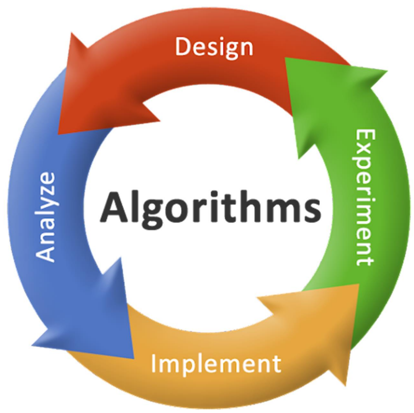
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Ramapuram Campus, Chennai- 600089.
FACULTY OF ENGINEERING AND TECHNOLOGY
Department of Computer Science & Engineering

DESIGN & ANALYSIS OF ALGORITHMS

LAB MANUAL



CLASS	**:**	**B.Tech. [U.G]**
YEAR	**:**	**II YEAR**
SEM.	**:**	**IV**
SOFTWARE REQUIREMENT	**:**	**Turbo C**

Prepared By

Dr. M.Azhagiri AP/CSE

Ex No	List of Experiment
1 A	Simple Algorithm
1 B	Insertion sort
2	Bubble Sort
3 A	Recurrence Type-Merge sort
3 B	Recurrence Type- Linear search
4 A	Quicksort
4 B	Binary search
5	Strassen Matrix multiplication
6 A	Finding Maximum and Minimum in an array
6 B	Convex Hull problem
7 A	Huffman coding Using Greedy
7 B	knapsack using greedy
8 A	Tree Traversal
8 B	Krukshall's MST
8 C	Prims MST
9	Longest common subsequence
10	N queen's problem
11	Travelling salesman problem
12	BFS and DFS implementation with array
13	Randomized quick sort
14	String matching algorithms
15	Discussion over analyzing a real time problem

Ex No: 1 A	FACTORIAL OF N NUMBERS
Date:	

AIM

To Write a C Program for finding Factorial of N numbers

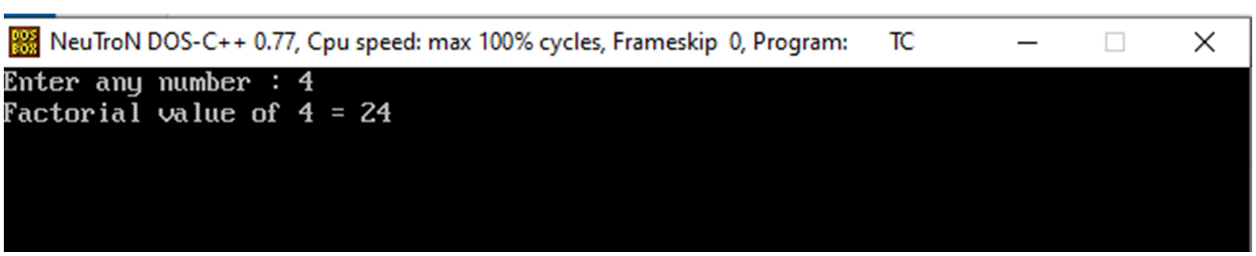
ALGORITHM

step 1. Start
step 2. Read the number n
step 3. [Initialize]
    i=1, fact=1
step 4. Repeat step 4 through 6 until i=n
step 5. fact=fact*i
step 6. i=i+1
step 7. Print fact
step 8. Stop

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int n,i,fact=1;
 clrscr();
 printf("Enter any number : ");
 scanf("%d", &n);
 for(i=1; i<=n; i++)
   fact = fact * i;
 printf("Factorial value of %d = %d",n,fact);
 getch();
}
```

INPUT & OUTPUT



RESULT

Thus the C Program for Finding factorial of N numbers has been executed and the output has been verified

Ex No: 1 B	INSERTION SORT
Date:	

AIM

To Write a C Program for Sorting of N numbers using Insertion Sort

ALGORITHM

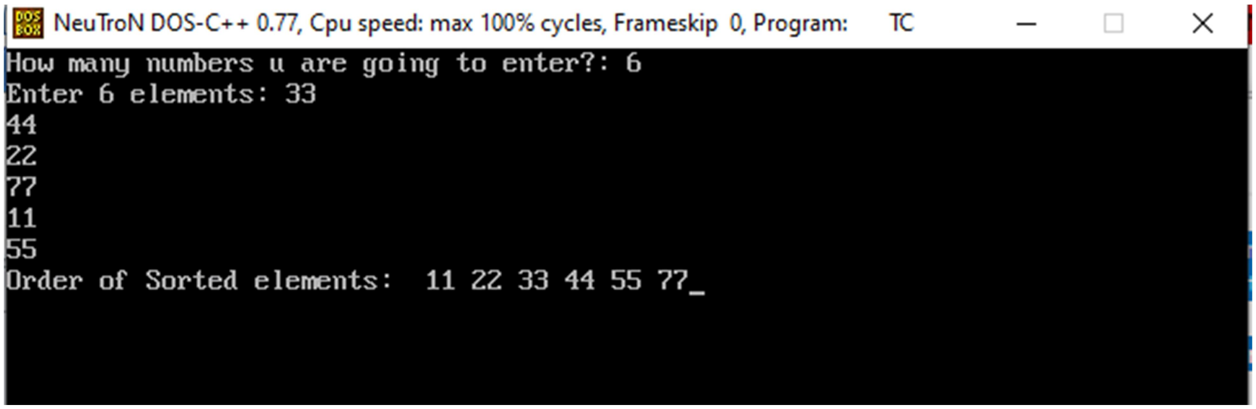
Step 1 − If the element is the first one, it is already sorted.
Step 2 – Move to next element
Step 3 − Compare the current element with all elements in the sorted array
Step 4 – If the element in the sorted array is smaller than the current element, iterate to the next element. Otherwise, shift the entire greater element in the array by one position towards the right
Step 5 − Insert the value at the correct position
Step 6 − Repeat until the complete list is sorted

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main(){
int i, j, count, temp, number[25];
clrscr();
printf("How many numbers u are going to enter?: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
// This loop would store the input numbers in array
for(i=0;i<count;i++)
scanf("%d",&number[i]);
// Implementation of insertion sort algorithm
for(i=1;i<count;i++){
temp=number[i];
j=i-1;
while((temp<number[j])&&(j>=0)){
number[j+1]=number[j];
j=j-1;
}
number[j+1]=temp;
}
printf("Order of Sorted elements: ");
for(i=0;i<count;i++)
```

```
printf(" %d",number[i]);
getch();
}
```

INPUT & OUTPUT



RESULT

Thus the C Program for Sorting of N numbers using Insertion Sort has been executed and the output has been verified

Ex No: 2	Bubble Sort
Date:	

AIM

To Write a C Program for Sorting of N numbers using Bubble Sort

ALGORITHM

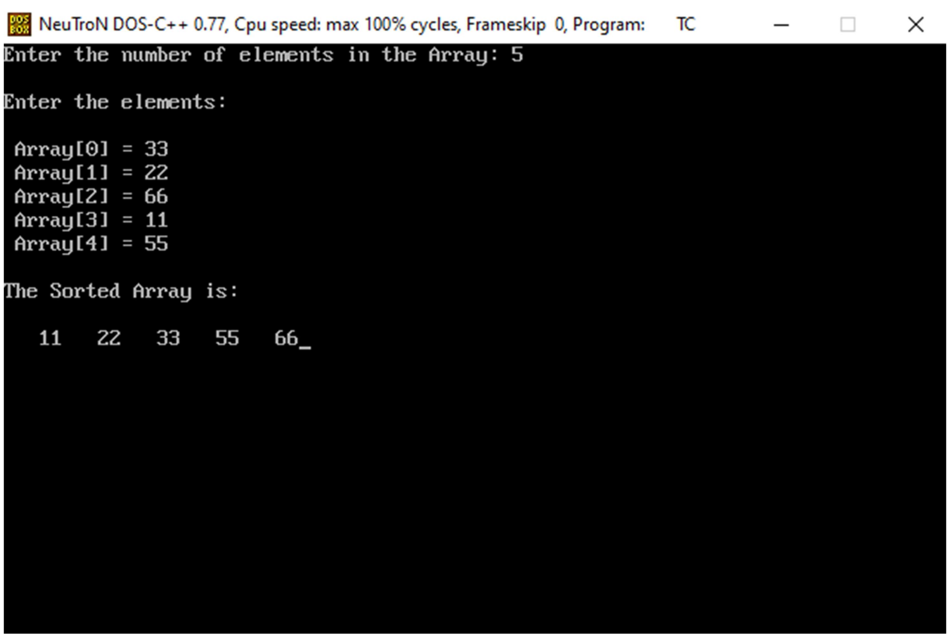
Step 1: Repeat Steps 2 and 3 for i=1 to 10
Step 2: Set j=1
Step 3: Repeat while j<=n
        if a[i] < a[j]
        Then interchange a[i] and a[j]
        [End of if]
          Set j = j+1
        [End of Inner Loop]
    [End of Step 1 Outer Loop]
Step 4: Exit

