bring the key as root of right-left
        root->right->left = splay(root->right->left, key);
        // Do first rotation for root->right
        if (root->right->left != NULL)
            root->right = rightRotate(root->right);
    } else if (root->right->key < key) // Zag-Zag (Right Right)
    {
        // Bring the key as root of right-right and do first rotation
        root->right->right = splay(root->right->right, key);
        root = leftRotate(root);
    }
    // Do second rotation for root
    return (root->right == NULL) ? root : leftRotate(root);
}
}
```

```
// The search function for Splay tree. Note that this function
// returns the new root of Splay Tree. If key is present in tree
// then, it is moved to root.
struct node *search(struct node *root, int key) {
    return splay(root, key);
}
// A utility function to print preorder traversal of the tree.
// The function also prints height of every node
void preOrder(struct node *root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}
int main()
{
    struct node *root = newNode(100);
    root->left = newNode(50);
    root->right = newNode(200);
    root->left->left = newNode(40);
    root->left->left->left = newNode(30);
    root->left->left->left->left = newNode(20);
    printf("Preorder traversal of the Splay tree is \n");
    preOrder(root);
    root = search(root, 20);
    printf("\nPreorder traversal of after search Splay tree is \n");
    preOrder(root);
    return 0;
}
```

}

OUTPUT:



```
Preorder traversal of the Splay tree is
100 50 40 30 20 200
Preorder traversal of after search Splay tree is
20 50 30 40 100 200
Process exited after 0.01631 seconds with return value 0
Press any key to continue . . .
```

```
#include<stdio.h>
void quicksort(int number[25],int first,int last)
{
int i, j, pivot, temp;
if(first<last)
{
```

RESULT: Hence The above Program Splay Tree is executed successfully.

7 AIM OF THE EXPERIMENT: To Write a program to implement Quick Sort

DESCRIPTION : Quick Sort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. Always pick the first element as a pivot. Always pick the last element as a pivot. Pick a random element as a pivot. Pick median as the pivot.

SOURCE CODE:

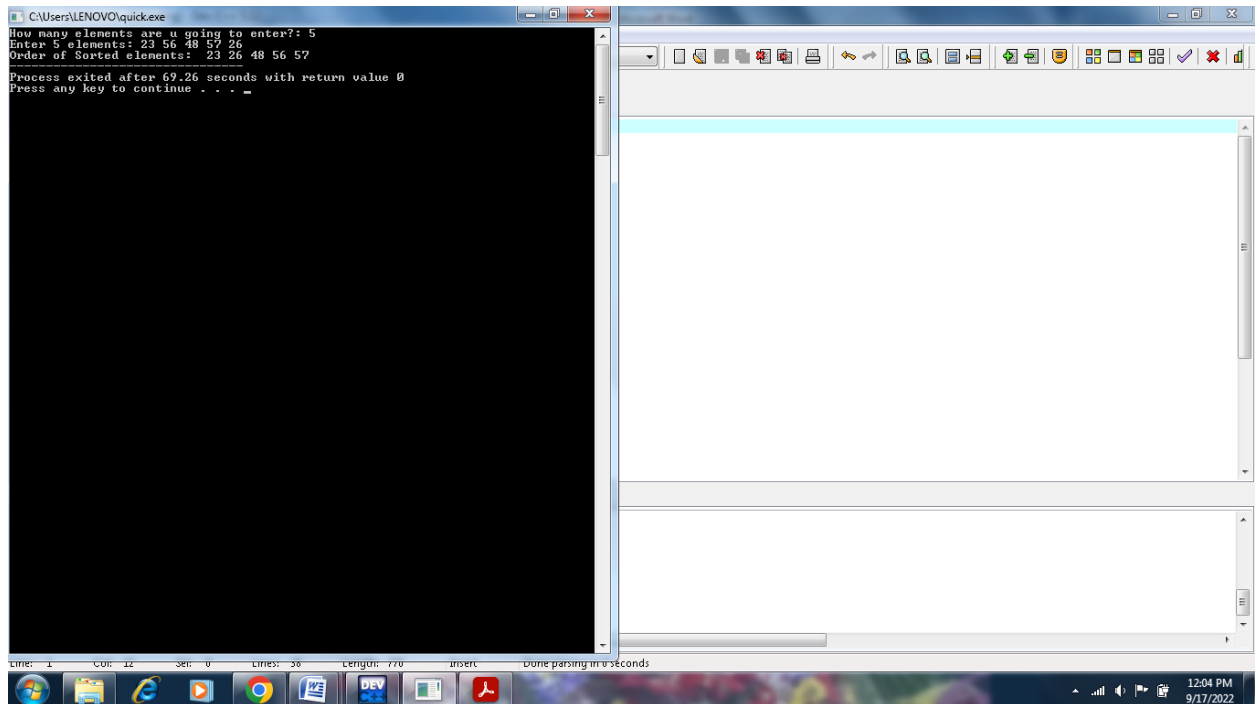
```
#include<stdio.h>
void quicksort(int number[25],int first,int last)
{
int i, j, pivot, temp;
if(first<last)
{
```

```
pivot=first;
i=first;
j=last;
while(i<j)
{
while(number[i]<=number[pivot]&& i<last)
i++;
while(number[j]>number[pivot])
j--;
if(i<j)
{
temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last);
}
}

int main()
{
int i, count, number[25];
printf("How many elements are u going to enter?: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
```

