1. Krushkals Algorithm

```
#include <stdio.h>
#define INFINITY 999
int i, j, u, v, a, b,cost[20][20], min, mincost = 0, n, parent[20],ne = 1;
void main()
{
printf("Enter the number of vertices:\n");
scanf("%d", &n);
    printf("Enter 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;
        }
    }
    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;
                }
            }
        }
        while (parent[u])
            u = parent[u];
        while (parent[v])
            v = parent[v];
        if (u != v) {
   printf("%d Edge <%d,%d> = %d\n", ne++, a, b, min);
       mincost += min;
       parent[v] = u;
    }
     cost[a][b] = cost[b][a] = 999;
    }
    printf("Minimum Cost = %d\n", mincost);
}
```

2. Prims Algorithm

```
#include <stdio.h>
#define INFINITY 999
int prim(int cost[10][10],int source, int n){
int i, j, sum = 0,visited[10], cmp[10], vertex(10);
int min, u,v;
for(i=1;i<=n;i++){
vertex[i] = source;
visited[i] = 0;
cmp[i] = cost[source] [i];
}
visited (source] = 1;
for(i=1; i<n-1; i++){
min = INFINITY;
for(j=1;j<=n;j++)
if(! visited[j] &&cmp[j] <min){
{
min = cmp[j];
u=j;
}
visited [u] = 1;
Sum = sum + cmp[u];
printf("\n%d->%d sum =%d",
vertex[u], u, sum);
for(v=1; v<=n; ++)
if(! visited[v] && cost[u] [v] < cmp[v]){
cmp[v] = cost[u][v];
vertex[v] = u;
}
}
return sum;
}
void main(){
int a[10][10], n, i, j, m, source;
printf("\n Enter the number of vertices: ");
scanf("%d", &n);
printf("\n Enter the cost matrix: 0-self loop & 999-no edge \n");
for(i=1; i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d", &a[i][j]);
printf("\n Enter the source: "); scanf("%d", &source);
m=prim (a, source, n);
printf("\n\n cost = "%d", m);
}
```

3.(a) Floyds Algorithm

```
#include <stdio.h>
#define INFINITY 999
int min(int a, int b) {
return a<b?a:b;
} void floyds(int a[10][10],int n)
{
int i,j,k;
for (k = 1; k<= n; k++)
for(i= 1; i<=n;i++)
for(j=1;j<=n;j++)
a[i][j] = min(a[i][j], a[i][k] + a[k][j]);
} void main() {
int n,i,j,a[10][10];
printf("Enter the number of vertices \n");
scanf("%d", &n);
printf("Enter the cost adjacency matrix: \n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d", &a[i][j]);
floyds (a, n);
printf("The shortest distance matrix is: \n");
for(i=1; i <= n i++) {
for(j=1;j<=n;j++) {
printf("%d\t", a[i][j]);
}
printf("\n");
}
}
```

3.(b) Warshalls Algorithm

```
#include <stdio.h>
void warshalls(int a[10][10],int n){
int i,j,k;
for(k =1;k<= n ;k++)
for(i =1;i<=n;i++)
for(j=1;j<=nj++)
a[i][j] =a[i][j]||(a[i][k] && a[k][j]);
} void main() {
int a[10][10], n,i,j;
printf ("Enter the number of vertices: \n");
scanf("%d", &n);
printf("Enter the adjacency matrix:\n");
for(i=1;i<= n;i++)
for(j=1;j<=n;j++)
scanf("%d", & a[i][j]);
warshalls(a,n);
printf("Transitive closure: \n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%d\t", a[i][j]);
printf("\n");
 }
}
```

4. Dijkstra Algorithm

```
#include <stdio.h>
#define INFINITY 999
void dijkstra(int cost[10][10], int n, int source, int distance[10]);
void main()
{
    int n, source, i, j, a[10][10], distance[10];
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the Cost efficiency matrix enter 999 for no edge:\n");
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);
   printf("Enter the source vertex:");
    scanf("%d", &source);
    dijkstra(a, n, source, distance);
    for(i = 1; i <= n; i++)
    printf("The shortest distance from %d->%d = %d\n", source, i, distance[i]);
    }
void dijkstra(int cost[10][10], int n, int source, int distance[10])
{
    int visited[10], min, u, i, j;
    for(i = 1; i <= n; i++)
 {
        distance[i] = cost[source][i];
        visited[i] = 0;
    }
    visited[source] = 1;
    for(i = 1; i <= n - 1; i++)
{
     min = INFINITY;
        for(j = 1; j <= n; j++)
if(visited[j]==0 && distance[j] < min)
 {
      min = distance[j];
       u = j;
   }
        visited[u] = 1;
        for(j = 1; j <= n; j++)
       if(visited[j]==0 && (distance[u] + cost[u][j]) < distance[j]){
   distance[j] = distance[u] +cost[u][j];
            }
        }
    }
```

