1) Reachable nodes using BFS

```
#include<stdio.h>

#include<stdlib.h>

int visited[100], queue[100], front = -1, rear = -1, n, i, j;

int adj[100][100];

void bfs(int v){

    while(front<=rear){

        v=queue[front++];

        for(int i=0;i<n;i++){

            if(adj[v][i] && !visited[i]){

                queue[rear++]=i;

                visited[i]=1;

            }

        }

    }

}

int main(){

    int v;

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

    scanf("%d",&n);

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

    for(i=0;i<n;i++){

        for(j=0;j<n;j++){

            scanf("%d",&adj[i][j]);

        }

    }

    printf("\nEnter the starting vertex: ");

    scanf("%d",&v);

    for(i=0;i<n;i++){

        queue[i]=0;

        visited[i]=0;

    }

    bfs(v);

    for(i=0;i<n;i++){

        if(visited[i]){

            printf("->%d",i);
        }
        else{
            printf("Not possible");
        }
    }
}
```

2)Topological sort

```
#include <stdio.h>
#include <stdlib.h>
int s[100],res[100];
int j;
void adjacency(int a[100][100],int n){
    printf("Enter the elements: \n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    }
}
void dfs(int u,int n,int a[100][100]){
    s[u]=1;
    int v;
    for(v=0;v<n;v++){
        if(a[u][v]&& !s[v]){
            dfs(v,n,a);
        }
    }
    res[j--]=u;
}
void topological(int n, int a[100][100]){
    int i,u;
    for(i=0;i<n;i++){
        s[i]=0;
    }
    j=n-1;
    for(u=0;u<n;u++){
        if(!s[u]){
            dfs(u,n,a);
        }
    }
}
```

```
int main()
{
    int a[100][100],n,i,j;
    printf("Enter the no. of vertices: ");
    scanf("%d",&n);
    adjacency(a,n);
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    printf("Topological Order: \n");
    topological(n,a);
    for(i=0;i<n;i++){
        printf("->%d",res[i]);
    }
    return 0;
}
```

3)Johnson Trotter

```
#include <stdio.h>
#include <stdlib.h>
#include<stdbool.h>
void printt(int perm[],int n){
    for (int i=0;i<n;i++){
        printf("%d",perm[i]);
    }
    printf("\n");
}
void johnsontrotter(int n)
{
    int perm[n],dir[n];
    for(int i=0;i<n;i++){
        perm[i]=i+1;
        dir[i]=-1;
    }
    printt(perm,n);
    while(true){
        int mobile=-1, mobileindex=-1;
        for(int i=0;i<n;i++){
            if(dir[i]==-1 && i>0 && perm[i]>perm[i-1]){
                if(perm[i]>mobile){
                    mobile=perm[i];
                    mobileindex=i;
                }
            }
            if(dir[i]==1 && i<n-1 && perm[i]>perm[i+1]){
                if(perm[i]>mobile){
                    mobile=perm[i];
                    mobileindex=i;
                }
            }
        }
        if(mobileindex==-1){break;}
        int swapindex=mobileindex+dir[mobileindex];
        int temp=perm[mobileindex];
        perm[mobileindex]=perm[swapindex];
        perm[swapindex]=temp;

        temp=dir[mobileindex];
        dir[mobileindex]=dir[swapindex];
        dir[swapindex]=temp;

        for(int i=0;i<n;i++){
            if(perm[i]>mobile){
                dir[i]=-dir[i];
            }
        }
        printt(perm,n);
    }
}
int main()
{
    int n=3;
    johnsontrotter(n);
    return 0;
}
```

4)Merge Sort

```
#include <stdio.h>

#include <stdlib.h>

#include<time.h>

int count=0;

void merge(int arr[], int l,int mid,int r){

    int an=mid-l+1;

    int bn=r-mid;

    int a[an];

    int b[bn];

    for(int i=0;i<an;i++){

        a[i]=arr[l+i];

    }

    for(int j=0;j<bn;j++){

        b[j]=arr[mid+1+j];

    }

    int i=0,j=0,k=l;

    while(i<an && j<bn){

            count++;

        if(a[i]<b[j]){

            arr[k++]=a[i++];

        }

        else{

            arr[k++]=b[j++];

        }

    }

    while(i<an){

        arr[k++]=a[i++];

    }

    while(j<bn){

        arr[k++]=b[j++];


    }

}

void mergesort(int arr[],int l,int r){

    if(l>=r){
        return;
    }
    int mid=(l+r)/2;
    mergesort(arr,l,mid);
    mergesort(arr,mid+1,r);
    merge(arr,l,mid,r);
}
void printarray(int a[],int n){
    for(int i=0;i<n;i++){
        printf("%d\t",a[i]);
    }
}

int main() {
    int n;
    printf("Enter the no. of elements: ");
    scanf("%d", &n);
    int aa[n], ad[n], ar[n];
    srand(time(0));
    for (int i = 0; i < n; i++) {
        aa[i] = i + 1;
        ad[i] = n - i;
        ar[i] = (rand() % n) + 1;
    }
    printf("Before Sorting\n");
    printf("Ascending:\n");
    printarray(aa, n);
    printf("\nDescending:\n");
    printarray(ad, n);
    printf("\nRandom:\n");
    printarray(ar, n);

    clock_t start, end;
    double time_taken;

    // Ascending array sort
```

```
start = clock();
mergesort(aa, 0, n - 1);
end = clock();
time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("\nTime taken for Ascending: %f seconds", time_taken);
printf("\nCount: %d\n", count);

// Reset count for next sort
count = 0;

// Descending array sort
start = clock();
mergesort(ad, 0, n - 1);
end = clock();
time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("\nTime taken for Descending: %f seconds", time_taken);
printf("\nCount: %d\n", count);

// Reset count for next sort
count = 0;

// Random array sort
start = clock();
mergesort(ar, 0, n - 1);
end = clock();
time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("\nTime taken for Random: %f seconds", time_taken);
printf("\nCount: %d\n", count);

printf("\nAfter Sorting:\n");
printf("Ascending:\n");
printarray(aa, n);
printf("\nDescending:\n");
printarray(ad, n);
printf("\nRandom:\n");
printarray(ar, n);
```

```
    return 0;
}
```

