Experiment 1

PROGRAM

```
/*
* Approach : The idea is to compare x with the last element in arr[].
* If an element is found at the last position, return it.
* Else recur elmntSrch() for remaining array and element x.
*/

#include <stdio.h>
// Recursive function to search x in arr[]
int elmntSrch(int arr[], int size, int x) {
	int rec;
	size--;
	if (size >= 0) {
	if (arr[size] == x)
		return size;
	else
		rec = elmntSrch(arr, size, x);
	}
	else
		return -1;
	return rec;
}
int main(void) {
	int arr[] = {12, 34, 54, 2, 3};
	int size = sizeof(arr) / sizeof(arr[0]);
	int x = 3;
	int indx;
	indx = elmntSrch(arr, size, x);
	if (indx != -1)
		printf("Element %d is present at index %d", x, indx);
	else
		printf("Element %d is not present", x);
	return 0;
}
```

Output

Element 3 is present at index 4

<u>**Experiment 2**</u>

PROGRAM:

```
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
int main() {
	int a[30],n,i;
	printf("Enter no of elements:");
	scanf("%d",&n);
	printf("Enter array elements:");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);
		mergesort(a,0,n-1);
		printf("\nSorted array is :");
	for(i=0;i<n;i++)
		printf("%d ",a[i]);
	return 0;
}
void mergesort(int a[],int i,int j) {
	int mid;
	if(i<j) {
		mid=(i+j)/2;
		mergesort(a,i,mid); //left recursion
		mergesort(a,mid+1,j); //right recursion
		merge(a,i,mid,mid+1,j); //merging of two sorted sub-arrays
	}
}
void merge(int a[],int i1,int j1,int i2,int j2) {
int temp[50]; //array used for merging
	int i,j,k;
	i=i1; //beginning of the first list
	j=i2; //beginning of the second list
	k=0;
	while(i<=j1 && j<=j2) //while elements in both lists
	{
		if(a[i]<a[j])
		temp[k++]=a[i++];
		else
		temp[k++]=a[j++];
	}
	while(i<=j1) //copy remaining elements of the first list
		temp[k++]=a[i++];
	while(j<=j2) //copy remaining elements of the second list
		temp[k++]=a[j++];
	//Transfer elements from temp[] back to a[]
	for(i=i1,j=0;i<=j2;i++,j++)
		a[i]=temp[j];
}
```

Output
Enter Number of Element 5
Enter array element 7 5 9 4 1
Sorted array is 1 4 5 7 9

Experiment 3

Algorithm Quicksort (p, q)

```
// Sorts the elements a[p],……,a[q] which reside in the global array a[1 : n] into
ascending order.
{
        if( p < r) then // if there are more than one element
        {
                // divide P into two sub problems
                j=Partition (a, p,q+1);
                // j is the position of the partitioning element
                // solve the sub problems
                QuickSort (p, j 1);
                QuickSort (j + 1, q);
        }
}
```

Partitioning the Array

Partitioning procedure rearranges the sub arrays inplace.

```
Algorithm Partition (a, m, p)
// Within a[m],a[m+1],….,a[p1]
the elements are rearranged in such a manner that if
initially t=a[m] then after completion a[q]=t for some q between m and p1,
a[k] |<=t for
m<=k<q, and a[k]>=t for q<k<p. q is returned. Set a[p]= ∞.
{
        v = a[m]; i=m; j=p;
        repeat
        {
                repeat i=i+1;
                until a[i] >=v;
                repeat j = j1;
                until a[j] <=v;
                if i < j then exchange A[i] ↔ A[j]
        } until ( i>= j);
        a[m]=a[j]; a[j]=v; return j;
}
```