```
for(i=0;i<count;i++)
scanf("%d",&number[i]);
quicksort(number,0,count-1);
printf("Order of Sorted elements: ");
for(i=0;i<count;i++)
printf(" %d",number[i]);
return 0;
}
```

OUTPUT:



RESULT: Hence The above Program Quick Sort is executed successfully.

8 AIM OF THE EXPERIMENT: To Write a program to implement Merge Sort

DESCRIPTION : The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

SOURCE CODE:

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

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
```



```
}
while (i < n1)
{
arr[k] = L[i];
i++;
k++;
}
while (j < n2)
{
arr[k] = R[j];
j++;
k++;
}
}

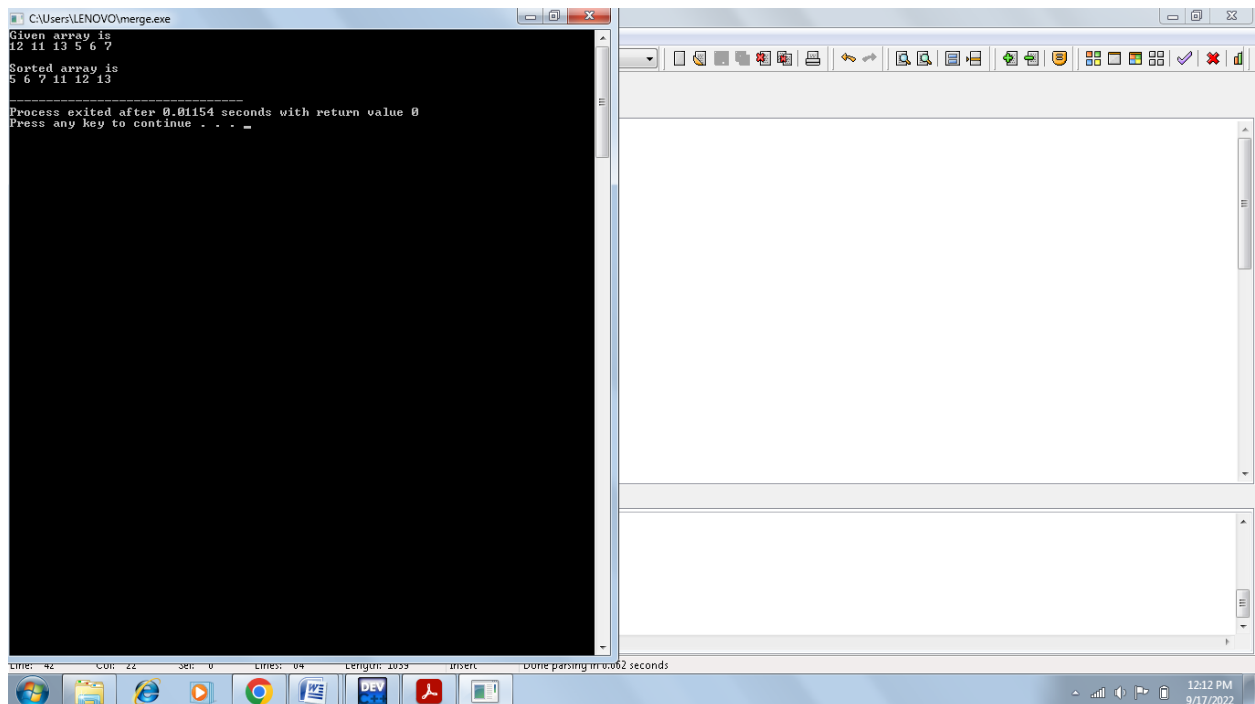
void mergeSort(int arr[], int l, int r)
{
if (l < r)
{
int m = l + (r - l) / 2;
mergeSort(arr, l, m);
mergeSort(arr, m + 1, r);
merge(arr, l, m, r);
}
}

void printArray(int A[], int size)
{
int i;
for (i = 0; i < size; i++)
printf("%d ", A[i]);
```

```
printf("\n");
}

int main()
{
int arr[] = { 12, 11, 13, 5, 6, 7 };
int arr_size = sizeof(arr) / sizeof(arr[0]);
printf("Given array is \n");
printArray(arr, arr_size);
mergeSort(arr, 0, arr_size - 1);
printf("\nSorted array is \n");
printArray(arr, arr_size);
return 0;
}
```

OUTPUT:



```
C:\Users\LENOVO\merge.exe
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
Process exited after 0.01154 seconds with return value 0
Press any key to continue . . .
```

RESULT: Hence The above Program Merge Sort is executed successfully.

9 AIM OF THE EXPERIMENT: To Write a program to find solution for knapsack using Greedy's method.

DESCRIPTION : The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem

SOURCE CODE:

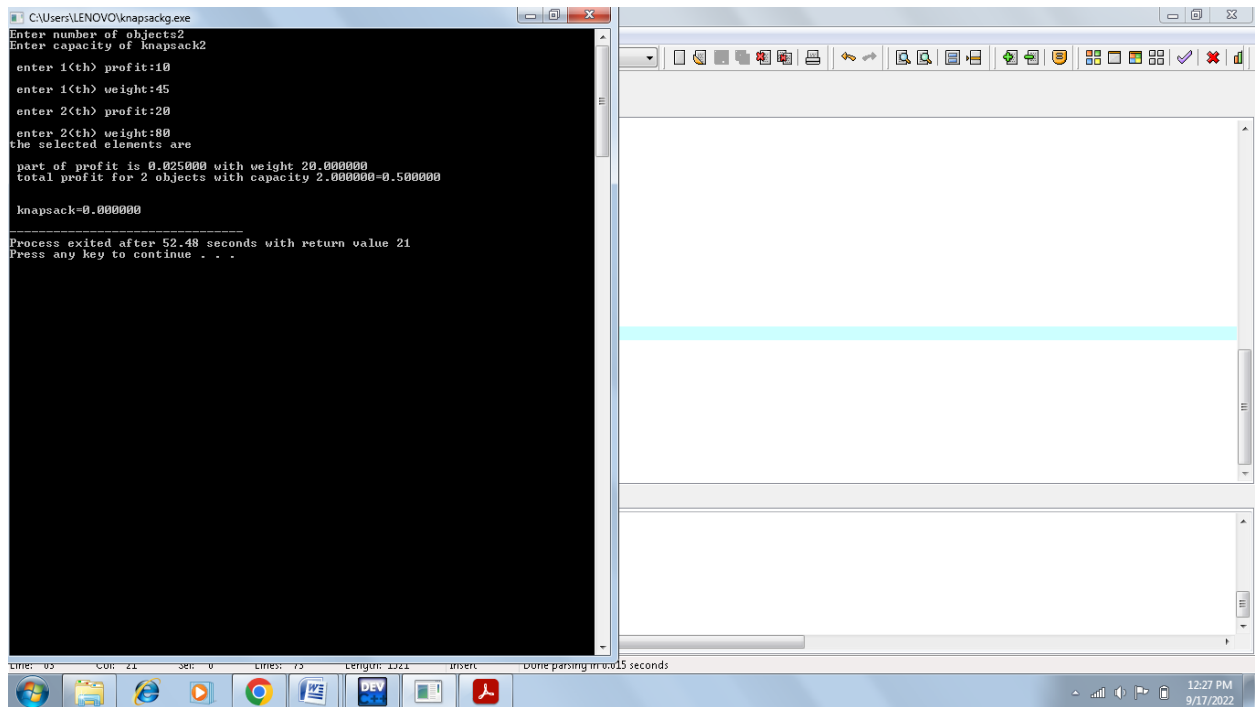
```
#include<stdio.h>
#include<time.h>
#include<conio.h>

void knapsack(float capacity,int n,float weight[],float profit[])
{
float x[20],totalprofit,y;
int i,j;
y=capacity;
totalprofit=0;
for(i=0;i<n;i++)
x[i]=0.0;
for(i=0;i<n;i++)
{
if(weight[i]>y)
break;
else
{
x[i]=1.0;
totalprofit=totalprofit+profit[i];
y=y-weight[i];
}
}
```

```
}
if(i<n)
x[i]=y/weight[i];
totalprofit=totalprofit+(x[i]*profit[i]);
printf("the selected elements are \n");
for(i=0;i<n;i++)
if( x[i]==1.0)
printf("\n profit is %f with height %f",profit[i],weight[i]);
else if(x[i]>0.0)
printf("\n part of profit is %f with weight %f",x[i],profit[i],weight[i]);
printf("\n total profit for %d objects with capacity %f=%f \n \n",n,capacity,totalprofit);
}
void main()
{
float weight[20],profit[20],ratio[20],t1,t2,t3;
int n;
time_t start,stop;
float capacity;
int i,j;
printf("Enter number of objects");
scanf("%d",&n);
printf("Enter capacity of knapsack");
scanf("%f",&capacity);
for(i=0;i<n;i++)
{
printf("\n enter %d(th) profit:",(i+1));
scanf("%f",&profit[i]);
printf("\n enter %d(th) weight:",(i+1));
scanf("%f",&weight[i]);
```

```
ratio[i]=profit[i]/weight[i];
}
start=time(NULL);
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(ratio[i]>ratio[j])
{
t1=ratio[i];
ratio[i]=ratio[j];
ratio[j]=t1;
t2=weight[i];
weight[i]=weight[j];
weight[j]=t2;
t3=profit[i];
profit[i]=profit[j];
profit[j]=t3;
}
}
knapsack(capacity,n,weight,profit);
stop=time(NULL);
printf("\n knapsack=%f \n",difftime(stop,start));
}
```

OUTPUT:



```
C:\Users\LENOVO\knapsackg.exe
Enter number of objects:2
Enter capacity of knapsack:2
enter 1<th> profit:10
enter 1<th> weight:45
enter 2<th> profit:20
enter 2<th> weight:80
the selected elements are
part of profit is 0.025000 with weight 20.000000
total profit for 2 objects with capacity 2.000000=0.500000

knapsack=0.000000

-----
Process exited after 52.48 seconds with return value 21
Press any key to continue . . .
```

RESULT: Hence The above Program knapsack using Greedy's method is executed successfully.

10 AIM OF THE EXPERIMENT: To Write a program to find minimum cost spanning tree using prim's algorithm.

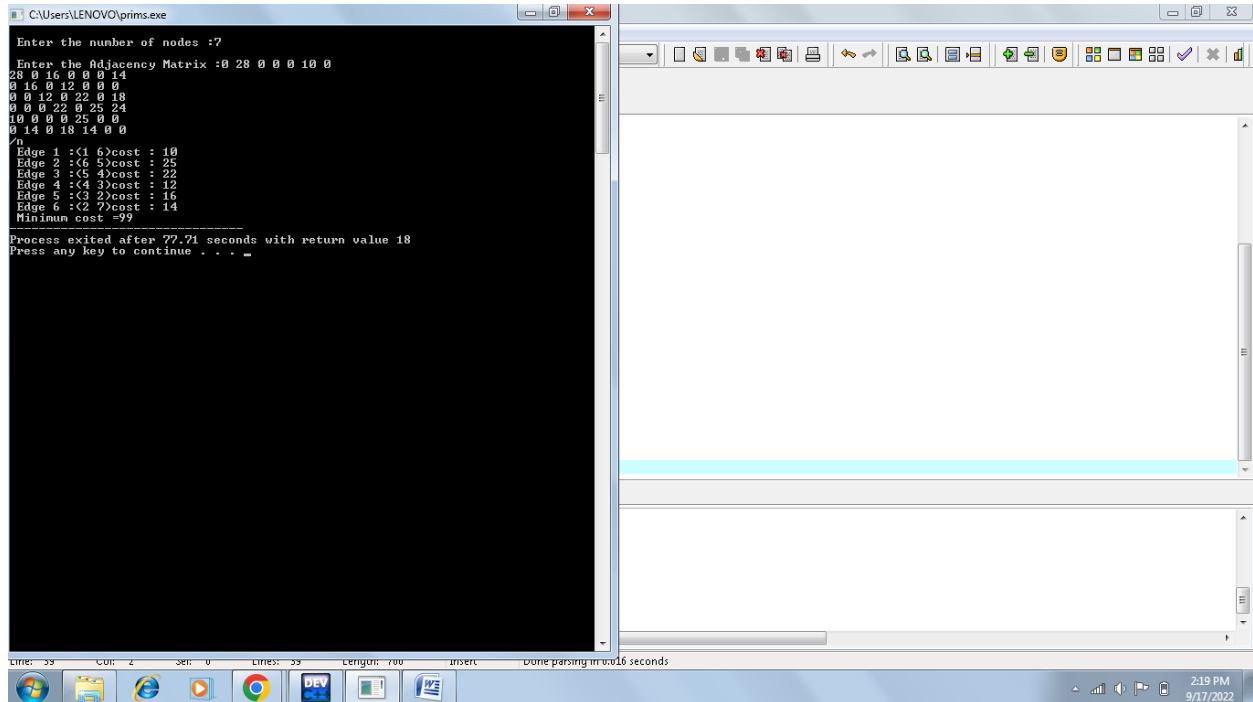
DESCRIPTION : Prim's algorithm is also a Greedy algorithm. Prim's algorithm always starts with a single node and it moves through several adjacent nodes, in order to explore all of the connected edges along the way. find a cut (of two sets, one contains the vertices already included in MST and the other contains the rest of the vertices), pick the minimum weight edge from the cut, and include this vertex to MST Set (the set that contains already included vertices).