5)Quick Sort

```
#include <stdio.h>
#include <stdlib.h>
int count=0;
void swap(int *a, int *b){
    int temp=*a;
    *a=*b;
    *b=temp;
}

int partition(int arr[],int low,int high){
    int pivot=arr[low];
    int i=low+1;
    int j=high;
    while(1){
        while(arr[i]<=pivot && i<=high){
            count++;
            i++;
        }
        while(arr[j]>pivot && j>=low){
            count++;
            j--;
        }
        if(i<j){
            swap(&arr[i],&arr[j]);
        }
        else{
            arr[low]=arr[j];
            arr[j]=pivot;
            return j;
        }
    }
```

```
}
void quicksort(int a[],int l,int r){
    if (l<=r){
        int p=partition(a,l,r);
        quicksort(a,l,p-1);
        quicksort(a,p+1,r);
    }
}
void printarray(int a[],int n){
    for(int i=0;i<n;i++){
        printf("%d\t",a[i]);
    }
}
int main(){
    int n;
    printf("Enter the no. of elements: ");
    scanf("%d",&n);
    int aa[n],ad[n],ar[n];
    srand(time(0));
    for(int i=0;i<n;i++){
        aa[i]=i+1;
        ad[i]=n-i;
        ar[i]=(rand()%n)+1;
    }
    printf("Before Sorting\n");
    printf("Ascending:\n");
    printarray(aa,n);
    printf("\nDescending:\n");
    printarray(ad,n);
    printf("\nRandom:\n");
    printarray(ar,n);
    printf("\n");

    quicksort(aa,0,n-1);
    printf("\nCount= %d",count);
    count=0;
    quicksort(ad,0,n-1);
    printf("\nCount= %d",count);

    count=0;
    quicksort(ar,0,n-1);
    printf("\nCount= %d",count);

    printf("\nAfter Sorting:\n");
    printf("Ascending:\n");
    printarray(aa,n);
    printf("\nDescending:\n");
    printarray(ad,n);
    printf("\nRandom:\n");
    printarray(ar,n);

}
```

6)Heap Sort

```
#include <stdio.h>

#include <stdlib.h>

int count=0;

void swap(int *a, int *b){

   int temp=*a;

   *a=*b;

   *b=temp;

}

void heapify(int arr[],int n,int i){

   int largest=i;

   int left=2*i+1;

   int right=2*i+2;

   count++;

   if(left<n && arr[left]>arr[largest]){

      largest=left;

   }

   if(right<n && arr[right]>arr[largest]){

      largest=right;

   }

   if(largest!=i){

      swap(&arr[i],&arr[largest]);

      heapify(arr,n,largest);

   }

}

void heapsort(int arr[],int n){

   for(int i=n/2-1;i>=0;i--){

      heapify(arr,n,i);

   }

   for(int i=n-1;i>0;i--){

      swap(&arr[0],&arr[i]);

      heapify(arr,i,0);

   }

}

void printarray(int arr[], int n){

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

int main()
{
    int n,i;
    printf("Enter the no. of elements: ");
    scanf("%d",&n);
    int arrA[n],arrD[n],arrR[n];
    srand(time(0));
    for(i=0;i<n;i++){
        arrA[i]=i+1;
        arrD[i]=n-i;
        arrR[i]=(rand()%n)+1;
    }
    printf("Before Sorting\n");
    printf("Ascending:\n");
    printarray(arrA,n);
    printf("\nDescending:\n");
    printarray(arrD,n);
    printf("\nRandom:\n");
    printarray(arrR,n);
    printf("\n");

    heapsort(arrA,n);
    printf("\nCount= %d",count);

    count=0;
    heapsort(arrD,n);
    printf("\nCount= %d",count);

    count=0;
    heapsort(arrR,n);
    printf("\nCount= %d",count);
    printf("\nAfter Sorting:\n");
    printf("Ascending:\n");
    printarray(arrA,n);
    printf("\nDescending:\n");
    printarray(arrD,n);
    printf("\nRandom:\n");
    printarray(arrR,n);
    return 0;
}
```