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,n,temp,j,arr[25];
 clrscr();
 printf("Enter the number of elements in the Array: ");
 scanf("%d",&n);
 printf("\nEnter the elements:\n\n");

 for(i=0 ; i<n ; i++)
 {
  printf(" Array[%d] = ",i);
  scanf("%d",&arr[i]);
 }

 for(i=0 ; i<n ; i++)
 {
  for(j=0 ; j<n-i-1 ; j++)
  {
```

```
  if(arr[j]>arr[j+1]) //Swapping Condition is Checked
  {
   temp=arr[j];
   arr[j]=arr[j+1];
   arr[j+1]=temp;
  }
 }
}
 printf("\nThe Sorted Array is:\n\n");
for(i=0 ; i<n ; i++)
{
 printf(" %4d",arr[i]);
}
getch();
}
```

INPUT& OUTPUT



RESULT

Thus the C Program for Sorting of N numbers using Bubble Sort has been executed and the output has been verified

Ex No: 3 A	Recurrence Type-Merge sort
Date:	

AIM

To Write a C Program for Sorting of N numbers using merge sort

ALGORITHM

Step 1 : Declare left and right var which will mark the extreme indices of the array
Step 2: Left will be assigned to 0 and right will be assigned to n-1
Step 3: Find mid = (left+right)/2
Step 4: Call mergeSort on (left,mid) and (mid+1,rear)

```
MergeSort(arr, left, right):
    if left > right
        return
    mid = (left+right)/2
    mergeSort(arr, left, mid)
    mergeSort(arr, mid+1, right)
    merge(arr, left, mid, right)
end
```

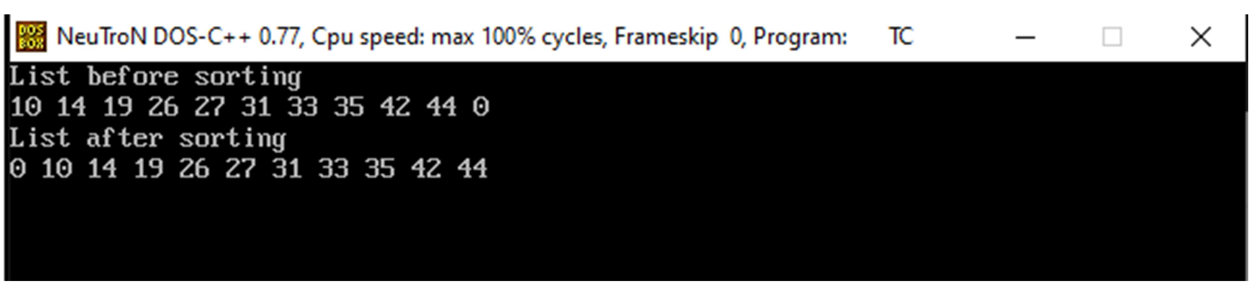
Step 5: continue till left<right
Step 6: merge on the 2 sub problems

PROGRAM

```
#include <stdio.h>
#define max 10
int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];
void merging(int low, int mid, int high) {
int l1, l2, i;
for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
 if(a[l1] <= a[l2])
b[i] = a[l1++];
 else
 b[i] = a[l2++];
}
while(l1 <= mid)
b[i++] = a[l1++];
while(l2 <= high)
b[i++] = a[l2++];
```

```
for(i = low; i <= high; i++)
a[i] = b[i];
}
void sort(int low, int high) {
int mid;
if(low < high) {
mid = (low + high) / 2;
sort(low, mid);
sort(mid+1, high);
merging(low, mid, high);
} else
{
return;
}
}
int main()
{
int i;
clrscr();
printf("List before sorting\n");
for(i = 0; i <= max; i++)
printf("%d ", a[i]);
sort(0, max);
printf("\nList after sorting\n");
for(i = 0; i <= max; i++)
printf("%d ", a[i]);
getch();
}
```

INPUT& OUTPUT



RESULT

Thus the C Program for Sorting of N numbers using Merge Sort has been executed and the output has been verified

Ex No: 3 B	Recurrence Type-Linear Search
Date:	

AIM

To Write a C Program for searching an element using Linear Search

ALGORITHM

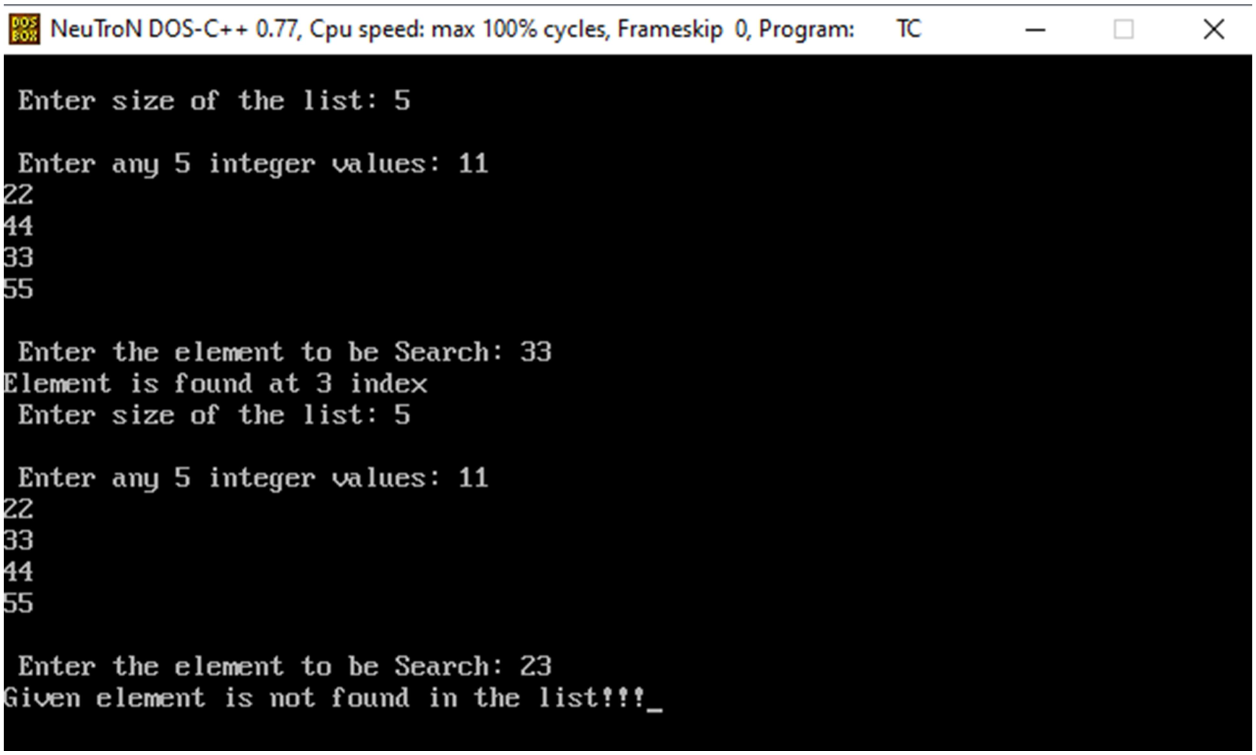
Step 1 - Read the search element from the user.
Step 2 - Compare the search element with the first element in the list.
Step 3 - If both are matched, then display "Given element is found!!!" and terminate the function
Step 4 - If both are not matched, then compare search element with the next element in the list.
Step 5 - Repeat steps 3 and 4 until search element is compared with last element in the list.
Step 6 - If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main(){
int list[20],size,i,sElement;
printf("Enter size of the list: ");
scanf("%d",&size);
printf("Enter any %d integer values: ",size);
for(i = 0; i < size; i++)
scanf("%d",&list[i]);
printf("Enter the element to be Search: ");
scanf("%d",&sElement);
// Linear Search Logic
for(i = 0; i < size; i++)
{
if(sElement == list[i])
{
printf("Element is found at %d index", i);
break;
}
}
if(i == size)
printf("Given element is not found in the list!!!");
getch();
```

}

INPUT& OUTPUT



RESULT

Thus the C Program for searching an element by using linear Search has been executed and the output has been verified

Ex No: 4 A	Quick sort
Date:	