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,i,j,n,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
```

```
{
printf("\n Enter the number of nodes :");
scanf("%d",&n);
printf("\n Enter the Adjacency Matrix :");
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[4]==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);  
}
```

OUTPUT:

```
C:\Users\LENOVO\prims.exe  
Enter the number of nodes :7  
Enter the Adjacency Matrix :0 28 0 0 0 10 0  
28 0 16 0 0 0 14  
0 16 0 12 0 0 0  
0 0 12 0 22 0 18  
0 0 0 22 0 25 24  
10 0 0 0 25 0 0  
0 14 0 18 14 0 0  
/n  
Edge 1 : (1 6) cost : 10  
Edge 2 : (6 5) cost : 25  
Edge 3 : (5 4) cost : 22  
Edge 4 : (4 3) cost : 12  
Edge 5 : (3 2) cost : 16  
Edge 6 : (2 7) cost : 14  
Minimum cost =99  
Process exited after 77.74 seconds with return value 18  
Press any key to continue . . .
```

RESULT: Hence The above Program to find minimum cost spanning tree using prim's algorithm is executed successfully.

11 AIM OF THE EXPERIMENT: To Write a program to find minimum cost spanning tree using kruskal's algorithm.

DESCRIPTION : Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the minimum cost spanning tree constructed so far all the edges in non-decreasing order of their weight. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it. Repeat step until there are (V-1) edges in the spanning tree.

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

int a,b,u,v,i,j,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

printf("\n\n\t Implementation of kruskal's Algorithm:\n\n");

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

scanf("%d",&n);

printf("\n Enter the cost Adjacency Matrix :");

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("\nThe edges of minimum cost spanning tree are\n\n");

while(ne<n)

{

for(i=1,min=999;i<=n;i++)

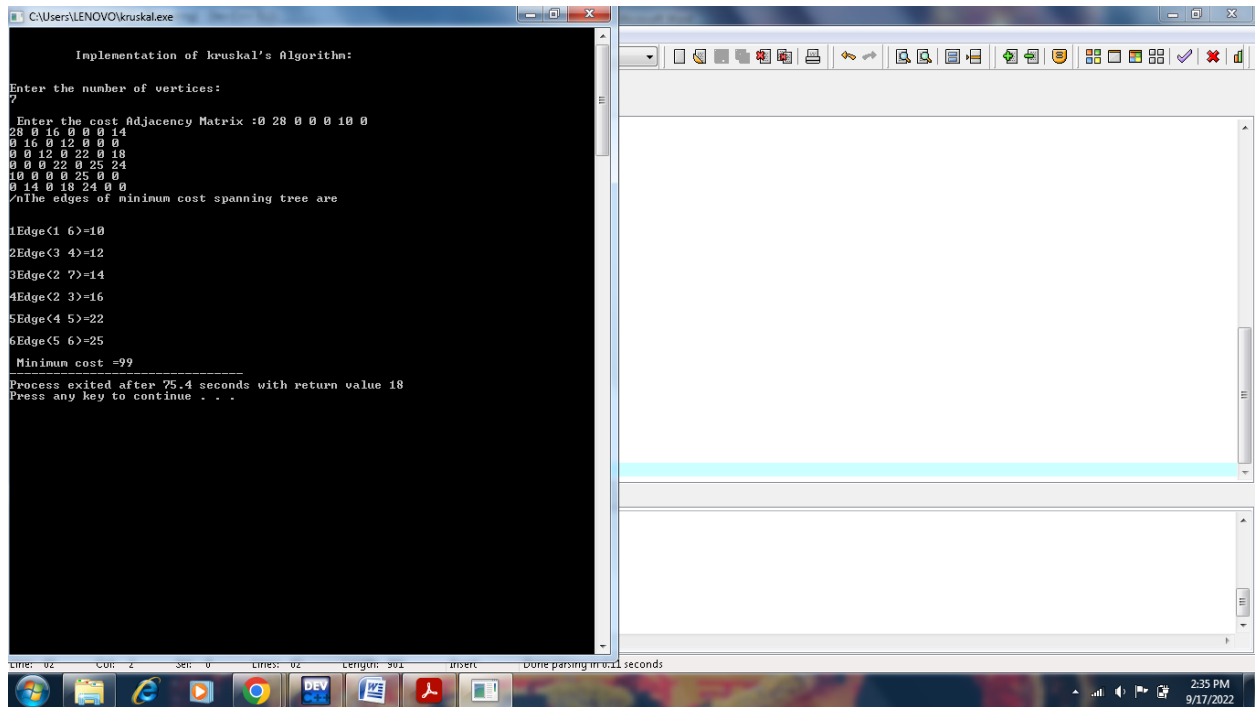
{

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("\n%dEdge(%d %d)=%d\n",ne++,a,b,min);
mincost+=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n Minimum cost =%d",mincost);
}
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;
}
```

OUTPUT:

```
Implementation of kruskal's algorithm:
Enter the number of vertices:
7
Enter the cost Adjacency Matrix :0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 18 24 0 0
\nThe edges of minimum cost spanning tree are
1Edge<1 6>=10
2Edge<3 4>=12
3Edge<2 7>=14
4Edge<2 3>=16
5Edge<4 5>=22
6Edge<5 6>=25
Minimum cost =99
-----
Process exited after 75.4 seconds with return value 18
Press any key to continue . . .
```

RESULT: Hence The above Program to find minimum cost spanning tree using Kruskal's algorithm is executed successfully.

