

BUBBLE SORT

```
#include<stdio.h>
#include<dos.h>
#include<time.h>
#include<stdlib.h>
#include<conio.h>
void bubblesort(int a[1000],int n)
{
    int i,j,temp;
    for(j=1;j<n;j++)
    {
        for(i=0;i<n-j;i++)
        {
            if(a[i]>a[j+1])
            {
                temp=a[i];
                a[i]=a[j+1];
                a[j+1]=temp;
            }
        }
        delay(10);
    }
}
void main()
{
    int a[10000],i,n;
    float res;
    clock_t end,start;
    clrscr();
    printf("Enter the number of elements to be sorted:\n");
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {
        a[i]=rand();
    }
    printf("unsorted array\n");
    for(i=0;i<n;++i)
        printf("%d\n",a[i]);
    start=clock();
    bubblesort(a,n);
    end=clock();
    res=(end-start)/CLK_TCK;
    printf("Sorted array is \n");
    for(i=0;i<n;i++)
        printf("%d\n",a[i]);
    printf("The time taken to sort %d elements in %f\n",n,res);
    getch();
}
```

SELECTION SORT

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
void selection_sort();
int a[1000], n;
void main()
{
    int i;
    float res;
    clock_t end,start;
    clrscr();
    printf("\nEnter size of an array: ");
    scanf("%d", &n);
    printf("\nEnter elements of an array:\n");
    for(i=0; i<n; i++)
        a[i]=rand();
    start=clock();
    selection_sort();
    end=clock();
    res=(end-start)/CLK_TCK;
    printf("\nAfter sorting:\n");
    for(i=0; i<n; i++)
        printf("%d\n", a[i]);
    printf("Time taken to sort the %d elements is %f",n,res);
    getch();
}
void selection_sort()
{
    int i,j,min,temp;
    for (i=0; i<n; i++)
    {
        min = i;
        for (j=i+1; j<n; j++)
        {
            if (a[j] < a[min])
                min = j;
        }
        delay(20);
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}
```

MERGE SORT

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<time.h>
#include<dos.h>
#include<stdlib.h>
void simple_merge(int a[],int low,int mid,int high)
{
    int i,j,k,c[10000];
    i=low;
    j=mid+1;
    k=low;
    while(i<=mid && j<=high)
    {
        if(a[i]<a[j])
        {
            c[k]=a[i];
            i++;
            k++;
        }
        else
        {
            c[k]=a[j];
            j++;
            k++;
        }
    }
    while(i<=mid)
    {
        c[k]=a[i];
        i++;
        k++;
    }
    while(j<=high)
    {
        c[k]=a[j];
        j++;
        k++;
    }
    for(i=low;i<=high;i++)
    {
        a[i]=c[i];
    }
}
void mergesort(int a[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid=(low+high)/2;
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        simple_merge(a,low,mid,high);
        delay(10);
    }
}
```

```

void main()
{
    int a[10000],n,i,mid,low,high;
    float res;
    clock_t end,start;
    clrscr();
    printf("Enter the size of an array\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        a[i]=rand();
    }
    start=clock();
    mergesort(a,0,n-1);
    end=clock();
    printf("The sorted array is \n");
    for(i=0;i<n;i++)
    {
        printf("%d\n",a[i]);
    }
    res=(end-start)/CLK_TCK;
    printf("Time taken to sort to %d elements %f\n",n,res);
    getch();
}

```

QUICK SORT

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include<dos.h>
#include<time.h>
int partition(int a[],int low,int high)
{
    int temp,key,i,j;
    key=a[low];
    i=low;
    j=high+1;
    while(i<=j)
    {
        do
        {
            i=i+1;
        }
        while(key<a[i]);
        do
        {
            j=j-1;
        }
        while(key<a[j]);
        if(i<j)
        {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    }
    temp=a[low];
    a[low]=a[j];
    a[j]=temp;
    return j;
}
void quicksort(int a[],int low,int high)
{
    int k;
    if(low<high);
    {
        k=partition(a,low,high);
        quicksort(a,low,k-1);
        quicksort(a,k+1,high);
    }
    delay(10);
}
void main()
{
    int a[1000],n,i,mid,low,high;
    float res;
    clock_t end,start;
    clrscr();
    printf("Enter the size of array\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
```