Experiment 3

PROGRAM:

```
/*c program for quick sorting*/
#include<stdio.h>
#include<conio.h>
void qsort(int arr[20], int fst, int last);
int main() {
	int arr[30];
	int i,size;
	printf("Enter total no. of the elements : ");
	scanf("%d",&size);
	printf("Enter total %d elements : \n",size);
	for(i=0; i<size; i++)
		scanf("%d",&arr[i]);
		qsort(arr,0,size-1);
		printf("Quick sorted elements are as : \n");
	for(i=0; i<size; i++)
		printf("%d\t",arr[i]);
	getch();
	return 0;
}
void qsort(int arr[20], int fst, int last) {
	int i,j,pivot,tmp;
	if(fst<last) {
		pivot=fst;
		i=fst;
		j=last;
		while(i< j) {
			while(arr[i]<=arr[pivot] && i<last)
				i++;
			while(arr[j]>arr[pivot])
				j--;
			if(i<j) {
				tmp=arr[i];
				arr[i]=arr[j];
				arr[j]=tmp;
			}
		}
		tmp=arr[pivot];
		arr[pivot]=arr[j];
		arr[j]=tmp;
		qsort(arr,fst,j-1);
		qsort(arr,j+1,last);
	}
}
```

Output -

```
Enter total no. of the elements : 5
Enter total -28787 elements :
Quick sorted elements are as :
Enter total no. of the elements : 5
Enter total 5 elements :
7 6 1 3 4
```

Quick sorted elements are as:
1 3 4 6 7

Experiment 4

```
// A divide and conquer program in C/C++ to find the smallest distance from a
// given set of points.

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>

 // A structure to represent a Point in 2D plane
struct Point {
	int x, y;
};
/* Following two functions are needed for library function qsort().
Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
// Needed to sort array of points according to X coordinate
int compareX(const void * a,
	const void * b) {
	Point * p1 = (Point * ) a, * p2 = (Point * ) b;
	return (p1 -> x - p2 -> x);
}
// Needed to sort array of points according to Y coordinate
int compareY(const void * a,
	const void * b) {
	Point * p1 = (Point * ) a, * p2 = (Point * ) b;
	return (p1 -> y - p2 -> y);
}
// A utility function to find the distance between two points
float dist(Point p1, Point p2) {
	return sqrt((p1.x - p2.x) * (p1.x - p2.x) +
		(p1.y - p2.y) * (p1.y - p2.y)
	);
}
// A Brute Force method to return the smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n) {
	float min = FLT_MAX;
	for (int i = 0; i < n; ++i)
		for (int j = i + 1; j < n; ++j)
			if (dist(P[i], P[j]) < min)
				min = dist(P[i], P[j]);
	return min;
}
// A utility function to find a minimum of two float values
float min(float x, float y) {
	return (x < y) ? x : y;
}
// A utility function to find the distance between the closest points of
// strip of a given size. All points in strip[] are sorted according to
// y coordinate. They all have an upper bound on minimum distance as d.
// Note that this method seems to be a O(n^2) method, but it's a O(n)
// method as the inner loop runs at most 6 times
```

```
float stripClosest(Point strip[], int size, float d) {
	float min = d; // Initialize the minimum distance as d
	qsort(strip, size, sizeof(Point), compareY);
	// Pick all points one by one and try the next points till the difference
	// between y coordinates is smaller than d.
	// This is a proven fact that this loop runs at most 6 times
	for (int i = 0; i < size; ++i)
		for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j)
			if (dist(strip[i], strip[j]) < min)
				min = dist(strip[i], strip[j]);
	return min;
}
// A recursive function to find the smallest distance. The array P contains
// all points sorted according to x coordinate
float closestUtil(Point P[], int n) {
	// If there are 2 or 3 points, then use brute force
	if (n <= 3)
		return bruteForce(P, n);
	// Find the middle point
	int mid = n / 2;
	Point midPoint = P[mid];
	// Consider the vertical line passing through the middle point
	// calculate the smallest distance dl on left of middle point and
	// dr on right side
	float dl = closestUtil(P, mid);
	float dr = closestUtil(P + mid, n - mid);
	// Find the smaller of two distances
	float d = min(dl, dr);
	// Build an array strip[] that contains points close (closer than d)
	// to the line passing through the middle point
	Point strip[n];
	int j = 0;
	for (int i = 0; i < n; i++)
		if (abs(P[i].x - midPoint.x) < d)
			strip[j] = P[i], j++;
	// Find the closest points in strip. Return the minimum of d and closest
	// distance is strip[]
	return min(d, stripClosest(strip, j, d));
}
// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n) {
	qsort(P, n, sizeof(Point), compareX);
	// Use recursive function closestUtil() to find the smallest distance
	return closestUtil(P, n);
}
// Driver program to test above functions
int main() {
	Point P[] = {
		{
			2,
			3
		},
```

```
		{
			12,
			30
		},
		{
			40,
			50
		},
		{
			5,
			1
		},
		{
			12,
			10
		},
		{
			3,
			4
		}
	};
	int n = sizeof(P) / sizeof(P[0]);
	printf("The smallest distance is %f ", closest(P, n));
	return 0;
}
```

