1) kruskal algorithm

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min, mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
        printf("\n\tImplementation of Kruskal's algorithm\n");
        printf("\nEnter the no. of vertices:");
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
        }
        printf("The edges of Minimum Cost Spanning Tree are\n");
        while(ne < n)
    {
                for(i=1,min=999;i<=n;i++)
                {
                        for(j=1;j <= n;j++)
                        {
                                if(cost[i][j] < min)
```

```
                                        {
                                                min=cost[i][j];
                                                a=u=i;
                                                b=v=j;
                                        }
                                }
                        }
                        u=find(u);
                        v=find(v);
                        if(uni(u,v))
                        {
                                printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                                mincost +=min;
                        }
                        cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
}
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:5

Enter the cost adjacency matrix:
999 5 999 6 999
5 999 999 3 999
999 1 999 4 6
6 3 4 999 2
999 999 6 2 999

The edges of Minimum Cost Spanning Tree are
1 edge (3,2) =1
2 edge (4,5) =2
3 edge (2,4) =3
4 edge (1,2) =5

    Minimum cost = 11
```

2)prims algorithm

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

void prims(int cost[][10], int n, int p[])
```

```
{
   int i, j, min, u, s[10], d[10], v;

   for (i = 0; i < n; i++)
   {
      d[i] = 999;
      s[i] = 0;
   }
   d[0] = 0;

   for (i = 1; i < n ; i++)
   {
      min = 999;
      for (j = 0; j < n; j++)
      {
         if (s[j] == 0 && d[j] < min)
         {
            min = d[j];
            u = j;
         }
      }

      s[u] = 1;

      for (v = 0; v < n; v++)
      {
         if (s[v] == 0 && cost[u][v] < d[v])
         {
            d[v] = cost[u][v];
            p[v] = u;
         }
```

```
        }
    }
}

void print_min_spanning_tree(int cost[][10], int n, int p[])
{
    int i, sum = 0;

    for (i = 1; i < n; i++)
    {
        sum += cost[i][p[i]];
    }

    if (sum >= 999)
        printf("No spanning tree\n");
    else
    {
        printf("Edge   Weight\n");
        for (i = 1; i < n ; i++)
            printf("%d->%d = %d\n", i, p[i], cost[i][p[i]]);

        printf("------------\nCost = %d\n", sum);
    }
}

void main()
{
    int n,i,j,cost[10][10],p[10];

    printf("Enter the number of vertices : ");
    scanf("%d", &n);
```

```
    printf("Enter cost adjacency matrix\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &cost[i][j]);

    prims(cost, n, p);

    print_min_spanning_tree(cost, n, p);
}
```

output

```
Enter the number of vertices : 6
Enter cost adjacency matrix

        999  25   15  999  999 999
         25  999 999   30  40  35
         15  999 999  999  20  999
        999  30  999  999  10   50
        999  40   20   10  999  45
        999  35  999   50  45  999

Edge   Weight
1->0 = 25
2->0 = 15
3->4 = 10
4->2 = 20
5->1 = 35
------------
Cost = 105
```

3a) floyds

```
#include<stdio.h>

void readf();
void amin();

int cost[20][20], a[20][20];
int i,j,k,n;

void readf()
{
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("\nEnter the weighted matrix - 999 for infinity\n");
   for(i=0; i<n; i++)
   {
      for(j=0; j<n; j++)
      {
         scanf("%d", &cost[i][j]);
         a[i][j] = cost[i][j];
      }
   }
}

void amin()
{
   for(k=0; k<n; k++)
```

```
    {
        for(i=0; i<n; i++)
        {
            for(j=0; j<n; j++)
            {
                if(a[i][k] + a[k][j] < a[i][j])
                {
                    a[i][j] = a[i][k] + a[k][j];
                }
            }
        }
    }
    printf("\nThe shortest path matrix is:\n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    readf();
    amin();
    return 0;
}
```

output

Enter the number of vertices: 4

Enter the weighted matrix - 999 for infinity

0 999 3 999

2 0 999 999

999 7 0 1

6 999 999 0

The shortest path matrix is:

0 10 3 4

2 0 5 6

7 7 0 1

6 16 9 0

3b) warshals

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

void warshall(int p[10][10], int n)
{
   int i,j,k;
   for(k=0; k<n; k++)
   {
      for(i=0; i<n; i++)
      {
         for(j=0; j<n; j++)
         {
            p[i][j] = p[i][j] || (p[i][k]&&p[k][j]);
```