5. Topological Order

```
#include<stdio.h>
int main() {
   int n, i, j, k, a[10][10], indeg[10], flag[10], count = 0;

   printf("Enter the number 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]);
   for(i = 0; i < n; i++) {
      indeg[i] = 0;
      flag[i] = 0;
   }

   for(i = 0; i < n; i++)
      for(j = 0; j < n; j++)
         indeg[i] += a[j][i];
   printf("Topological order: ");
   while(count < n) {
      for(k = 0; k < n; k++) {
         if(indeg[k] == 0 && flag[k] == 0) {
            printf("%d", k + 1);
            flag[k] = 1;
            for(i = 0; i < n; i++)
               if(a[k][i] == 1)
                  indeg[i]--;
               }
            }
      count++;
   }
   printf("\n");
}
```

6. 0/1 Knapsack problem

```
#include<stdio.h>
int wt[50],val[50];
int max(int a, int b)
{
   return (a > b)? a: b;
}
int knapsack(int W, int n) {
   int i, w;
   int knap[n+1][W+1];
   for (i = 0; i <= n; i++){
   for (w=0; w <= W; w++){
   if (i==0 || w==0)
   knap[i][w] = 0;
else if (wt[i-1] <= w)
knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
else
knap[i][w] = knap[i-1][w];
}
}
for(i=0;i<=n;i++){
for(j=0;j<=W;j++){
printf("%d\t",knap[i][j]);
}
printf("\n");
}
return knap[n][W];
}
int main()
{ int i,n,W;
printf("\n enter the number of objects");
scanf("%d",&n);
printf("enter the knapscak capacity");
scanf("%d",&W);
printf("\n enter the profit:\n");
for(i=0;i<n;i++)
scanf("%d",&val[i]);
printf("\n enter the weight:\n");
for(i=0;i<n;i++)
scanf("%d",&wt[i]);
printf("The solution is: %d", knapsack(W, n));
return 0;
}
```

7. Knapsack using greedy method

```
#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;
}
```

8. Subset

```
#include <stdio.h>
int flag = 0;
void printsubset(int subset[], int size) {
   printf("Subset: { ");
   for(int i = 0; i < size; i++) {
      flag = 1;
      printf("%d ", subset[i]);
   }
   printf("}\n");
}
void Sumofsubsetuntil(int weights[], int targetsum, int n, int subset[], int subsetsize, int sum, int
index) {
   if (sum == targetsum) {
      flag = 1;
      printsubset(subset, subsetsize);
      return;
   }
   for (int i = index; i < n; i++) {
      if (sum + weights[i] <= targetsum) {
         subset[subsetsize] = weights[i];
         Sumofsubsetuntil(weights, targetsum, n, subset, subsetsize + 1, sum + weights[i], i + 1);
      }
   }
}
void sumofsubset(int weights[], int targetsum, int n) {
   int subset[n];
   Sumofsubsetuntil(weights, targetsum, n, subset, 0, 0, 0);
}
int main() {
   int n, targetsum;
   printf("Enter the number of Elements: \n");
   scanf("%d", &n);
   int weights[n];
   printf("Enter the elements: \n");
   for (int i = 0; i < n; i++)
      scanf("%d", &weights[i]);
   printf("Enter the target sum:\n");
   scanf("%d", &targetsum);
   sumofsubset(weights, targetsum, n);
   if (flag == 0)
      printf("No solution\n");
   return 0;
}
```