12 AIM OF THE EXPERIMENT: To Write a program to find single source shortest path for given graph.

DESCRIPTION : Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's minimum spanning tree, generate a SPT (shortest path tree) with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet

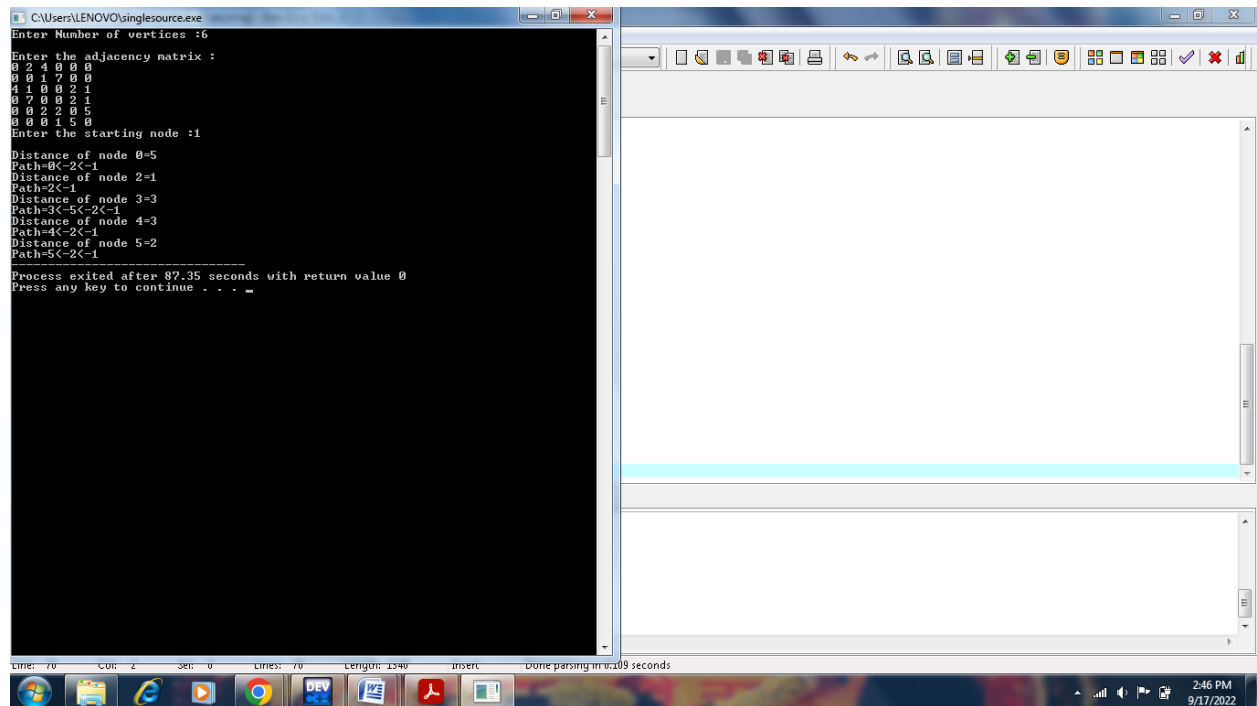
included) and has a minimum distance from the source.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.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 Number 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("Enter the starting node :");
    scanf("%d",&u);
    dijkstra(G,n,u);
    return 0;
}
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);  
}  
}
```

OUTPUT:

```
C:\Users\LENOVO\inglesource.exe  
Enter Number of vertices :16  
Enter the adjacency matrix :  
0 2 4 0 0 0  
0 0 1 7 0 0  
4 1 0 0 2 1  
0 7 0 0 2 1  
0 0 2 2 0 5  
0 0 0 1 5 0  
Enter the starting node :1  
Distance of node 0=5  
Path=0<-2<-1  
Distance of node 2=1  
Path=2<-1  
Distance of node 3=3  
Path=3<-5<-2<-1  
Distance of node 4=3  
Path=4<-2<-1  
Distance of node 5=2  
Path=5<-2<-1  
-----  
Process exited after 87.35 seconds with return value 0  
Press any key to continue . . .
```

RESULT: Hence The above Program to find single source shortest path for the graph is executed successfully.

13 AIM OF THE EXPERIMENT: To Write a program to find solution of Job Sequence with deadlines problem.

DESCRIPTION : Job Sequencing Problem with Deadline consists of n jobs each associated with a deadline and profit and our objective is to earn maximum profit. We will earn profit only when job is completed on or before deadline. We assume that each job will take unit time to complete.

SOURCE CODE:

```
#include <stdio.h>
#define MAX 100
typedef struct Job
{
    char id[5];
    int deadline;
    int profit;
} Job;
void jobSequencingWithDeadline(Job jobs[], int n);
int minValue(int x, int y) {
    if(x < y) return x;
    return y;
}
int main(void)
{
    //variables
    int i, j;
    //jobs with deadline and profit
    Job jobs[5] =
    {
        {"j1", 2, 60},
        {"j2", 1, 100},
```

```
{ "j3", 3, 20},
{ "j4", 2, 40},
{ "j5", 1, 20},
};

//temp
Job temp;

//number of jobs
int n = 5;

//sort the jobs profit wise in descending order
for(i = 1; i < n; i++)
{
    for(j = 0; j < n - i; j++)
    {
        if(jobs[j+1].profit > jobs[j].profit)
        {
            temp = jobs[j+1];
            jobs[j+1] = jobs[j];
            jobs[j] = temp;
        }
    }
}

printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
for(i = 0; i < n; i++)
{
    printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
}