```
        }
      }
    }
}

void main()
{
    int p[10][10]={0}, n, i, j,e,u,v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &e);
    printf("Enter the edges: (u,v) \n");
    for(i=0; i<e; i++)
    {
        scanf("%d %d", &u,&v);
        p[u-1][v-1] = 1; // subtract 1 because array indices start from 0
    }
    printf("\n Matrix of input data: \n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            printf("%d\t", p[i][j]);
        }
        printf("\n");
    }
    warshall(p, n);
    printf("The transitive closure matrix is:\n");
    for(i=0; i<n; i++)
    {
```

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

output

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter the edges: (u,v)
1 2
2 4
4 1
4 3

 Matrix of input data:
0     1     0     0
0     0     0     1
0     0     0     0
1     0     1     0
The transitive closure matrix is:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

4)Dijkstra's algorithm

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

void dijkstra(int cost[][10], int n, int source, int d[], int p[])
{
   int i, j, min, u, s[10], v;

   for (i = 0; i < n; i++)
   {
      d[i] = 999;
      p[i] = source;
      s[i] = 0;
   }

   d[source] = 0;

   for (i = 1; i < n ; i++)
   {
      min = 999;
      for (j = 0; j < n; j++)
      {
         if (s[j] == 0 && d[j] < min)
         {
            min = d[j];
            u = j;
         }
      }

      s[u] = 1;
```

```
        for (v = 0; v < n; v++)
        {
            if (s[v] == 0 && d[u] + cost[u][v] < d[v])
            {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
            }
        }
    }
}

void main()
{
    int n, i, j, d[10], p[10], source;
    int destination;
    int cost[10][10];

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter cost adjacency matrix\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &cost[i][j]);


    printf("Enter the source and destination: ");
    scanf("%d %d", &source, &destination);

    dijkstra(cost, n, source, d, p);
```

```
    if (d[destination] == 999 )
    {
        printf("Path does not exist\n");
        exit(0);
    }

    printf("Path : ");

    i = destination;
    while (i != source)
    {
        printf("%d<--", i);
        i = p[i];
    }
    printf("%d = %d\n", i, d[destination]);
}
```

output

```
Enter the number of vertices: 6

Enter cost adjacency matrix
        0    15   10  999 45  999
        999  0    15  999 20  999
        20   999  0   20  999 999
        999  10   999 0   35  999
        999  999  999 30  0   999
        999  999  999 4   999 0

Enter the source and destination: 5   0
```

Path : 0<--2<--1<--3<--5 = 49

5)Topological ordering

```
#include <stdio.h>
const int MAX = 10;

void fntopological (int a[MAX][MAX],int n);

int main(void)
{
   int a[MAX][MAX],n;
   int i,j;

   printf("Topological Sorting Algorithm");
   printf ("\nEnter the no of vertices:");
   scanf("%d",&n);

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

   for (i=0; i<n;i++)
     for (j=0; j<n;j++)
         scanf ("%d", &a[i][j]);
   fntopological (a, n);
   printf ("\n");
   return 0;

}
void fntopological (int a[MAX][MAX],int n)
```

```
{
    int in[MAX], out[MAX], stack[MAX],top = -1;
    int i, j, k = 0;
    for (i=0;i<n;i++)
    {
        in[i] = 0;
        for(j=0;j<n;j++)
            if(a[j][i]==1)
                in[i]++;
    }
    while (1)
    {
        for (i = 0 ; i < n ;i++)
        {
            if (in[i]==0)
            {
                stack [++top]=i;
                in[i]=-1;
            }
        }
        if( top ==-1)
            break;
        out[k] = stack[top--];

        for( i = 0 ;i<n;i++)
        {
            if (a[out[k]][i]==1)
                in[i]--;
        }
        k++;
    }
```

```
   printf("Topological sorting as follows:- \n");
   for(i=0; i<k;i++)
      printf("%d", out[i]+1);
}
```

output

```
Topological Sorting Algorithm
Enter the no of vertices:5

Enter the adjacency matrix:
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0

Topological sorting as follows:-
21345
```

6) 0/1 knapsack

```
#include <stdio.h>

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

```
}
int knapsack (int w[], int p[], int n, int m)
{
   if (m==0)
      return 0;
   if (n==0)
      return 0;
   if (w[n-1]>m)
      return knapsack (w,p, n-1,m);
   return max(knapsack (w,p,n-1,m), p[n-1]+knapsack(w, p, n-1,m-w[n-1]));
}
void main()
{
   int i, n, m, w[10],p[10];

   printf("Enter no of items \n");
   scanf("%d",&n);

   printf("Enter the weight and price of all items: \n");
   for (i=0; i<n;i++)
   {
      scanf("%d %d", &w [i], &p[i]);
   }
   printf("Enter the capacity of knapscak:\n");
   scanf("%d", &m);

   printf ("Enter the maximum value item that can be put into knapsack is = %d\n", knapsack
(w,p,n,m));