Output

The smallest distance is 1.414214

Experiment 5

PROGRAM

```
/* A Naive recursive implementation
of 0-1 Knapsack problem */
#include <stdio.h>
 // A utility function that returns maximum of two integers
int max(int a, int b) {
	return (a > b) ? a : b;
}
// Returns the maximum value that can be
// put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n) {
	// Base Case
	if (n == 0 || W == 0)
		return 0;
	// If weight of the nth item is more than
	// Knapsack capacity W, then this item cannot
	// be included in the optimal solution
	if (wt[n - 1] > W)
		return knapSack(W, wt, val, n - 1);
	// Return the maximum of two cases:
	// (1) nth item included
	// (2) not included
	else
		return max(
			val[n - 1] +
			knapSack(W - wt[n - 1],
				wt, val, n - 1),
			knapSack(W, wt, val, n - 1));
}
// Driver program to test above function
int main() {
	int val[] = {		60,		100,		120	};
	int wt[] = {		10,		20,		30	};
	int W = 50;
	int n = sizeof(val) / sizeof(val[0]);
	printf("%d", knapSack(W, wt, val, n));
	return 0;
}
```

Output

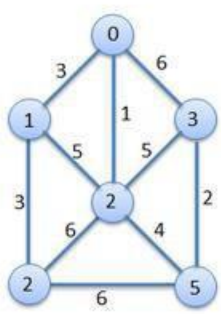
220

Experiment 6

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
    A ← Ø
    for each vertex v  V[G]
        do MAKE-SET(v)
    sort the edges of E into nondecreasing order by weight w
    for each edge (u, v)  E, taken in nondecreasing order by wei
        do if FIND-SET(u) ≠ FIND-SET(v)
            then A ← A  {(u, v)}
                UNION(u, v)
    return A
```

Example



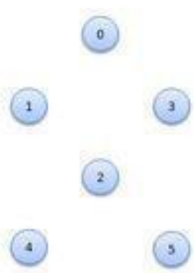
Procedure for finding Minimum Spanning Tree

Step1. Edges are sorted in ascending order by weight.

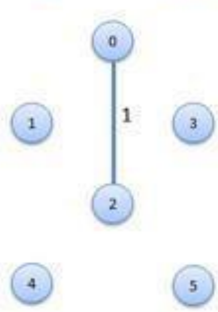
Edge No.	Vertex Pair	Edge Weight
E1	(0,2)	1
E2	(3,5)	2
E3	(0,1)	3
E4	(1,4)	3
E5	(2,5)	4
E6	(1,2)	5
E7	(2,3)	5
E8	(0,3)	6
E9	(2,4)	6
E10	(4,5)	6

Step2. Edges are added in sequence.

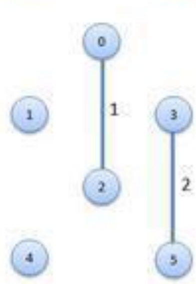
Graph



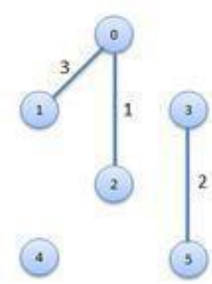
Add Edge E1



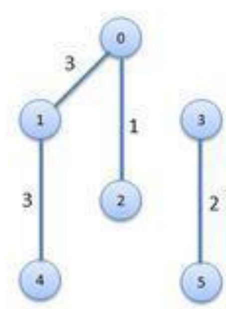
Add Edge E2



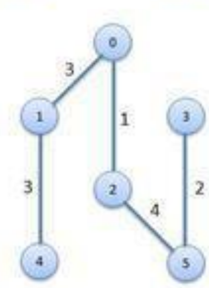
Add Edge E3



Add Edge E4



Add Edge E5



Total Cost = 1+2+3+3+4 = 13s

Program:

```
#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() {
	clrscr();
	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:

```
Enter the no. of vertices:6
Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
3 0 6 0 0 6
0 0 4 2 6 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4
Minimum cost = 13
```

Experiment 7

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

int a,b,u,v,n,i,j,ne=1;
int visited[10] = {0}, min, mincost = 0, cost[10][10];
void main() {
	clrscr();
	printf("\nEnter the number of nodes:");
	scanf("%d", & n);
	printf("\nEnter the adjacency matrix:\n");
	for (i = 1; i <= n; i++)
		for (j = 1; j <= n; j++) {
			scanf("%d", & cost[i][j]);
			if (cost[i][j] == 0)
				cost[i][j] = 999;
		}
	visited[1] = 1;
	printf("\n");
	while (ne < n) {
		for (i = 1, min = 999; i <= n; i++)
			for (j = 1; j <= n; j++)
				if (cost[i][j] < min)
					if (visited[i] != 0) {
						min = cost[i][j];
						a = u = i;
						b = v = j;
					}
		if (visited[u] == 0 || visited[v] == 0)

		{
			printf("\n Edge %d:(%d %d) cost:%d", ne++, a, b, min);
			mincost += min;
			visited[b] = 1;
		}
		cost[a][b] = cost[b][a] = 999;
	}
	printf("\n Minimun cost=%d", mincost);
	getch();
}
```
Output

<u>**Experiment 8**</u>

Program

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

#define V 4 // Number of vertices
/* Define Infinite as a large enough value. This value will be used
for vertices not connected to each other */
#define INF 999
void printSolution(int dist[][V]);
void floydWarshall(int graph[][V]) {
	int dist[V][V], i, j, k;
	for (i = 0; i < V; i++)
		for (j = 0; j < V; j++)
			dist[i][j] = graph[i][j];
	for (k = 0; k < V; k++) {
		for (i = 0; i < V; i++) {
			for (j = 0; j < V; j++) {
				if (dist[i][k] + dist[k][j] < dist[i][j])
					dist[i][j] = dist[i][k] + dist[k][j];
			}
		}
	}
	printSolution(dist);
}
void printSolution(int dist[][V]) {
	printf("The following matrix shows shortest distances between every pair of vertices \n\n");
	for (int i = 0; i < V; i++) {
		for (int j = 0; j < V; j++) {
			if (dist[i][j] == INF)
				printf("%7s", "INF");
			else
				printf("%7d", dist[i][j]);
		}
		printf("\n");
	}
}
int main() {
	clrscr();
	int graph[V][V] = {
		{
			0,
			5,
			INF,
			10
		},
		{
			INF,
			0,
			3,
			INF
		},
```

```
        {
            INF,
            INF,
            0,
            1
        },
        {
            INF,
            INF,
            INF,
            0
        }
    };
    // Print the solution
    floydWarshall(graph);
    getch();
    return 0;
}
```

Output

The following matrix shows shortest distances between every pair of vertices

0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0

Experiment 10

Program

```
#include<stdio.h>
#include<conio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
void bfs(int v) {
	for (i = 1; i <= n; i++)
		if (a[v][i] && !visited[i])
			q[++r] = i;
	if (f <= r) {
		visited[q[f]] = 1;
		bfs(q[f++]);
	}
}
void main() {
	clrscr();
	int v;
	printf("Enter the number of vertices: ");
	scanf("%d", & n);
	for (i = 1; i <= n; i++) {
		q[i] = 0;
		visited[i] = 0; }
	printf("\nEnter graph data in matrix form:\n");
	for (i = 1; i <= n; i++) {
		for (j = 1; j <= n; j++) {
			scanf("%d", & a[i][j]);
		}}
	printf("Enter the starting vertex: ");
	scanf("%d", & v);
	bfs(v);
	printf("\nThe node which are reachable are:");
	for (i = 1; i <= n; i++) {
		if (visited[i])
			printf(" %d", i);
		else {
			printf("\nBFS is not possible. All nodes are not reachable!");
			break;}
	}
	getch();
}
```