AIM

To Write a C Program for Sorting of N numbers using Quick Sort

ALGORITHM

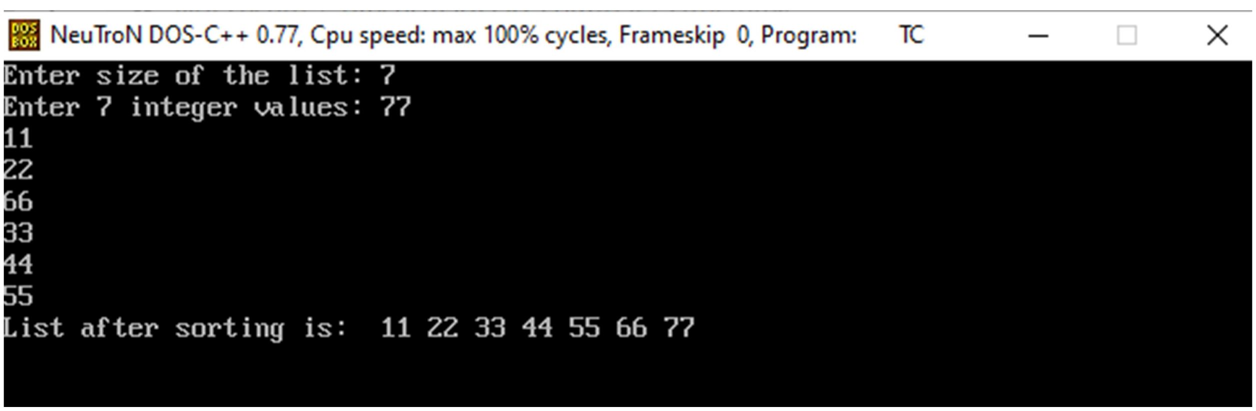
Step 1 - Consider the first element of the list as pivot (i.e., Element at first position in the list).
Step 2 - Define two variables i and j. Set i and j to first and last elements of the list respectively.
Step 3 - Increment i until list[i] > pivot then stop.
Step 4 - Decrement j until list[j] < pivot then stop.
Step 5 - If i < j then exchange list[i] and list[j].
Step 6 - Repeat steps 3,4 & 5 until i > j.
Step 7 - Exchange the pivot element with list[j] element.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void quickSort(int [10],int,int);
void main(){
int list[20],size,i;
printf("Enter size of the list: ");
scanf("%d",&size);
printf("Enter %d integer values: ",size);
for(i = 0; i < size; i++)
scanf("%d",&list[i]);
quickSort(list,0,size-1);
printf("List after sorting is: ");
for(i = 0; i < size; i++)
printf(" %d",list[i]);
getch();
}
void quickSort(int list[10],int first,int last){
int pivot,i,j,temp;
if(first < last)
{
pivot = first;
i = first;
```

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

INPUT & OUTPUT



RESULT

Thus the C Program for Sorting of N numbers using Quick Sort has been executed and the output has been verified

Ex No: 4 B	**Binary Search**
Date:	

AIM

To Write a C Program for searching an element using Binary Search

ALGORITHM

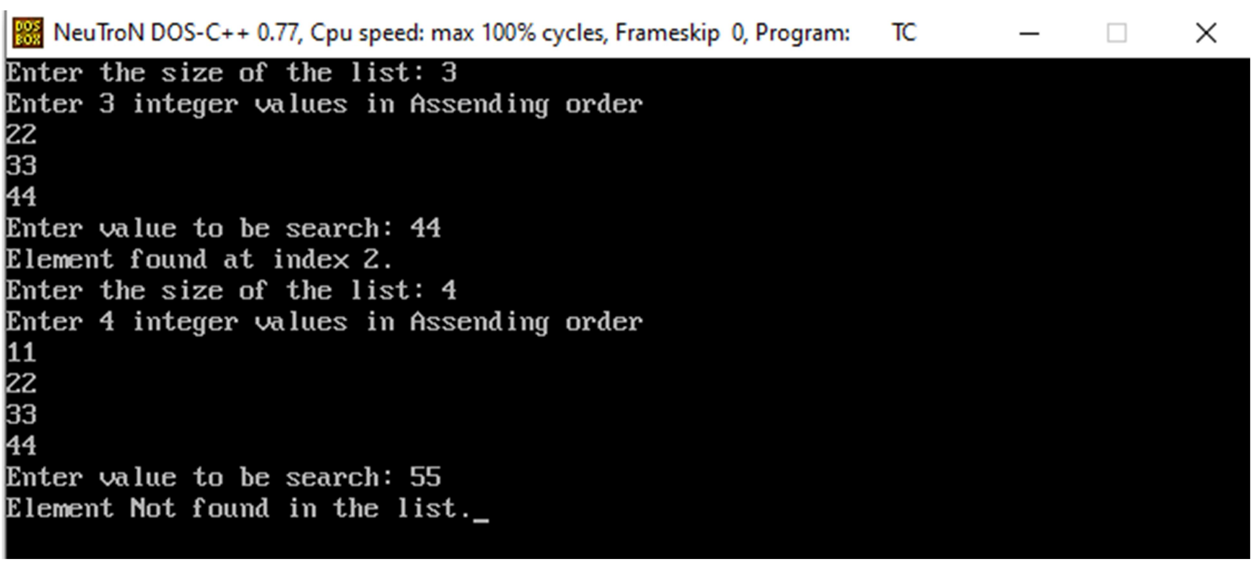
Step 1 - Read the search element from the user.
Step 2 - Find the middle element in the sorted list.
Step 3 - Compare the search element with the middle element in the sorted list.
Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.
Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.
Step 6 - If the search element is smaller than middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.
Step 7 - If the search element is larger than middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.
Step 8 - Repeat the same process until we find the search element in the list or until sublist contains only one element.
Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int first, last, middle, size, i, sElement, list[100];
clrscr();
printf("Enter the size of the list: ");
scanf("%d",&size);
printf("Enter %d integer values in Assending order\n", size);
for (i = 0; i < size; i++)
scanf("%d",&list[i]);
printf("Enter value to be search: ");
scanf("%d", &sElement);
first = 0;
last = size - 1;
```

```
middle = (first+last)/2;
while (first <= last) {
if (list[middle] < sElement)
first = middle + 1;
else if (list[middle] == sElement) {
printf("Element found at index %d.\n",middle);
break;
}
else
last = middle - 1;
middle = (first + last)/2;
}
if (first > last)
printf("Element Not found in the list.");
getch();
}
```

INPUT & OUTPUT



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the size of the list: 3
Enter 3 integer values in Assending order
22
33
44
Enter value to be search: 44
Element found at index 2.
Enter the size of the list: 4
Enter 4 integer values in Assending order
11
22
33
44
Enter value to be search: 55
Element Not found in the list._
```

RESULT

Thus the C Program for searching an element by using Binary Search has been executed and the output has been verified

Ex No: 5	**Strassen Matrix multiplication**
Date:	

AIM

To Write a C Program for performing matrix Multiplication using divide and Conquer

ALGORITHM

Step 1: Algorithm Strassen(n, a, b, d)
begin
Step 2: If n = threshold then compute
C = a * b is a conventional matrix.
Else
Step 3: Partition a into four sub matrices a11, a12, a21, a22.
Partition b into four sub matrices b11, b12, b21, b22.
Strassen (n/2, a11 + a22, b11 + b22, d1)
Strassen (n/2, a21 + a22, b11, d2)
Strassen (n/2, a11, b12 – b22, d3)
Strassen (n/2, a22, b21 – b11, d4)
Strassen (n/2, a11 + a12, b22, d5)
Strassen (n/2, a21 – a11, b11 + b22, d6)
Strassen (n/2, a12 – a22, b21 + b22, d7)
Step 4: C = d1+d4-d5+d7 d3+d5
d2+d4 d1+d3-d2-d6

end if

return (C)
end.

PROGRAM

```
#include<stdio.h>
int main(){
int a[2][2],b[2][2],c[2][2],i,j;
int m1,m2,m3,m4,m5,m6,m7;
printf("Enter the 4 elements of first matrix: ");
for(i=0;i<2;i++)
for(j=0;j<2;j++)
```

```
scanf("%d",&a[i][j]);
printf("Enter the 4 elements of second matrix: ");
for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&b[i][j]);
printf("\nThe first matrix is\n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",a[i][j]);
}
printf("\nThe second matrix is\n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",b[i][j]);
}
m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
m2= (a[1][0]+a[1][1])*b[0][0];
m3= a[0][0]*(b[0][1]-b[1][1]);
m4= a[1][1]*(b[1][0]-b[0][0]);
m5= (a[0][0]+a[0][1])*b[1][1];
m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
c[0][0]=m1+m4-m5+m7;
c[0][1]=m3+m5;
c[1][0]=m2+m4;
c[1][1]=m1-m2+m3+m6;
printf("\nAfter multiplication using \n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",c[i][j]);
}
return 0;
}
```

INPUT & OUTPUT

```
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program:    TC
Enter the 4 elements of first matrix: 1
2
3
4
Enter the 4 elements of second matrix: 5
6
7
8

The first matrix is

1       2
3       4
The second matrix is

5       6
7       8
After multiplication using

19      22
43      50      Enter the 4 elements of first matrix:
```

RESULT

Thus the C Program for multiplication of 2*2 elements Using Strassen Matrix multiplication has been executed and the output has been verified

Experiment 6A

Finding Max and Min in Array

Aim-

To write a C program to find the maximum and minimum elements of a given array.

Algorithm-

1. Let **maxE** and **minE** be the variable to store the minimum and maximum element of the array.
2. Initialise **minE** as INT_MAX and **maxE** as INT_MIN.
3. Traverse the given array arr[].
4. If the current element is smaller than **minE**, then update the **minE** as current element.
5. If the current element is greater than **maxE**, then update the **maxE** as current element.
6. Repeat the above two steps for the element in the array.

Code-

```
#include <limits.h>

#include <stdio.h>

void recursiveMinMax(int arr[], int N,

int* minE, int* maxE)

{

	if (N < 0) {

		return;

	}

	if (arr[N] < *minE) {

		*minE = arr[N];

	}

	if (arr[N] > *maxE) {

		*maxE = arr[N];

	}

	recursiveMinMax(arr, N - 1, minE, maxE);
```

```c
}
void findMinimumMaximum(int arr[], int N)
{
    int i;
    int minE = INT_MAX, maxE = INT_MIN;
    recursiveMinMax(arr, N - 1, &minE, &maxE);
    printf("The minimum element is %d", minE);
    printf("\n");
    printf("The maximum element is %d", maxE);
    return;
}
int main()
{
    int arr[] = { 1, 2, 4, -1 };
    int N = sizeof(arr) / sizeof(arr[0]);
    findMinimumMaximum(arr, N);
    return 0;
}
```

Output-

```c
#include <limits.h>
#include <stdio.h>
void recursiveMinMax(int arr[], int N,
int* minE, int* maxE)
{
    if (N < 0) {
        return;
    }
    if (arr[N] < *minE) {
        *minE = arr[N];
    }
    if (arr[N] > *maxE) {
        *maxE = arr[N];
    }
    recursiveMinMax(arr, N - 1, minE, maxE);
}
void findMinimumMaximum(int arr[], int N)
{
    int i;

    int minE = INT_MAX, maxE = INT_MIN;
    recursiveMinMax(arr, N - 1, &minE, &maxE);
    printf("The minimum element is %d", minE);
    printf("\n");
    printf("The maximum element is %d", maxE);
```