}
```

output

Enter no of items

4

Enter the weight and price of all items:

10 15

20 25

30 35

40 45

Enter the capacity of knapscak:

45

Enter the maximum value item that can be put into knapsack is = 50

7) discrete knapsack

```
#include <stdio.h>
#include <stdlib.h>
struct Item
{
   int value;
   int weight;
};

int compare(const void *a, const void *b)
{
   struct Item *item1 = (struct Item *)a;
   struct Item *item2 = (struct Item *)b;
   double ratio1 = (double)item1 -> value/item1 -> weight;
   double ratio2 = (double)item2 -> value/item2 -> weight;
   return (ratio2 > ratio1)-(ratio2 < ratio1);
```

```
}
void discreteKnapsack(struct Item items[], int n, int capacity)
{
   qsort(items, n, sizeof(struct Item), compare);
   int currentWeight = 0;
   double totalValue = 0.0;
   for (int i=0; i<n; ++i)
   {
      if (currentWeight + items[i].weight <= capacity)
      {
         currentWeight += items[i].weight;
         totalValue += items[i].value;

      }

      else
      {
         int remainingCapacity= capacity - currentWeight;
         totalValue += items[i].value* ((double)remainingCapacity/items[i].weight);
         break;
      }

   }
   printf("Total value obtained in discrete Knapsack: %.2f\n", totalValue);
}
int main()
{
   struct Item items[] = {{60, 10}, {100,20}, {120, 30}};
   int n = sizeof(items) / sizeof(items[0]);
   int capacity = 50;
   discreteKnapsack(items, n, capacity);
```

```
    return 0;
}
```

output

Total value obtained in discrete Knapsack: 240.00

8) subset

```
#include<stdio.h>
int s[10],d,n,set[10],count=0;
void display(int);
int flag=0;
int subset(int, int);

void main()
{
    int i;
    printf("Enter the number of elements in set\n");
    scanf("%d", &n);
    printf("Enter the set values \n");
    for(i=0;i<n;++i)
        scanf("%d" ,&s[i]);
    printf("Enter the sum\n");
    scanf("%d", &d);
    printf("The program output is \n");
    subset(0,0);

    if(flag==0)
    printf("There is no solution");
```

```
}
int subset (int sum,int i)
{
   if(sum==d)
   {
      flag =1;
      display(count);
      return (0);
   }
   if(sum>d || i>=n)
      return 0;
   else
   {
      set[count]=s[i];
      count++;
      subset(sum+s[i],i+ 1);
      count--;
      subset(sum,i+ 1);
   }
}
void display(int count)
{
   int i;
   printf("{");
   for(i=0;i<count;i++)
   printf("%d ", set[i]);
   printf("}");
}
```

output

Enter the number of elements in set

6

Enter the set values

6 5 4 3 2 1

Enter the sum

5

The program output is

{5 }{4 1 }{3 2 }

9) selection sort

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

void swap(long int *a, long int *b)
{
   long int temp = *a;
   *a = *b;
   *b = temp;
}

void selectionSort(long int arr[], long int n)
{
   long int i, j, midx;
   for (i = 0; i < n - 1; i++)
   {
      midx = i;
      for (j = i + 1; j < n; j++)
      {
```

```
            if (arr[j] < arr[midx]) {
                midx = j;
            }
        }
        swap(&arr[i], &arr[midx]);
    }
}
void main()
{
    long int n = 1000;
    int it = 0;
    double tim[10];
    printf("input size, selection sorting time\n");

    while (it++ < 5)
    {
        long int a[n];
        for (int i = 0; i < n; i++)
        {
            long int no = rand() % n + 1;
            a[i] = no;
        }

        clock_t start, end;
        start = clock();
        selectionSort(a, n);
        end = clock();

        tim[it] = (double)(end - start)/100 ;
        printf("%ld = %ld  ms\n", n, (long int) tim[it]);
        n += 100;
```

```
    }
}
```

output

input size, selection sorting time

1000 = 14 ms

1100 = 17 ms

1200 = 20 ms

1300 = 23 ms

1400 = 25 ms

10)quick sort

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

static int max = 5000;

static int partition(long int arr[], int low, int high)
{
    int pivot = arr[low];
    int i = low;
    int j = high+1;
    while(i<=j)
    {
        do
        {
            i++;
```

```
        }
        while(pivot>=arr[i] && i<=high);
        do
        {
           j--;
        }
        while(pivot<arr[j]);
        if(i<j)
        {
           int temp = arr[i];
           arr[i] = arr[j];
           arr[j] = temp;
        }
    }
    int temp = arr[low];
    arr[low] = arr[j];
    arr[j] = temp;

    return j;
}

static void quicksort(long int arr[], int low, int high)
{
    int mid;
    if(low<high)
    {
        mid= partition(arr, low, high);
        quicksort(arr, low, mid-1);
        quicksort(arr, mid+1, high);
    }
}
```