Output

Enter the number of vertices 3
Enter graph data in matrix form:
2
4
5
2
3
4
1
7
8
Enter the starting vertex: 2
The node which are reachable are: 1 2 3

Experiment 11

Program

```
#include<stdio.h>

#include<conio.h>

int a[20][20], reach[20], n;
void dfs(int v) {
	int I;
	reach[v] = 1;
	for (i = 1; i <= n; i++)
		if (a[v][i] && !reach[i]) {
			printf("\n % d -> % d", v, i);
			dfs(i);
		}
}
void main() {
	clrscr();
	int I, j, count = 0;
	printf("\nEnter number of vertices: ");
	scanf(" % d", & n);
	for (i = 1; i <= n; i++) {
		reach[i] = 0;
		for (j = 1; j <= n; j++)
			a[i][j] = 0;
	}
	printf("Enter the adjacency matrix: \n");
	for (i = 1; i <= n; i++)
		for (j = 1; j <= n; j++)
			scanf(" % d", & a[i][j]);
	dfs(1);
	printf("\n");
	for (i = 1; i <= n; i++) {
		if (reach[i])
			count++;
	}
	if (count == n)
		printf("Graph is connected");
	else
		printf("Graph is not connected");
	getch();
}
```

Output

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

Experiment 12

```
Algorithm N Queens ( k, n) //Prints all Solution to the n-queens problem
{
    for i: = 1 to n do {
        if Place(k, i) then {
            x[k]: = i;
            if (k = n) then write(x[1: n]
                else NQueens(k + 1, n);
            }
        }
    }
}
```

```
Algorithm Place (k, i)
{
for j := 1 to k-1 do
if (( x[ j ] = // in the same column
or (Abs( x [ j ] - i) =Abs ( j – k ))) // or in the same diagonal
then return false;
return true; }
```

Program Code

```
#include<conio.h>
#include<math.h>
#include<time.h>
#include<stdlib.h>

/* For printing the Grid */
void print_grid(int n, int x[]) {
	char arr[20][20];
	int i, j;
	clrscr();
	for (i = 1; i <= n; i++) {
		for (j = 1; j <= n; j++) {
			arr[i][j] = '-';
		}
	}
	for (i = 1; i <= n; i++) {
		arr[i][x[i]] = 'Q';
	}
	for (i = 1; i <= n; i++) {
		for (j = 1; j <= n; j++) {
			printf("\t%c", arr[i][j]);
		}
		printf("\n");
	}
}
/* For checking Queens placement is safe or not */
int safetoplace(int x[], int k) {
	int i;
	for (i = 1; i < k; i++) {
		if (x[i] == x[k] || i - x[i] == k - x[k] || i + x[i] == k + x[k]) {
			return 0; /* False*/
		}
	}
	return 1; /*true*/
}
/* For printing the Queens and Placing them in Grid */
void nqueens(int n) {
	int x[20];
	int count = 0;
	int k = 1;
	x[k] = 0;
	while (k != 0) {
		x[k] = x[k] + 1;
		while ((x[k] <= n) && (!safetoplace(x, k))) {
			x[k] = x[k] + 1;
		}
		if (x[k] <= n) {
			if (k == n) {
				count++;
				printf("\n\tPlacement %d is : \n\n\n", count);
				print_grid(n, x);
```

```
                    getch();
                } else {
                    k++;
                    x[k] = 0;
                }
            } else {
                k--;
            }
        }
        return;
}
int main() {
        int n;
        clrscr();
        printf("---------------------------------------------------\n");
        printf("---------------------------------------------------\n\n");
        printf("\t C PROGRAM OF N-QUEEN PROBLEM\n\n");
        printf("\nEnter the no. of Queens : ");
        scanf("%d", & n);
        printf("\n\n\tUSING %d QUEEN'S STRATEGY \n\n", n);
        nqueens(n);
        system("pause");
        getch();
}
```

Output