```
The minimum element is -1
The maximum element is 4

...Program finished with exit code 0
Press ENTER to exit console.
```

Result- A code has been written to find the maximum and minimum elements, and the output has been verified.

Experiment 6B

Convex Hull Problem

Aim-

To write a C program to use convex hull algorithm to find the convex hull length of a set of 2D points.

Algorithm-

1.Choose a point roughly in the centre of your point cloud.

2.Then sort the points radially, by angle from the centre. The topmost point must be in the convex hull, so define it as having an angle of 0.0 and being first in the list.

3.Put point 2 in the "tentative" hull list.

4.Then check point 3. If the angle P1-P2-P3 is concave (relative to the centre point), remove P2 from the list, if it is convex, keep it.

5. Continue steps 3-6, backtracking and removing points if they go concave.

6. You only need two points in your "tentative" list, once you have three, they become definite.

7.You stop when you go full circle and get back to P1

Code-

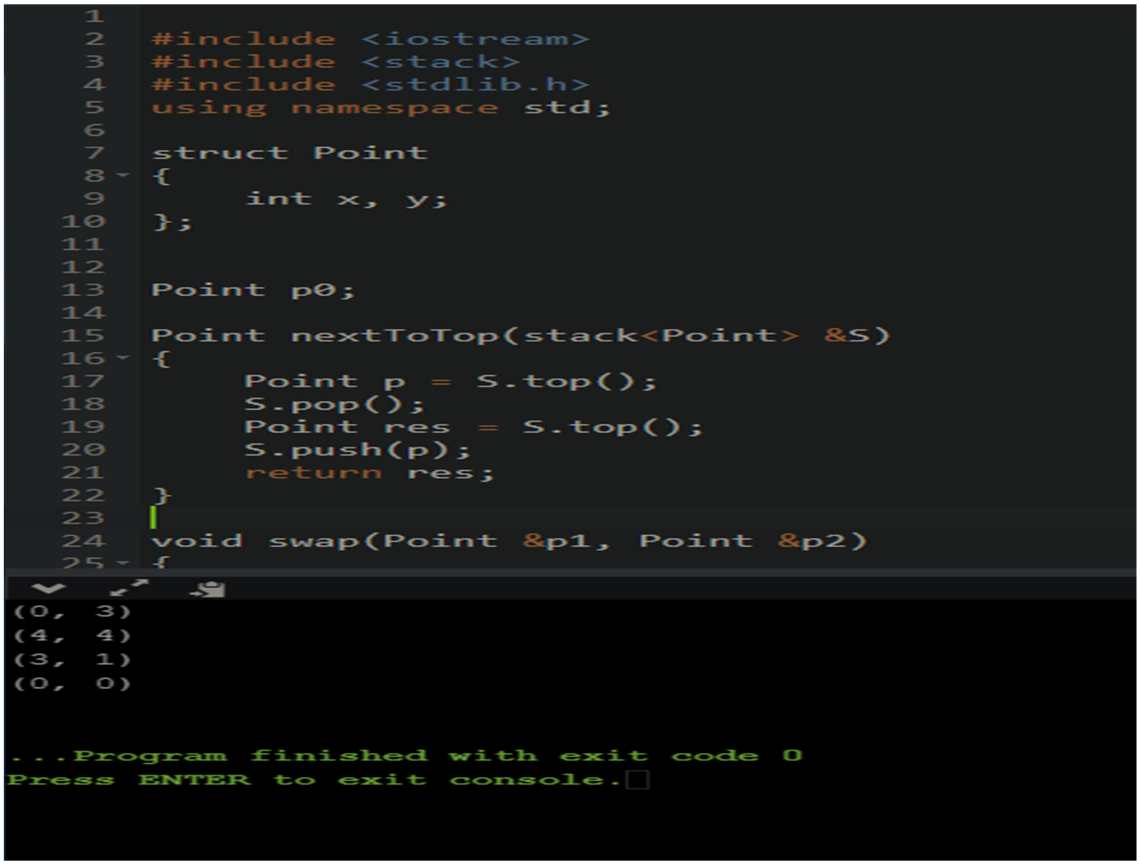
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct point
{
   double x;
   double y;
}POINT,VECTOR;

POINT b[1000];
VECTOR normal;
int n;

int upper_lower(int i, VECTOR ab, double c) {
```

```
    double x, y,result;
    y = b[i].y;
    x = normal.x*b[i].x;
    result = -(x + c) / normal.y;
    if (y>result) return 1;
    if (y == result) return 0;
    else
        return -1;

}

int ccw(VECTOR v,VECTOR v2)
{
    double cp;

    cp = v2.x*v.y - v2.y*v.x;

if (cp == abs(cp)) return 1;
else
    return -1;

}

double vector_length(VECTOR v)
{
    return sqrt(pow(v.x, 2) + pow(v.y, 2));

}

int cmp_points(const void *p1, const void *p2)
{
    const POINT *pt1 = p1;
    const POINT *pt2 = p2;
if (pt1->x > pt2->x)
    return 1;
if (pt1->x < pt2->x)
    return -1;
if (pt1->y > pt2->y)
    return 1;
if (pt1->y < pt2->y)
    return -1;
return 0;
}

int main()
{
```

```
    int i,poloha,upper[1000],lower[1000],h=0,d=0;
    scanf("%d", &n);
    if (n <= 0 && n > 1000) return 0;
    for (i = 0; i < n; i++)
    {
        scanf("%lf %lf", &b[i].x, &b[i].y);
    }
    qsort(b, n, sizeof(POINT), cmp_points);

VECTOR ab;
double c;
ab.x = b[n - 1].x - b[0].x;
ab.y = b[n - 1].y - b[0].y;
normal.x = -ab.y;
normal.y = ab.x;
c = -normal.x*b[0].x - (normal.y*b[0].y);
for (i = 0; i < n; i++)
{
    poloha = upper_lower(i,ab,c);
    if (poloha == 1) upper[h++] = i;
    if (poloha == -1) lower[d++]=i;
    if (poloha == 0)
    {
        upper[h++] = i;
        lower[d++] = i;
    }

}
int j = 0;
double v,  length = 0;
VECTOR v1, v2, v3,v4;
v3.x = 0; v3.y = 0;
for (i = 0; ; i++)
{
    int in = 0;
    if (lower[i + 2] < 0)
    {
        v1.x = b[lower[i + 1]].x - b[lower[0]].x;
        v1.y = b[lower[i + 1]].y - b[lower[0]].y;

        v2.x = b[lower[i]].x - b[lower[i + 1]].x;
        v2.y = b[lower[i]].y - b[lower[i + 1]].y;

        lenght += vector_length(v1);
        length += vector_length(v2);
        break;
```

```
    }
    v1.x = b[lower[i + 1]].x - b[lower[i]].x;
    v1.y = b[lower[i + 1]].y - b[lower[i]].y;

    v2.x = b[lower[i + 2]].x - b[lower[i]].x;
    v2.y = b[lower[i + 2]].y - b[lower[i]].y;
    in = ccw(v1, v2);
    if (in == 1)
    {
        length += vector_length(v1);
        v3 = v2;
        v4 = v1;
    }
    if (in == -1)
    {
        length -= vector_length(v4);
        if (v3.x != 0 && v3.y != 0)
        {
            length += vector_length(v3);
            v3.x = 0; v3.y = 0;
        }
        else
        {
            length += vector_length(v2);

        }


    }
}

printf("%.3lf", length);

return 0;
}
```

Output-



```cpp
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x, y;
};


Point p0;

Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

void swap(Point &p1, Point &p2)
{
```

```
(0, 3)
(4, 4)
(3, 1)
(0, 0)


...Program finished with exit code 0
Press ENTER to exit console.
```

Result-

A code has been written and the output has been verified.

Experiment 7A

Huffman Coding using Greedy

Aim-

To write a C program to implement Huffman coding using Greedy algorithm.

Algorithm-

1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

2. Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.

3. Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25.

4. Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30.

5. Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55.

6. Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100.

7. Print the resultant output.

Code-

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_TREE_HT 100

struct MinHeapNode {

    char data;

    unsigned freq;

    struct MinHeapNode *left, *right;
```

```
};
struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};
struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
        sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap
        = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHeapNode**)malloc(
        minHeap->capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}
```

```
void swapMinHeapNode(struct MinHeapNode** a,
                    struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
 void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
     if (left < minHeap->size
        && minHeap->array[left]->freq
               < minHeap->array[smallest]->freq)
        smallest = left;
     if (right < minHeap->size
        && minHeap->array[right]->freq
               < minHeap->array[smallest]->freq)
        smallest = right;
     if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
```

```
}
int isSizeOne(struct MinHeap* minHeap)
{
     return (minHeap->size == 1);
}
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}
void insertMinHeap(struct MinHeap* minHeap,
                   struct MinHeapNode* minHeapNode)