```
void main()
{
   int n,i;
   long int  a[5000], no;
   double tim;
   clock_t start, end;
   printf("\nEnter the no of elements: ");
   scanf("%d", &n);
   for(i=0;i<n;i++)
   {
      no = rand()%n+1;
      a[i] = no;
   }

   start = clock(); // start the timer
   quicksort(a, 0, n - 1); // sort the array
   end = clock(); // stop the timer

   tim =(end - start) ;

   printf("\n%d: %lf nanoseconds\n",n, tim);
}
```

11)merge sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 5000
```

```
int array[max];

void merge (int low, int mid, int high)
{
   int temp [max];
   int i = low;
   int j = mid+1;
   int k = low;

   while ((i<=mid) && (j<=high))
   {
      if (array [i]<= array[j])
         temp[k++] = array [i++];
      else
         temp[k++] = array [j++];
   }

   while (i<=mid)
   {
      temp[k++] = array[i++];
   }
   while (j<=high)
   {
      temp [k++] = array [j++];
   }
   for (i= low; i<=high; i++)
   {
      array[i] = temp[i];
   }
}
void merge_sort (int low, int high)
```

```
{
   int mid;
   if (low <high)
   {
      mid = (low + high)/2;
      merge_sort (low, mid);
      merge_sort (mid+1, high);
      merge (low, mid, high);
   }
}

void main()
{
   int i, n, no;
   double tm,
   clock_t, start, end;

   printf("Enter the no of elements: ");
   scanf ("%d",&n);

for (i=0;i<n;i++)
{
   no = rand() %n+1;
   array [i]=no;
}
printf("unsorted array is : \n");
for (i=0; i<n;i++)
   printf ("%d", array[i]);
start=clock();
merge_sort (0,n-1);
printf("\nSorted list is : \n");
```

```
for (i=0;i<n;i++)
   printf ("%d", array[i]);
   printf("\n");

end = clock();
tm = (end-start);

printf("%d = %lf Nano seconds \n",n,tm);
printf("\n");
}
```

output

```
Enter the no of elements: 9
unsorted array is :
281868247
Sorted list is :
122467888
9 = 29.000000 Nano seconds
```

12) N queens

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

int board[10], count = 0;

void print(int n);
int place(int row, int col, int n);
```

```
void queen(int row, int n);

void main()
{
   int n;
   printf("-N Queens Problem using backtracking method\n");
   printf("Enter the number of queens: ");
   scanf("%d",&n);
   queen(1,n);
}

void print(int n)
{
   int i,j;
   printf("\nSolution %d:\n", ++count);
   for(i=1; i<=n; i++)
   {
      for(j=1; j<=n; j++)
      {
         if(board[i]==j)
            printf("Q ");
         else
            printf("x ");
      }
      printf("\n");
   }
}

int place(int row, int col, int n)
{
   int i;
```

```
   for(i=1; i<row; i++)
   {
      if(board[i] == col)
         return 0;
      else
         if(abs(board[i]-col) == abs(i-row))
            return 0;
   }
   return 1;
}

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

output

-N Queens Problem using backtracking method

Enter the number of queens: 4

Solution 1:

x Q x x

x x x Q

Q x x x

x x Q x

or

nqueens

```
#define N 4
#include <stdbool.h>
#include <stdio.h>
void printSolution (int board [N][N])
{
   for (int i=0;i<N;i++)
   {
      for (int j=0;j<N;j++)
      {
         if (board[i][j])
            printf("Q");
         else
            printf(".");
      }
      printf("\n");

   }
}
```

```
bool isSafe (int board [N][N],int row, int col)
{
   int i,j;
   for (i=0;i<col;i++)
      if (board [row][i])
         return false;
   for(i=row, j=col; i>=0&&j>=0;i--,j--)
      if (board [i][j])
         return false;
   for(i=row, j=col; j>=0&&i<N;i++,j--)
      if (board [i][j])
         return false;
   return true;
}
bool solveNQUtil (int board [N][N], int col)
{
   if(col>=N)
      return true;
   for(int i=0;i<N;i++)

   {
      if (isSafe (board, i, col))
      {
         board [i][col]=1;
         if (solveNQUtil (board, col+1))
            return true;
         board [i][col]=0;
      }

   }
   return false;
```

```
}
bool solveNQ()
{
   int board [N][N]={{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}};
   if(solveNQUtil(board,0)==false)
   {
      printf("no solution");
      return false;
   }
   printSolution(board);
   return true;
}
int main()
{
   solveNQ();
   return 0;
}
```

output

```
..Q.
Q...
...Q
.Q..
```