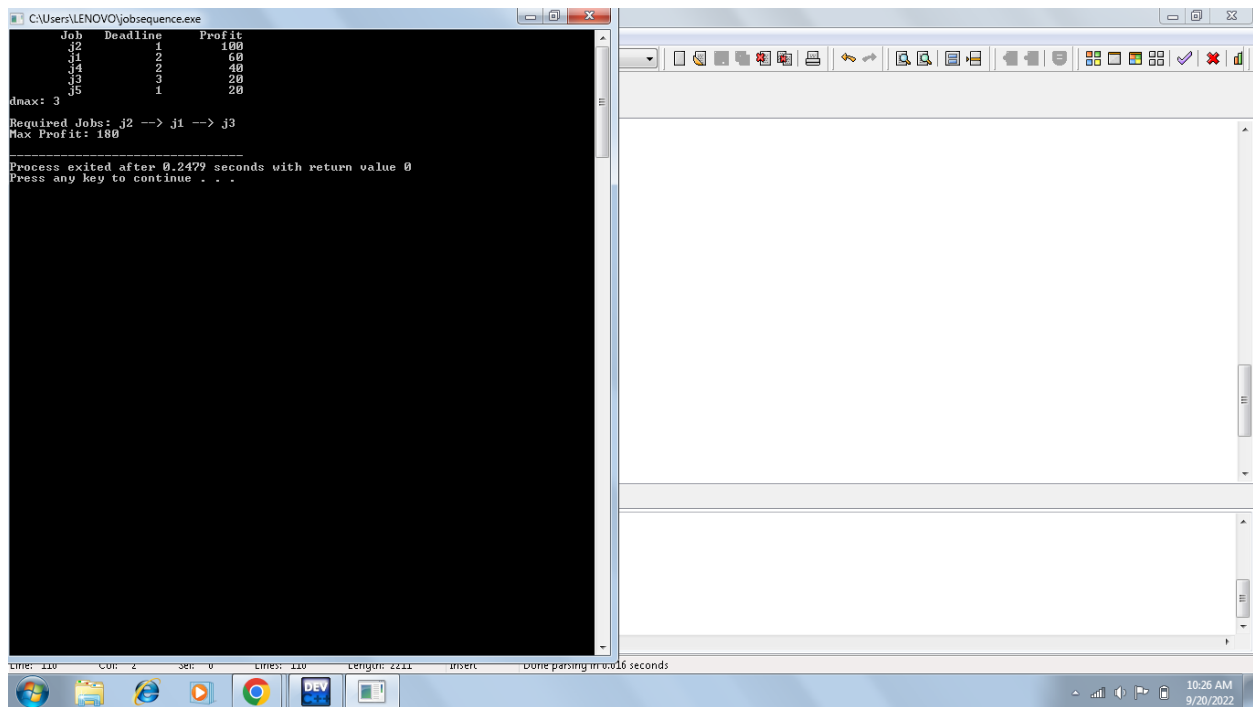
jobSequencingWithDeadline(jobs, n);

return 0;
}
```



```
void jobSequencingWithDeadline(Job jobs[], int n)
{
    //variables
    int i, j, k, maxprofit;
    //free time slots
    int timeslot[MAX];
    //filled time slots
    int filledTimeSlot = 0;
    //find max deadline value
    int dmax = 0;
    for(i = 0; i < n; i++) {
        if(jobs[i].deadline > dmax) {
            dmax = jobs[i].deadline;
        }
    }
    //free time slots initially set to -1 [-1 denotes EMPTY]
    for(i = 1; i <= dmax; i++)
    {
        timeslot[i] = -1;
    }
    printf("dmax: %d\n", dmax);
    for(i = 1; i <= n; i++)
    {
        k = minValue(dmax, jobs[i - 1].deadline);
        while(k >= 1)
        {
            if(timeslot[k] == -1)
            {
                timeslot[k] = i-1;
            }
        }
    }
}
```

```
        filledTimeSlot++;
        break;
    }
    k--;
}
//if all time slots are filled then stop
if(filledTimeSlot == dmax)
{
    break;
}
}
//required jobs
printf("\nRequired Jobs: ");
for(i = 1; i <= dmax; i++)
{
    printf("%s", jobs[timeslot[i]].id);
    if(i < dmax)
    {
        printf(" --> ");
    }
}
//required profit
maxprofit = 0;
for(i = 1; i <= dmax; i++)
{
    maxprofit += jobs[timeslot[i]].profit;
}
printf("\nMax Profit: %d\n", maxprofit);
}
```

OUTPUT:

```
C:\Users\LENOVO\jobsequence.exe
Job Deadline Profit
j2      1     100
j1      2      60
j4      2      40
j3      3      20
j5      1      20
dmax: 3
Required Jobs: j2 -> j1 -> j3
Max Profit: 180

Process exited after 0.2479 seconds with return value 0
Press any key to continue . . .
```

RESULT: Hence The above Program find solution of Job Sequence with deadlines problem is executed successfully.

14 AIM OF THE EXPERIMENT: To Write a program to find solution of 0/1 knapsack problem using dynamic programming.

DESCRIPTION : A knapsack is a bag. And the knapsack problem deals with the putting items to the bag based on the value of the items. It aim is to maximise the value inside the bag. In 0-1 Knapsack you can either put the item or discard it, there is no concept of putting some part of item in the knapsack.

SOURCE CODE:

```
#include<stdio.h>

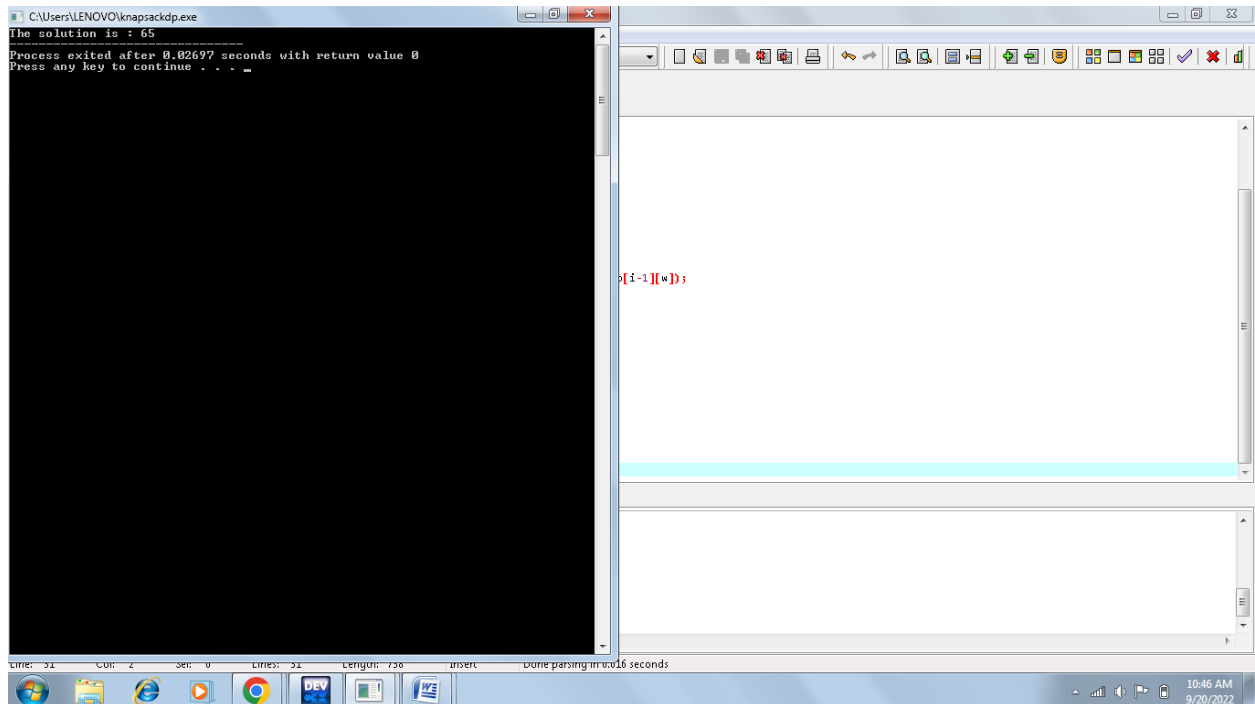
int max(int a, int b)
{
    if(a>b)
    {
        return a;
    }
}
```

```
else
{
    return b;
}
}

int knapsack(int W, int wt[], int val[], int n)
{
    int i, w;
    int knap[n+1][W+1];
    for (i = 0; i <= n; i++)
    {
        for (w = 0; w <= W; w++)
        {
            if (i==0 || w==0)
                knap[i][w] = 0;
            else if (wt[i-1] <= w)
                knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
            else
                knap[i][w] = knap[i-1][w];
        }
    }
    return knap[n][W];
}

int main()
{
    int val[] = {20, 25, 40};
    int wt[] = {25, 20, 30};
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("The solution is : %d", knapsack(W, wt, val, n));
    return 0;
}
```

```
}
```

OUTPUT:

```
C:\Users\LENOVO\knapsackdp.exe
The solution is : 65
Process exited after 0.02697 seconds with return value 0
Press any key to continue . . .

p[i-1][*]);
```

RESULT: Hence The above Program find solution of 0/1 knapsack problem using dynamic programming is executed successfully.

15 AIM OF THE EXPERIMENT: To Implement N Queen's problem using Backtracking.

DESCRIPTION : Backtracking means taking a step back , tracing the route back upto a particular checkpoint from where you can again go ahead and take another route to your desired destination. The N Queen is the problem of placing N chess queens on an $N \times N$ chessboard so that no two queens attack each other.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int chess_board[20], count;
void nqueen_function(int row, int num)
```

```
{
int col;
for(col = 1; col <= num; col++)
{
if(placeholder(row, col))           //function call to check where to place the queen
{
chess_board[row] = col;
if(row == num)                     //ensures if all queens are placed or not
{
display(num);                     //once all queens placed; display solution.
}
}
else
{
nqueen_function(row + 1, num); //function call to place remaining queens.
}
}
}
}

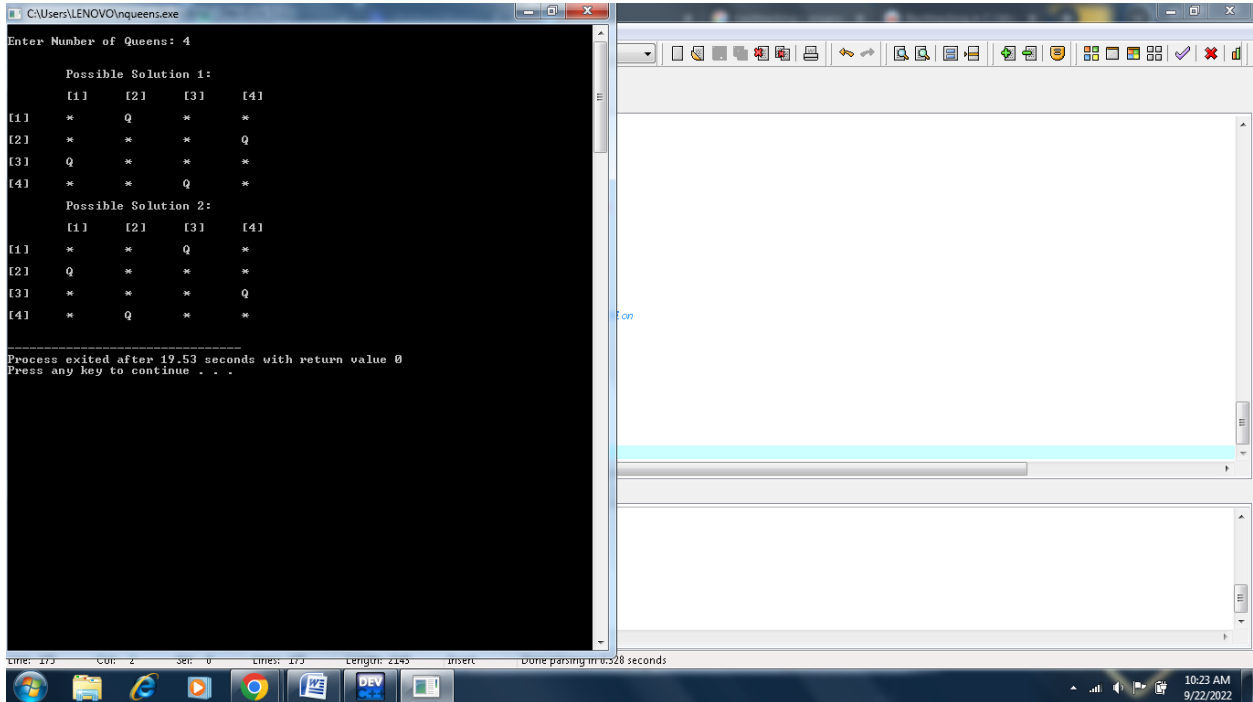
int placeholder(int row, int col)
{
int count;
for(count = 1; count <= row - 1; count++)
{
if(chess_board[count] == col) //checks if there is any threat from queens placed previous
{
return 0;
}
}
else
{