{

    ++minHeap->size;
    int i = minHeap->size - 1;
     while (i
          && minHeapNode->freq
                 < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
       i = (i - 1) / 2;
```

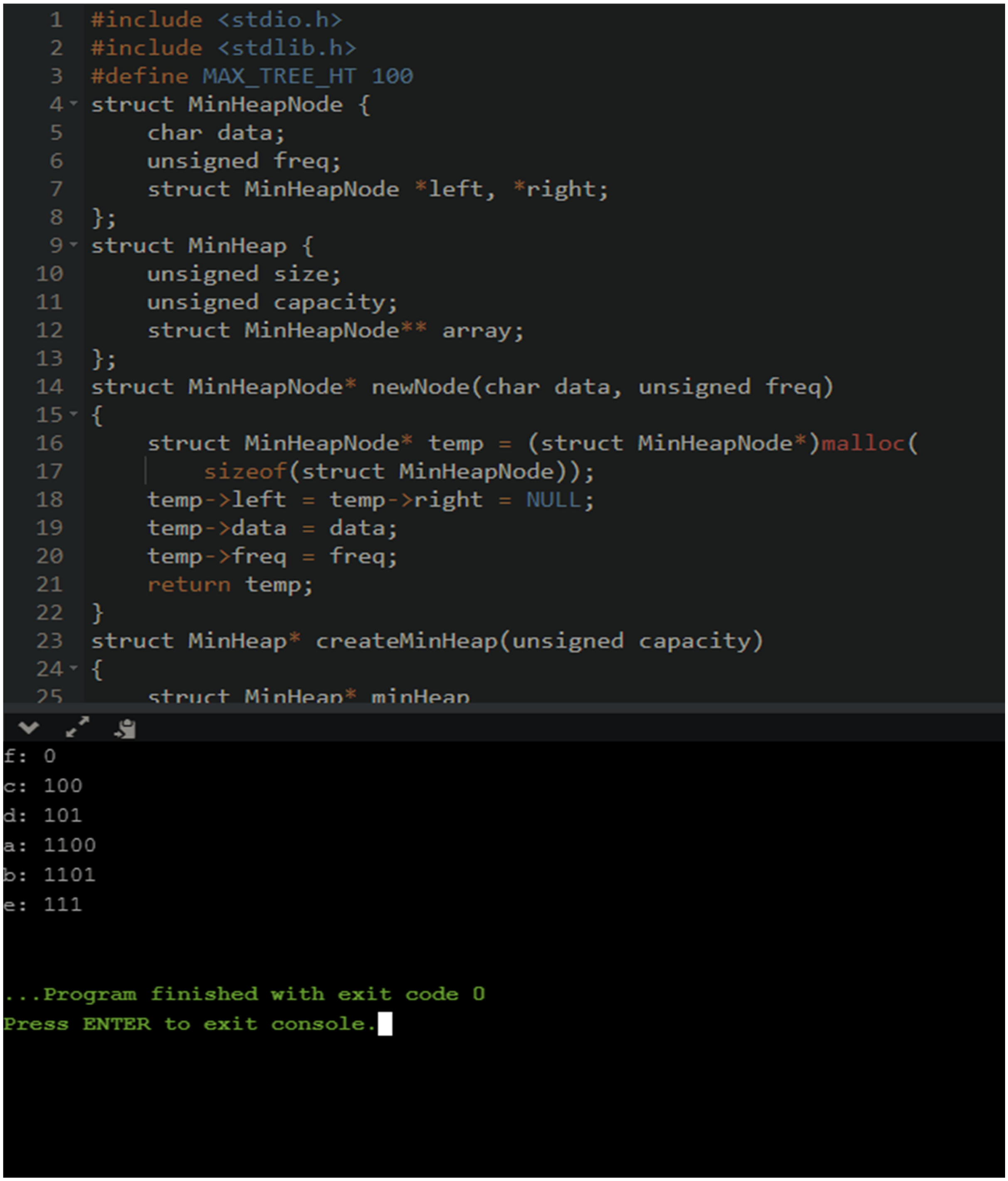
```
    }
    minHeap->array[i] = minHeapNode;
}
void buildMinHeap(struct MinHeap* minHeap)
 {
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}
int isLeaf(struct MinHeapNode* root)
{
    return !(root->left) && !(root->right);
}
struct MinHeap* createAndBuildMinHeap(char data[],
                                      int freq[], int size)
{
     struct MinHeap* minHeap = createMinHeap(size);
```

```
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
     return minHeap;
}
struct MinHeapNode* buildHuffmanTree(char data[],
                                 int freq[], int size)
 {
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap
        = createAndBuildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
         top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}
void printCodes(struct MinHeapNode* root, int arr[],
                int top)
{
```

```
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)) {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}
void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode* root
        = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}
int main()
{
     char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };
     int size = sizeof(arr) / sizeof(arr[0]);
```

```
    HuffmanCodes(arr, freq, size);

    return 0;

}
```

Output-



```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TREE_HT 100
struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};
struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};
struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
        sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap
```

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111


...Program finished with exit code 0
Press ENTER to exit console.
```

Result-

The code has been written, and the output has been verified.

Experiment 7B **Knapsack using Greedy**

Aim-

To solve knapsack problems using greedy algorithm.

Code-

```
#include<stdio.h>
int main()
{
    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
    int n,I,j;
    printf("Enter the number of items :");
    scanf("%d",&n);
   for (I = 0; I < n; i++)
   {
      printf("Enter Weight and Profit for item[%d] :\n",i);
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("Enter the capacity of knapsack :\n");
   scanf("%f",&capacity);

    for(i=0;i<n;i++)
        ratio[i]=profit[i]/weight[i];

   for (I = 0; I < n; i++)
     for (j = I + 1; j < n; j++)
```

```
    if (ratio[i] < ratio[j])
   {
      temp = ratio[j];
      ratio[j] = ratio[i];
      ratio[i] = temp;

      temp = weight[j];
      weight[j] = weight[i];
      weight[i] = temp;

      temp = profit[j];
      profit[j] = profit[i];
      profit[i] = temp;
    }

printf("Knapsack problems using Greedy Algorithm:\n");
for (I = 0; I < n; i++)
{
 if (weight[i] > capacity)
    break;
  else
 {
    Totalvalue = Totalvalue + profit[i];
    capacity = capacity – weight[i];
  }
```

```
    }
     if (I < n)
     Totalvalue = Totalvalue + (ratio[i]*capacity);
    printf("\nThe maximum value is :%f\n",Totalvalue);
    return 0;
}
```

Output-

```c
                temp = weight[j];
                weight[j] = weight[i];
                weight[i] = temp;

                temp = profit[j];
                profit[j] = profit[i];
                profit[i] = temp;
            }

    printf("Knapsack problems using Greedy Algorithm:\n");
    for (i = 0; i < n; i++)
    {
     if (weight[i] > capacity)
         break;
      else
     {
         Totalvalue = Totalvalue + profit[i];
         capacity = capacity - weight[i];
      }
    }
      if (i < n)
      Totalvalue = Totalvalue + (ratio[i]*capacity);
    printf("\nThe maximum value is :%f\n",Totalvalue);
    return 0;
}
```

```
Enter the number of items :1 2
Enter Weight and Profit for item[0] :
3 4
Enter the capacity of knapsack :
Knapsack problems using Greedy Algorithm:

The maximum value is :-1502137548800.000000


...Program finished with exit code 0
Press ENTER to exit console.
```

Result-

The code has been written, and the output has been verified.

Experiment 8A **Tree Traversal**

Aim- To write a code to demonstrate tree traversal.

Algorithm-

Inorder traversal

1.First, visit all the nodes in the left subtree

2.Then the root node

3.Visit all the nodes in the right subtree

Preorder traversal

1.Visit root node

2.Visit all the nodes in the left subtree

3.Visit all the nodes in the right subtree

Postorder traversal

1.Visit all the nodes in the left subtree

2.Visit all the nodes in the right subtree

3..Visit the root node

Code-

```
#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node* left;

    struct node* right;

};
```

```
struct node* newNode(int data)
{
        struct node* node
                = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;
        return (node);
}
void printPostorder(struct node* node)
{
        if (node == NULL)
                return;
        printPostorder(node->left);
        printPostorder(node->right);
        printf("%d ", node->data);
}
void printInorder(struct node* node)
{
        if (node == NULL)
                return;
        printInorder(node->left);
        printf("%d ", node->data);
        printInorder(node->right);
}
```

```
void printPreorder(struct node* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    printPreorder(node->left);
    printPreorder(node->right);
}
int main()
{
    struct node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("\nPreorder traversal of binary tree is \n");
    printPreorder(root);
    printf("\nInorder traversal of binary tree is \n");
    printInorder(root);
    printf("\nPostorder traversal of binary tree is \n");
    printPostorder(root);
    getchar();
    return 0;
}
```