7)Knapsack

```
#include <stdio.h>

#include <stdlib.h>

int max(int a,int b){

    if(a>b){return a;}

    else{return b;}

}

int knapsack(int n, int C,int w[],int p[]){

    int v[n+1][C+1];

    int i,j;

    for(i=0;i<=n;i++){

        for(j=0;j<=C;j++){

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

               v[i][j]=0;

            }

            else if(w[i-1]>j){

               v[i][j]=v[i-1][j];

            }

            else{

               v[i][j]=max(v[i-1][j],p[i-1]+v[i-1][j-w[i-1]]);

            }

        }

    }


    return v[n][C];

}

int main(){

    int n,w[50],p[50],i,C,a;

    printf("Enter the no. of items: ");

    scanf("%d",&n);

    printf("\nEnter the weight and profit: \n");

    for(i=0;i<n;i++){

        scanf("%d %d",&w[i],&p[i]);

    }
    printf("\nEnter the size of kanpsack: ");
    scanf("%d",&C);
    a=knapsack(n,C,w,p);
    printf("%d",a);
    return 0;
}
```

8) Floyd’s Algorithm

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

#define INF 99999
 void print(int distance[100][100],int n){
   printf("Shortest distance between every pair of vertices: \n");
   for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
       if(distance[i][j]==INF){
         printf("%7s","INF");
       }
       else{
         printf("%7d",distance[i][j]);
       }
     }
     printf("\n");
   }
 }
 void Floyd(int graph[100][100],int n){
   int distance[100][100];
   for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
       distance[i][j]=graph[i][j];
     }
   }
   for(int k=0;k<n;k++){
     for(int i=0;i<n;i++){
       for(int j=0;j<n;j++){
         if(distance[i][j] > distance[i][k]+distance[k][j]){
           distance[i][j]=distance[i][k]+distance[k][j];
         }
       }
     }
   }
   print(distance,n);
 }

int main()
{
    int n;
    printf("Enter the no. of vertices: ");
    scanf("%d",&n);
    int graph[100][100];
    printf("\nEnter the adjacency matrix:\n ");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d", &graph[i][j]);
        }
    }
    Floyd(graph,n);
    return 0;
}
```

9) Connected using DFS

```
#include <stdio.h>
#include <stdlib.h>
int n,visited[100],adj[100][100];
void dfs(int u){
    int v;
    visited[u]=1;
    for(v=0;v<n;v++){
        if(adj[u][v] && !visited[v]){
                printf("%d->%d\n",u,v);
            dfs(v);
        }
    }
}
int main()
{
    int i,j,count=0;
    printf("Enter the no. of vertices: ");
    scanf("%d",&n);
    for(i=0;i<n;i++){
        visited[i]=0;
    }
    for(j=0;j<n;j++){
        adj[i][j]=0;
    }
    printf("\nEnter the elements in array: \n");
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            scanf("%d",&adj[i][j]);
        }
    }
    dfs(0);
    printf("\n");
    for(i=0;i<n;i++){
        if(visited[i]){
            count++;
        }
    }
    if(count==n){
        printf("\nConnected");
    }
    else{
        printf("Not connected");
    }
    return 0;
}
```

10)Prims Algorithm

```
#include <stdio.h>

#include <stdbool.h>

#include <limits.h>


#define V 5  // Number of vertices in the graph


int minKey(int key[], bool mstSet[]) {

   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++) {

      if (mstSet[v] == false && key[v] < min) {

         min = key[v];

         min_index = v;

      }

   }

   return min_index;

}


void primMST(int graph[V][V]) {

   int parent[V];  // Array to store constructed MST

   int key[V];     // Key values used to pick minimum weight edge in cut

   bool mstSet[V]; // To represent set of vertices included in MST


   // Initialize all keys as INFINITE

   for (int i = 0; i < V; i++) {

      key[i] = INT_MAX, mstSet[i] = false;

   }


   // Always include the first vertex in MST.

   // Make key 0 so that this vertex is picked as first vertex.

   key[0] = 0;     // First vertex is always picked as root of MST

   parent[0] = -1; // First node is always root of MST


   // The MST will have V vertices

   for (int count = 0; count < V - 1; count++) {

      // Pick the minimum key vertex from the set of vertices
```

```
        // not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of the adjacent vertices
        // of the picked vertex. Consider only those vertices which are
        // not yet included in MST
        for (int v = 0; v < V; v++) {
            // graph[u][v] is non zero only for adjacent vertices of m
            // mstSet[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {
                parent[v] = u, key[v] = graph[u][v];
            }
        }
    }

    // Calculate the total minimum cost of the MST
    int minCost = 0;
    for (int i = 1; i < V; i++) {
        minCost += graph[i][parent[i]];
    }

    // Print the total minimum cost of the MST
    printf("Minimum Cost of MST: %d\n", minCost);
}

int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

    // Find and print the minimum cost of the Minimum Spanning Tree (MST)
    primMST(graph);

    return 0;
}
```