```

```
if(abs(chess_board[count] - col) == abs(count - row)) //check for diagonal conflicts.
{
return 0;
}
}
}
return 1; //no threats, queen can be placed.
}

int display(int num)
{
int m, n;
printf("\n\n\tPossible Solution %d:\n\n", ++count);
for(m = 1; m <= num; m++)
{
printf("\t[%d]", m);
}
for(m = 1; m <= num; m++)
{
printf("\n\n[%d]", m);
for(n = 1; n <= num; n++)
{
if(chess_board[m] == n)
{
printf("\tQ"); //queen at (i,j) position.
}
else
{
printf("\t*"); //empty slot.
}
}
```

```
}  
}  
}  
int main()  
{  
    int num;  
    printf("\nEnter Number of Queens:\t");  
    scanf("%d", &num);  
    if(num <= 3)  
    {  
        printf("\nNumber should be greater than 3 to form a Matrix\n");  
    }  
    else  
    {  
        nqueen_function(1, num);           //call to nqueen function  
    }  
    printf("\n\n");  
    return 0;  
}
```


OUTPUT:

```
C:\Users\LENOVO\nqueens.exe
Enter Number of Queens: 4

Possible Solution 1:
[1] * Q * *
[2] * * * Q
[3] Q * * *
[4] * * Q *

Possible Solution 2:
[1] * * Q *
[2] Q * * *
[3] * * * Q
[4] * Q * *

Process exited after 19.53 seconds with return value 0
Press any key to continue . . .
```

RESULT: Hence The above Program to implement N Queen's Problem using Backtracking is executed successfully.

16 AIM OF THE EXPERIMENT: To write a program to perform Binary Search for given set of integer values.

DESCRIPTION : Binary search in C language to find an element in a sorted array. If the array isn't sorted, you must sort it using a sorting technique such as merge sort. If the element to search is present in the list, then we print its location. The program assumes that the input numbers are in ascending order.

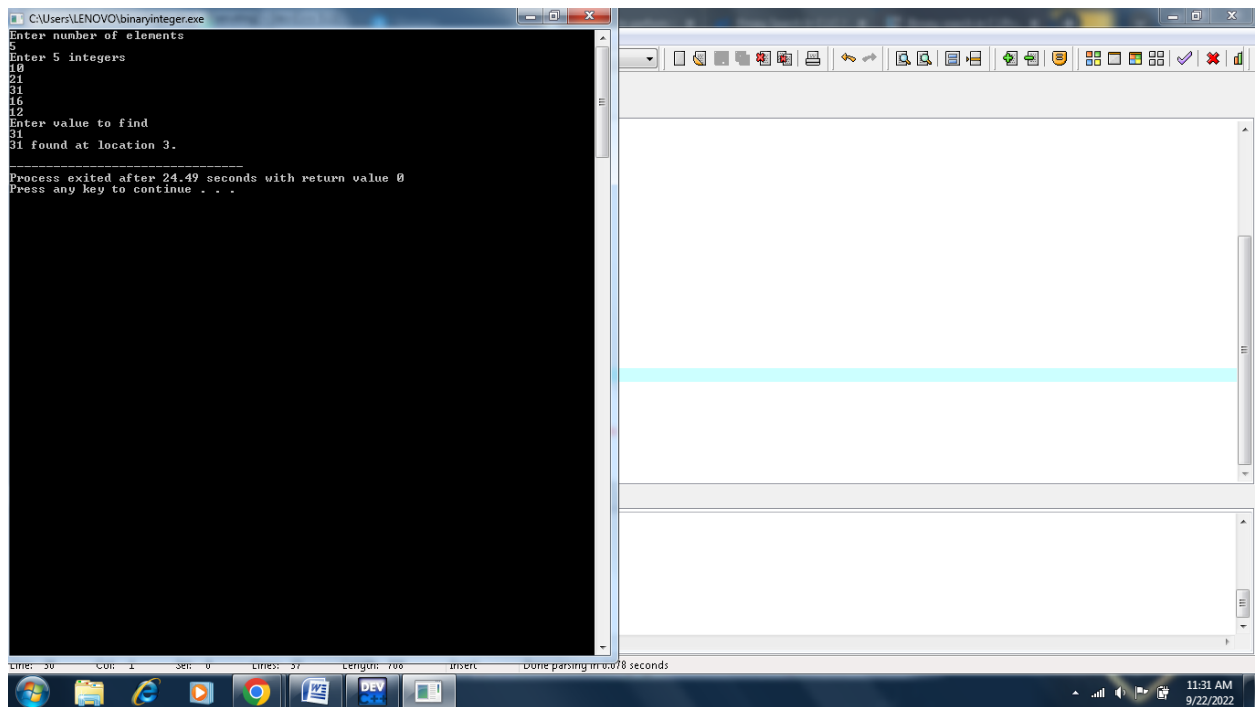
SOURCE CODE:

```
#include <stdio.h>

int main()
{
    int c, first, last, middle, n, search, array[100];
    printf("Enter number of elements\n");
    scanf("%d", &n);
```

```
printf("Enter %d integers\n", n);
for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
printf("Enter value to find\n");
scanf("%d", &search);
first = 0;
last = n - 1;
middle = (first+last)/2;
while (first <= last)
{
    if (array[middle] < search)
        first = middle + 1;
    else if (array[middle] == search)
    {
        printf("%d found at location %d.\n", search, middle+1);
        break;
    }
    else
        last = middle - 1;
    middle = (first + last)/2;
}
if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);
return 0;
}
```

OUTPUT:



```
C:\Users\LENOVO\binaryinteger.exe
Enter number of elements
5
Enter 5 integers
10
21
31
10
12
Enter value to find
31
31 found at location 3.

Process exited after 24.49 seconds with return value 0
Press any key to continue . . .
```

RESULT: Hence The above Program to perform binary search for given set of integer values is executed successfully.