Output-

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node* left;
    struct node* right;
};
struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return (node);
}
void printPostorder(struct node* node)
{
    if (node == NULL)
        return;
    printPostorder(node->left);
    printPostorder(node->right);
    printf("%d ", node->data);
}
void printInorder(struct node* node)
{
    if (node == NULL)
```

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```

Result-

The code has been written and the output has been verified.

Experiment 8B

Krushkal's Minimum Spanning Tree

Aim- To write a code to demonstrate Krushkal's MST algorithm.

Algorithm-

1. Sort all the edges in non-decreasing order of their weight.

2. 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.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Code-

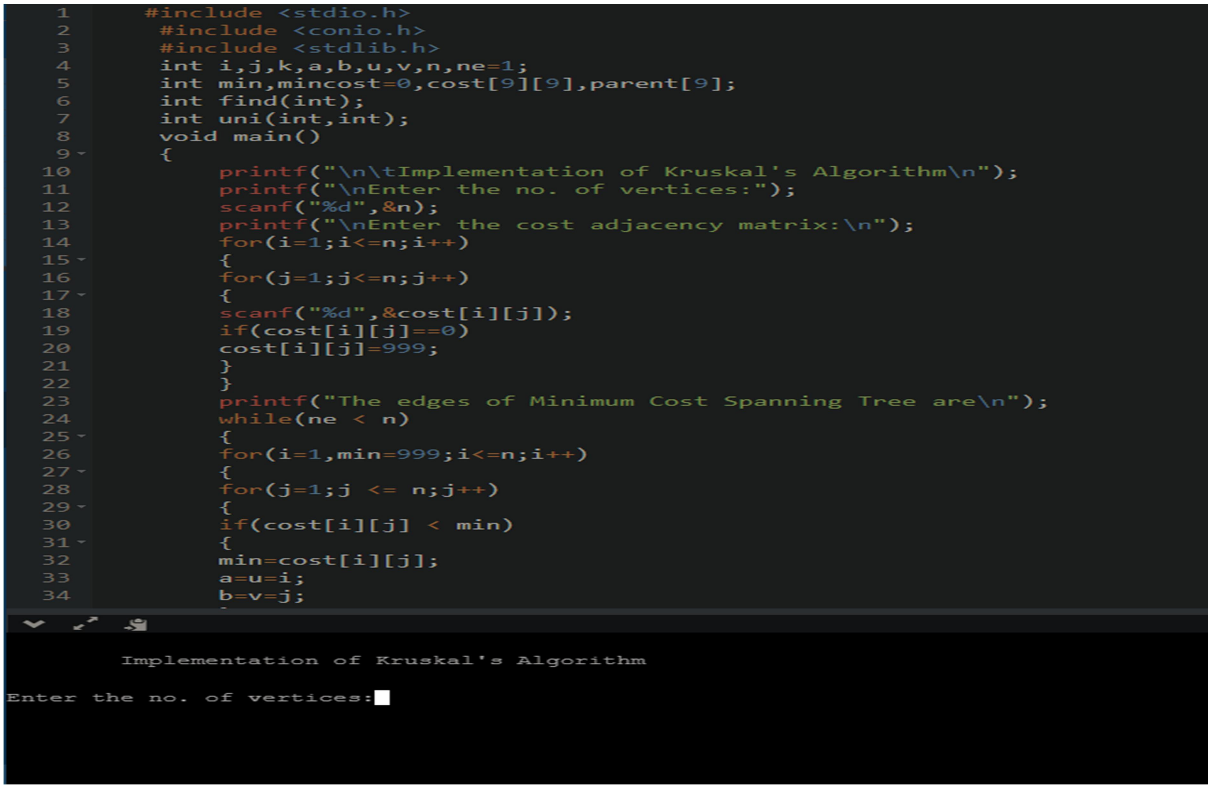
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
        printf("\n\tImplementation of Kruskal's Algorithm\n");
        printf("\nEnter the no. of vertices:");
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix:\n");
```

```
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
}
}
u=find(u);
```

```
        v=find(v);
        if(uni(u,v))
        {
        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
        mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
        parent[j]=i;
        return 1;
        }
        return 0;
```

}

Output-



Result-

The code has been written and the output has been verified.

Experiment 8C

Prim's Minimum Spanning Tree

Aim-

To write a code to demonstrate Prim's MST.

Algorithm-

1. Create edge list of given graph, with their weights.
2. Draw all nodes to create skeleton for spanning tree.
3. Select an edge with lowest weight and add it to skeleton and delete edge from edge list.
4. Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.
5. Repeat step 5 until n-1 edges are added.
6. Return.

Code-

```
#include<stdio.h>

#include<stdlib.h>

#define infinity 9999

#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()

{

        int i,j,total_cost;

        printf("Enter no. of vertices:");

        scanf("%d",&n);
```

```
	printf("\nEnter the adjacency matrix:\n");

	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&G[i][j]);

	total_cost=prims();
	printf("\nspanning tree matrix:\n");

	for(i=0;i<n;i++)
	{
		printf("\n");
		for(j=0;j<n;j++)
			printf("%d\t",spanning[i][j]);
	}

	printf("\n\nTotal cost of spanning tree=%d",total_cost);
	return 0;
}

int prims()
{
	int cost[MAX][MAX];
	int u,v,min_distance,distance[MAX],from[MAX];
	int visited[MAX],no_of_edges,i,min_cost,j;
```

```
for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
                if(G[i][j]==0)
                        cost[i][j]=infinity;
                else
                        cost[i][j]=G[i][j];
                        spanning[i][j]=0;
        }
distance[0]=0;
visited[0]=1;

for(i=1;i<n;i++)
{
        distance[i]=cost[0][i];
        from[i]=0;
        visited[i]=0;
}

min_cost=0;
no_of_edges=n-1;

while(no_of_edges>0)
{
        min_distance=infinity;
```

```
        for(i=1;i<n;i++)
            if(visited[i]==0&&distance[i]<min_distance)
            {
                v=i;
                min_distance=distance[i];
            }

        u=from[v];
        spanning[u][v]=distance[v];
        spanning[v][u]=distance[v];
        no_of_edges--;
        visited[v]=1;
        for(i=1;i<n;i++)
            if(visited[i]==0&&cost[i][v]<distance[i])
            {
                distance[i]=cost[i][v];
                from[i]=v;
            }
        min_cost=min_cost+cost[u][v];
    }
    return(min_cost);
}
```

Output-

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

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
    int i,j,total_cost;
    printf("Enter no. of vertices:");
    scanf("%d",&n);

    printf("\nEnter the adjacency matrix:\n");

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

    total_cost=prims();
    printf("\nspanning tree matrix:\n");

    for(i=0;i<n;i++)
    {
        printf("\n");
        for(j=0;j<n;j++)
            printf("%d\t",spanning[i][j]);
    }

    printf("\n\nTotal cost of spanning tree=%d",total_cost);
    return 0;
```

input

```
Enter no. of vertices:
```

Result-

The code has been written and the output has been verified.

Experiment 9

Longest Common Subsequence

Aim-

To write a C program to print the longest common susequence.

Algorithm-

1) Construct L[m+1][n+1]

2) The value L[m][n] contains length of LCS. Create a character array lcs[] of length equal to the length of lcs plus 1 (one extra to store \0).

2) Traverse the 2D array starting from L[m][n]. Do following for every cell L[i][j] …..a) If characters (in X and Y) corresponding to L[i][j] are same (Or X[i-1] == Y[j-1]), then include this character as part of LCS. …..b) Else compare values of L[i-1][j] and L[i][j-1] and go in direction of greater value.

Code-

```
#include<stdio.h>

#include<string.h>

int i,j,m,n,c[20][20];

char x[20],y[20],b[20][20];

void print(int i,int j)

{

        if(i==0 || j==0)

                return;

        if(b[i][j]=='c')

        {

                print(i-1,j-1);

                printf("%c",x[i-1]);

        }
```

```
        else if(b[i][j]=='u')
                print(i-1,j);
        else
                print(i,j-1);
}
void lcs()
{
        m=strlen(x);
        n=strlen(y);
        for(i=0;i<=m;i++)
                c[i][0]=0;
        for(i=0;i<=n;i++)
                c[0][i]=0;
        for(i=1;i<=m;i++)
                for(j=1;j<=n;j++)
                {
                        if(x[i-1]==y[j-1])
                        {
                                c[i][j]=c[i-1][j-1]+1;
                                b[i][j]='c';
                        }
                        else if(c[i-1][j]>=c[i][j-1])
                        {
                                c[i][j]=c[i-1][j];
                                b[i][j]='u';
```

```
                }
                else
                {
                        c[i][j]=c[i][j-1];
                        b[i][j]='l';
                }
            }
}

int main()
{
        printf("Enter 1st sequence:");
        scanf("%s",x);
        printf("Enter 2nd sequence:");
        scanf("%s",y);
        printf("\nThe Longest Common Subsequence is ");
        lcs();
        print(m,n);
            return 0;
}
```

Output-

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

 int i, j, m, n, c[20][20];

char x[20], y[20], b[20][20];


void
print (int i, int j)
{

if (i == 0 || j == 0)

return;

if (b[i][j] == 'c')

     {

print (i - 1, j - 1);

printf ("%c", x[i - 1]);

}

  else if (b[i][j] == 'u')

print (i - 1, j);

  else

print (i, j - 1);

```

```
Enter 1st sequence:
```

Result-

The code has been written and the output has been verified.

Experiment 10

N Queen problem

Aim-

To write a C program to display and explain the N Queens problem.

Algorithm-

1) Start in the leftmost column

2) If all queens are placed

   return true

3) Try all rows in the current column.

   Do following for every tried row.

   a) If the queen can be placed safely in this row

      then mark this [row, column] as part of the

      solution and recursively check if placing

      queen here leads to a solution.

   b) If placing the queen in [row, column] leads to

      a solution then return true.

   c) If placing queen doesn't lead to a solution then

      unmark this [row, column] (Backtrack) and go to

      step (a) to try other rows.

3) If all rows have been tried and nothing worked,

   return false to trigger backtracking.

Code-

```
#include<stdio.h>

#include<math.h>
```

```
int board[20],count;

int main()
{
 int n,i,j;
 void queen(int row,int n);

 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}
void print(int n)
{
 int i,j;
 printf("\n\nSolution %d:\n\n",++count);

 for(i=1;i<=n;++i)
  printf("\t%d",i);

 for(i=1;i<=n;++i)
 {
  printf("\n\n%d",i);
```

```
  for(j=1;j<=n;++j) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
 }
}
int place(int row,int column)
{
 int i;
 for(i=1;i<=row-1;++i)
 {
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
 }

 return 1; //no conflicts
}
void queen(int row,int n)
{
```

```
 int column;
 for(column=1;column<=n;++column)
 {
  if(place(row,column))
  {
   board[row]=column;
   if(row==n)
    print(n);
   else
    queen(row+1,n);
  }
 }
}
```

Output-

```c
#include<stdio.h>
#include<math.h>

int board[20],count;

int main()
{
 int n,i,j;
 void queen(int row,int n);

 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}
void print(int n)
{
 int i,j;
 printf("\n\nSolution %d:\n\n",++count);

 for(i=1;i<=n;++i)
  printf("\t%d",i);

 for(i=1;i<=n;++i)
 {
  printf("\n\n%d",i);
  for(j=1;j<=n;++j) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
```

input

```
main.c:45:7: warning: implicit declaration of function ‘abs’ [-Wimplicit-function-declaration]
 - N Queens Problem Using Backtracking -

Enter number of Queens:
```

Result-

The code has been written and the output has been verified.