9. Selection Sort

```
# include <stdio.h>
# include <time.h>
void main()
{
int n,a[10000],i,j,temp,min;
clock_t st, et;
double ts;
printf("\n Enter the number of array elements: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(i=0; i<n; i++)
{
a[i]=rand()%100+1;
printf("%d\t",a[i]);
}
st=clock();
for(i=0; i<=n-2; i++)
 {
 min = i;
 for(j=i+1; j<=n-1; j++)
 {
 if(a[j]<a[min])
 {
min =j;
 }
 }
 temp =a[i];
 a[i]=a[min];
 a[min]=temp;
 }
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\nSorted Numbers are: \n ");
for(i=0; i<n; i++)
{
printf("%d\t", a[i]);
}
printf("\nThe time taken is %lf",ts);
}
```

10 Quick Sort

```
# include <stdio.h>
# include <time.h>
#include<stdlib.h>
int a[10000],i,j,lb,ub;
void quicksort(int a[], int lb, int ub);
void main()
{
int n;
clock_t st, et;
double ts;
printf("\n Enter the number of array elements: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(i=0; i<n; i++)
{
a[i]=rand()%100+1;
printf("%d\t",a[i]);
}
st=clock();
quicksort(a,0,n-1);
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\nSorted Numbers are: \n ");
for(i=0; i<n; i++)
{
printf("%d\t", a[i]);
}
printf("\nThe time taken is %lf",ts);
}
void quicksort(int a[], int lb, int ub)
{ int mid;
 if(lb<ub)
 {
 mid = partition(a,lb,ub);
 quicksort(a,lb,mid-1);
 quicksort(a,mid+1,ub);
 }
}
int partition(int a[],int lb, int ub)
{
int i, j, pivot,temp;
pivot = a[lb];
i = lb;
j = ub;
while(i<j)
{
while(a[i]<=pivot)
{
i++;
}
while(a[j]>pivot)
{
j--;
}
if(i<j)
{
temp = a[i];
a[i]= a[j];
a[j] = temp;
 }}
temp=a[lb];
a[lb] = a[j];
a[j] = temp;
return j;
}
```

11 Merge Sort

```
# include <stdio.h>
# include <time.h>
#include<stdlib.h>
int a[15000],b[15000],i,j,k,lb,ub;
void mergesort(int a[], int lb, int ub);
void merge(int a[], int lb, int mid, int ub);
void main()
{
int n;
clock_t st, et;
double ts;
printf("\n Enter the number of array elements: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(i=0; i<n; i++)
{
a[i]=rand()%10000;
printf("%d\t",a[i]);
}
st=clock();
mergesort(a,0,n-1);
et=clock();
ts=((double)(et-st))/CLOCKS_PER_SEC;
printf("\nSorted Numbers are: \n ");
for(i=0; i<n; i++)
{
printf("%d\t", a[i]);
}
printf("\nThe time taken is %lf",ts);
}
void mergesort(int a[], int lb, int ub)
{ int mid;
 if(lb<ub)
 {
 mid = (lb+ub)/2;
 mergesort(a,lb,mid);
 mergesort(a,mid+1,ub);
 merge(a,lb,mid,ub);
 }
}
void merge(int a[], int lb, int mid, int ub)
{ i=lb;
 j=mid+1;
k=lb;
while(i<=mid && j<=ub){
if(a[i]<= a[j]){
b[k] = a[i];
i++;
}
else{
b[k] = a[j];
j++;
}
k++;
}
if(i>mid)
{
while(j<=ub)
{
b[k]=a[j];
j++;
k++;
}
}
else
{
while(i<=mid)
{
b[k]=a[i];
i++;
k++;
}
}
for(k=lb; k<=ub; k++){
a[k] = b[k];
}
}
```

12 N-queens problem

```
#include<stdio.h>
#include<math.h>
int board[20],count;
void queen(int row,int n);
int main(){
int n,i,j;
printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
if(n==2||n==3)
printf("no solution\n");
else
queen(1,n);
}
void print(int n){
int i,j;
printf("Solution %d:\n",++count);
for(i=1;i<=n;++i)
 printf("\t%d",i);
printf("\n");
for(i=1;i<=n;++i){
for(j=1;j<=n;j++){
 printf("\n\n%d",i);
 for(j=1;j<=n;++j){
 if(board[i]==j)
 printf("\tQ");
 else
 printf("\t-");
 } printf("\n");
}
}
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;
}
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);
}
}
}
```