EX.NO.11

TRAVELLING SALESMAN PROBLEM

Aim:

To write a C program to solve the travelling sales man problem using the dynamic programming approach.

Algorithm:

Step1: Start the process
Step2: Enter the number of cities
Step3: Enter the cost matrix of all the cities
Step4: Find all possible feasible solutions by taking the permutation of the cities which is to be covered.
Step5: Find the cost of each path using the cost matrix.
Step6: Find out the path with minimum cost.
Step7: If more than one path having the same cost considers the first occurring path.
Step8: That is selected as the optimum solution.
Step9: Stop the process.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
int i,j;
printf("\n\nEnter Number of Cities: ");
scanf("%d",&n);
printf("\nEnter Cost Matrix: \n");
for( i=0;i<n;i++)
{
printf("\n Enter Elements of Row # : %d\n",i+1);
for( j=0;j<n;j++)
scanf("%d",&a[i][j]);
visited[i]=0;
}
printf("\n\nThe Cost Matrix is:\n");
for( i=0;i<n;i++)
{
printf("\n\n");
for(j=0;j<n;j++)
printf("\t%d",a[i][j]);
}
}
```

```
void mincost(int city)
{
int i,ncity,least(int city);
visited[city]=1;
printf("%d ===> ",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=a[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i<n;i++)
{
if((a[c][i]!=0)&&(visited[i]==0))
if(a[c][i]<min)
{
min=a[i][0]+a[c][i];
kmin=a[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
void put()
{
printf("\n\nMinimum cost:");
printf("%d",cost);
}
void main()
{
clrscr();
get();
printf("\n\nThe Path is:\n\n");
mincost(0);
put();
getch();
```

}

OUTPUT

```
The Cost Matrix is:

        14      15      18      34      98      12

        45      67      81      34      12      8

        23      86      15      3       57      34

        47      92      31      7       68      39

        95      85      10      55      72      43

        63      86      93      56      13      49

The Path is:

1 ===> 6 ===> 5 ===> 3 ===> 4 ===> 2 ===> 1

Minimum cost:175_
```

RESULT

Thus the travelling Salesman problem using the dynamic programming approach was executed successfully.

EX.NO.12a

BFS Implementation with Array

Aim:

To write a C program to solve the BFS problem with array.

Algorithm:

Step1: Start by putting any one of the graph's vertices at the back of a queue.
Step2: Take the front item of the queue and add it to the visited list.
Step3: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
Step4: Keep repeating steps 2 and 3 until the queue is empty.
Step5: The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);
int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
        create_graph();
        BF_Traversal();
        return 0;
}

void BF_Traversal()
{
```

```
	int v;

	for(v=0; v<n; v++)
		state[v] = initial;

	printf("Enter Start Vertex for BFS: \n");
	scanf("%d", &v);
	BFS(v);
}

void BFS(int v)
{
	int i;
	insert_queue(v);
	state[v] = waiting;
	while(!isEmpty_queue())
	{
		v = delete_queue( );
		printf("%d ",v);
		state[v] = visited;

		for(i=0; i<n; i++)
		{
			if(adj[v][i] == 1 && state[i] == initial)
			{
				insert_queue(i);
				state[i] = waiting;
			}
		}
	}
	printf("\n");
}

void insert_queue(int vertex)
{
	if(rear == MAX-1)
		printf("Queue Overflow\n");
	else
	{
		if(front == -1)
			front = 0;
		rear = rear+1;
		queue[rear] = vertex ;
	}
}
```

```
int isEmpty_queue()
{
	if(front == -1 || front > rear)
		return 1;
	else
		return 0;
}

int delete_queue()
{
	int delete_item;
	if(front == -1 || front > rear)
	{
		printf("Queue Underflow\n");
		exit(1);
	}

	delete_item = queue[front];
	front = front+1;
	return delete_item;
}

void create_graph()
{
	int count,max_edge,origin,destin;

	printf("Enter number of vertices : ");
	scanf("%d",&n);
	max_edge = n*(n-1);

	for(count=1; count<=max_edge; count++)
	{
		printf("Enter edge %d( -1 -1 to quit ) : ",count);
		scanf("%d %d",&origin,&destin);

		if((origin == -1) && (destin == -1))
			break;

		if(origin>=n || destin>=n || origin<0 || destin<0)
		{
			printf("Invalid edge!\n");
			count--;
		}
		else
		{
			adj[origin][destin] = 1;
```

```
                    }
            }
    }
```

OUTPUT

```
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$ ./a.out
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0
1
Enter edge 2( -1 -1 to quit ) : 0
3
Enter edge 3( -1 -1 to quit ) : 0
4
Enter edge 4( -1 -1 to quit ) : 1
2
Enter edge 5( -1 -1 to quit ) : 3
6
Enter edge 6( -1 -1 to quit ) : 4
7
Enter edge 7( -1 -1 to quit ) : 6
4
Enter edge 8( -1 -1 to quit ) : 6
7
Enter edge 9( -1 -1 to quit ) : 2
5
Enter edge 10( -1 -1 to quit ) : 4
5
Enter edge 11( -1 -1 to quit ) : 7
5
Enter edge 12( -1 -1 to quit ) : 7
8
Enter edge 13( -1 -1 to quit ) : -1
-1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$
```

RESULT

Thus the BFS problem using array was executed successfully.

EX.NO.12b

DFS Implementation with Array

Aim:

To write a C program to solve the DFS problem with array implementation.

Algorithm:

Step1: Start by putting any one of the graph's vertices on top of a stack.
Step2: Take the top item of the stack and add it to the visited list.
Step3: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
Step4: Keep repeating steps 2 and 3 until the stack is empty.

Program:

```
#include <stdio.h>
#include <stdlib.h>
/*          ADJACENCY MATRIX                              */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
   int j;
   visited[i]=1;
   printf(" %d->",i+1);
   for(j=0;j<V;j++)
   {
      if(G[i][j]==1&&visited[j]==0)
         DFS(j);
   }
}
int main()
{
   int i,j,v1,v2;
   printf("\t\t\tGraphs\n");
   printf("Enter the no of edges:");
   scanf("%d",&E);
   printf("Enter the no of vertices:");
   scanf("%d",&V);
   for(i=0;i<V;i++)
   {
      for(j=0;j<V;j++)
         G[i][j]=0;
   }
   /*    creating edges :P    */
   for(i=0;i<E;i++)
```

```
    {
        printf("Enter the edges (format: V1 V2) : ");
        scanf("%d%d",&v1,&v2);
        G[v1-1][v2-1]=1;
    }
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
            printf(" %d ",G[i][j]);
        printf("\n");
    }
    printf("Enter the source: ");
    scanf("%d",&source);
        DFS(source-1);
    return 0;
}
```

Output:

```
E:\CodeBlocks\share\CodeBlocks\graphs\bin\Debug\graphs.exe
                              Graphs
Enter the no of edges:11
Enter the no of vertices:10
Enter the edges (format: V1 V2) : 1 2
Enter the edges (format: V1 V2) : 1 3
Enter the edges (format: V1 V2) : 2 4
Enter the edges (format: V1 V2) : 2 5
Enter the edges (format: V1 V2) : 3 6
Enter the edges (format: V1 V2) : 3 7
Enter the edges (format: V1 V2) : 4 8
Enter the edges (format: V1 V2) : 5 9
Enter the edges (format: V1 V2) : 6 10
Enter the edges (format: V1 V2) : 8 9
Enter the edges (format: V1 V2) : 9 10
 0  1  1  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  1  1  0  0  0
 0  0  0  0  0  0  0  1  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  0  0  0
Enter the source: 1
 1-> 2-> 4-> 8-> 9-> 10-> 5-> 3-> 6-> 7->
Process returned 0 (0x0)   execution time : 42.232 s
Press any key to continue.
```

RESULT

Thus the BFS problem using array was executed successfully.

EX.NO.13

Randomized Quick Sort

Aim:

To write a C program to solve the Randomized quick sort.

Algorithm:

QUICKSORT(A,p,r)
    if p < r
    then q PARTITION(A,p,r)
    QUICKSORT(A,p,q)
    QUICKSORT(A,q + 1,r)
To sort an entire array A, the initial call is QUICKSORT(A, 1, length[A]).

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray A[p . . r] in place.

PARTITION(A,p,r)
x ← A[p]
I ← p - 1
j ← r + 1
    while TRUE
do repeat j ← j - 1
until A[j] ≤ x
    repeat i i + 1
until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
    else return j

RANDOMIZED-QUICKSORT(A,p,r)
    if p < r
    then q ← RANDOMIZED-PARTITION(A,p,r)
    RANDOMIZED-QUICKSORT(A,p,q)
    RANDOMIZED-QUICKSORT(A,q + 1,r)

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void random_shuffle(int arr[])
{
   srand(time(NULL));
   int i, j, temp;
```

```
    for (i = MAX - 1; i > 0; i--)
    {
        j = rand()%(i + 1);
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int partion(int arr[], int p, int r)
{
    int pivotIndex = p + rand()%(r - p + 1); //generates a random number as a pivot
    int pivot;
    int i = p - 1;
    int j;
    pivot = arr[pivotIndex];
    swap(&arr[pivotIndex], &arr[r]);
    for (j = p; j < r; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }

    }
    swap(&arr[i+1], &arr[r]);
    return i + 1;
}

void quick_sort(int arr[], int p, int q)
{
    int j;
    if (p < q)
    {
        j = partion(arr, p, q);
        quick_sort(arr, p, j-1);
        quick_sort(arr, j+1, q);
    }
```

```
}
int main()
{
    int i;
    int arr[MAX];
    for (i = 0; i < MAX; i++)
        arr[i] = i;
    random_shuffle(arr); //To randomize the array
    quick_sort(arr, 0, MAX-1); //function to sort the elements of array
    for (i = 0; i < MAX; i++)
         printf("%d \n", arr[i]);
    return 0;
}
```

Output:

```
$ gcc randomizedquicksort.c -o randomizedquicksort
$ ./randomizedquicksort
```

Sorted array is : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

RESULT

Thus the Randomized quick sort problem using array was executed successfully.

EX.NO.14

String Matching Algorithm

Aim:

To write a C program to solve the String Matching algorithm.

Algorithm:

```
Start
   pat_len := pattern Size
   str_len := string size
   for i := 0 to (str_len - pat_len), do
      for j := 0 to pat_len, do
         if text[i+j] ≠ pattern[j], then
         break
   if j == patLen, then
   display the position i, as there pattern found
End
```

Program:

```
#include <stdio.h>
#include <string.h>
int match(char [], char []);

int main() {
  char a[100], b[100];
  int position;

  printf("Enter some text\n");
  gets(a);

  printf("Enter a string to find\n");
  gets(b);

  position = match(a, b);

  if (position != -1) {
    printf("Found at location: %d\n", position + 1);
  }
  else {
    printf("Not found.\n");
  }

  return 0;
}
```

```
int match(char text[], char pattern[]) {
  int c, d, e, text_length, pattern_length, position = -1;

  text_length    = strlen(text);
  pattern_length = strlen(pattern);

  if (pattern_length > text_length) {
    return -1;
  }

  for (c = 0; c <= text_length - pattern_length; c++) {
    position = e = c;

    for (d = 0; d < pattern_length; d++) {
      if (pattern[d] == text[e]) {
        e++;
      }
      else {
        break;
      }
    }
    if (d == pattern_length) {
      return position;
    }
  }

  return -1;
}
```

Output

RESULT

Thus the String matching algorithm was executed successfully.

EX.NO.15

Analysing - Real Time Problem

Aim:

To write a C program to sort an array using Merge sort and manipulate the time complexity of the program.

Algorithm:

Step1: Mergesort(A[O .. n - 1])

Step2: Sorts array A[O .. n - 1] by recursive mergesort

Step3: Input: An array A[O .. n - 1] of orderable elements

Step4: Output: Array A[O .. n - 1] sorted in nondecreasing order

Step5: Merge(B[O .. p- 1], C[O .. q -1], A[O.. p + q -1])

Step6: Merges two sorted arrays into one sorted array

Program:

```
#include <stdio.h>

 #include <stdlib.h>

 #include <sys/time.h>

 #include <omp.h>

void simplemerge(int a[], int low, int mid, int high)

{

    int i,j,k,c[20000];

    i=low;

    j=mid+1;

    k=low;

    int tid;

    omp_set_num_threads(10);
```

```
{
   tid=omp_get_thread_num();
   while(i<=mid&&j<=high)
   {
      if(a[i] < a[j])
      {
         c[k]=a[i];
         //printf("%d%d",tid,c[k]);
         i++;
         k++;
      }
      else
      {
         c[k]=a[j];
         //printf("%d%d", tid, c[k]);
         j++;
         k++;
      }
   }
}
while(i<=mid)
{
   c[k]=a[i];
   i++;
   k++;
```

```
    }
    while(j<=high)
    {
        c[k]=a[j];
        j++;
        k++;
    }
    for(k=low;k<=high;k++)
    a[k]=c[k];
}
void merge(int a[],int low,int high)
{
    int mid;
    if(low < high)
    {
        mid=(low+high)/2;
        merge(a,low,mid);
        merge(a,mid+1,high);
        simplemerge(a,low,mid,high);
    }
}
void getnumber(int a[], int n)
{
    int i;
    for(i=0;i < n;i++)
```

```
        a[i]=rand()%100;
    }
    int main()
    {
        FILE *fp;
        int a[2000],i;
        struct timeval tv;
        double start, end, elapse;
        fp=fopen("mergesort.txt","w");
        for(i=10;i<=1000;i+=10)
        {
            getnumber(a,i);
            gettimeofday(&tv,NULL);
            start=tv.tv_sec+(tv.tv_usec/1000000.0);
            merge(a,0,i-1);
            gettimeofday(&tv,NULL);
            end=tv.tv_sec+(tv.tv_usec/1000000.0);
            elapse=end-start;
            fprintf(fp,"%d\t%lf\n",i,elapse);
        }
        fclose(fp);
        system("gnuplot");
        return 0;
    }
```

mergesort.gpl

Gnuplot script file for plotting data in file "mergesort.txt" This file is called mergesort.gpl

set terminal png font arial

set title "Time Complexity for Merge Sort"

set autoscale

set xlabel "Size of Input"

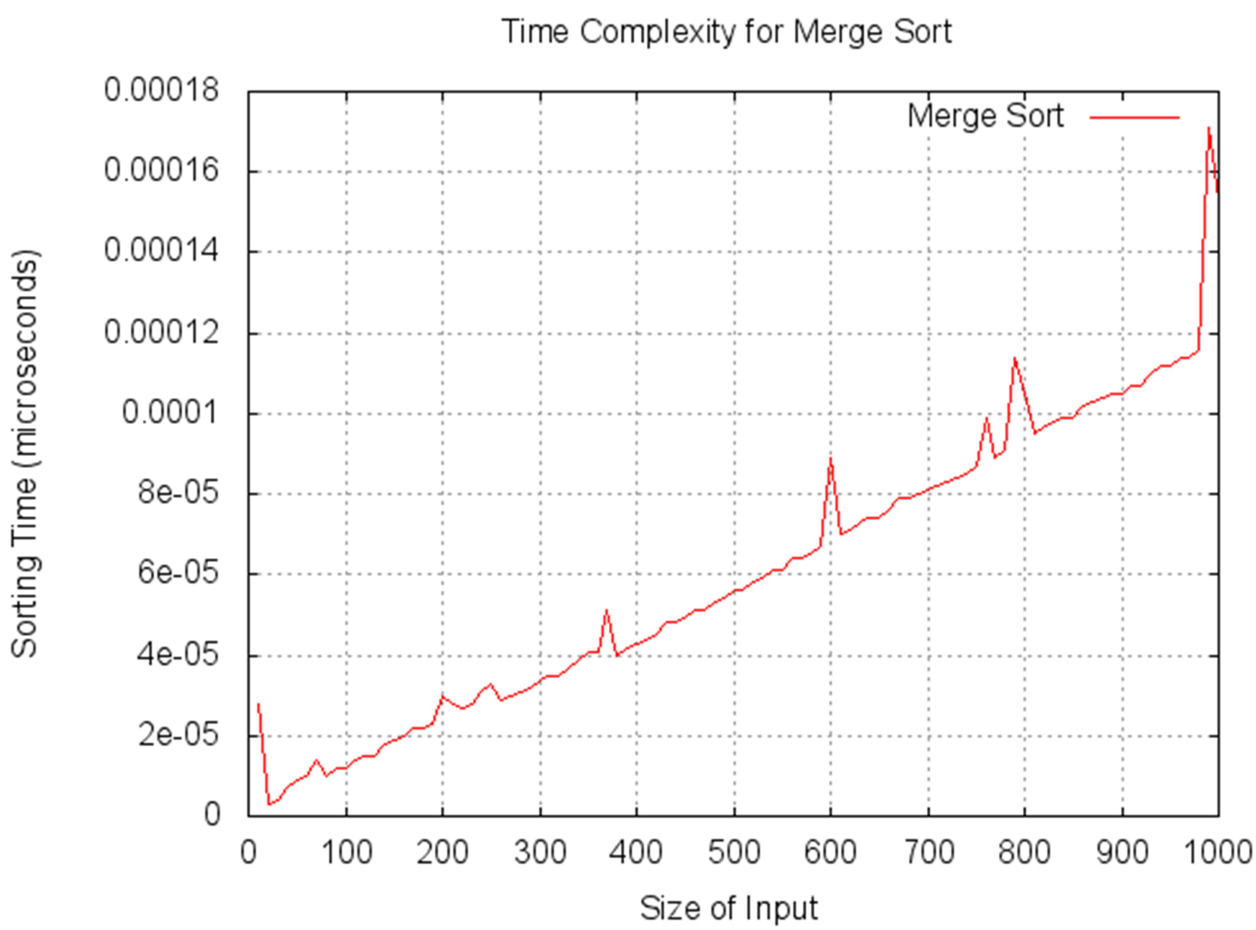
set ylabel "Sorting Time (microseconds)"

set grid

set output "mergesort.png"

plot "mergesort.txt" t "Merge Sort" with lines

OUTPUT



RESULT

Thus the merge sort program was executed successfully with the time complexity.