```
{
    a[i]=rand();
}
printf("The unsorted array\n");
for(i=0;i<n;i++)
{
    printf("%d\n",a[i]);
}
start=clock();
quicksort(a,0,n-1);
end=clock();
res=(end-start)/CLK_TCK;
printf("The sorted array\n");
for(i=0;i<n;i++)
{
    printf("%d\n",a[i]);
}
res=(end-start)/CLK_TCK;
printf("Time taken to sort %d element is %f\n",n,res);
getch();
}
```

TOPOLOGICAL ORDER

```
#include<stdio.h>
#include<conio.h>
void find_indegree(int a[10][10],int n,int indegree[10])
{
    int j,i,sum;
    for(j=0;j<n;j++)
    {
        sum=0;
        for(i=0;i<n;i++)
            sum+=a[i][j];
        indegree[j]=sum;
    }
}
void toposort(int a[10][10],int n)
{
    int u,v,t[10],s[10],indegree[10],top,k,i;
    find_indegree(a,n,indegree);
    top=-1;
    k=0;
    for(i=0;i<n;i++)
        if(indegree[i]==0)
            s[++top]=i;
    while(top!=1)
    {
        u=s[top--];
        t[k++]=u;
        for(v=0;v<n;v++)
        {
            if(a[u][v]==1)
            {
                indegree[v]--;
                if(indegree[v]==0)
                    s[++top]=v;
            }
        }
    }
    printf("Topological sorting is\n");
    for(i=0;i<n;i++)
        printf("%d\t",t[i]);
    return;
}
void main()
{
    int n,a[10][10],i,j;
    clrscr();
    printf("Enter the number of nodes\n");
    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]);
    toposort(a,n);
    getch();
}
```

WARSHALL ALGORITHM

```
#include<stdio.h>
#include<conio.h>
void warshal(int n,int a[100][100],int p[100][100])
{
    int i,j,k;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            p[i][j]=a[i][j];
        }
    }
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                if(p[i][k]==1 && p[k][j]==1)
                    p[i][j]=1;
            }
        }
    }
}

void main()
{
    int n,i,j,a[100][100],p[100][100];
    clrscr();
    printf("Enter the no of nodes\n");
    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]);
    warshal(n,a,p);
    printf("The transitice closure matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",p[i][j]);
        }
        printf("\n");
    }
    getch();
}
```


KNAPSACK

```
#include<stdio.h>
#include<conio.h>
int max(int a ,int b);
void optimal(int v[10][10],int n,int m,int p[10],int w[10])
{
    int i,j;
    for(i=0;i<=n;i++)
        for(j=0;j<=m;j++)
            v[i][j]=0;
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=m;j++)
        {
            if(i==0 || j==0)
                v[i][j]=0;
            else if(j<w[i])
                v[i][j]=v[i-1][j];
            else
                v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);
        }
    }
    printf("The resultant matrix is \n");
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=m;j++)
            printf("%d\t",v[i][j]);
        printf("\n");
    }
    printf("The optimal solution is %d \n",v[n][m]);
}

int max(int a,int b)
{
    return (a>b)? a:b;
}

void main()
{
    int i,j,n,m,w[10],p[10],v[10][10];
    clrscr();
    printf("Enter the no. of objects \n");
    scanf("%d",&n);
    printf("Enter the weight of the objects \n");
    for(i=1;i<=n;i++)
        scanf("%d",&w[i]);
    printf("Enter the profits of the object \n");
    for(i=1;i<=n;i++)
        scanf("%d",&p[i]);
    printf("Enter the capacity of the knapsack \n");
    scanf("%d",&m);
    optimal(v,n,m,p,w);
    getch();
}
```

DIJKSTRA'S ALG

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define infinity 999
int dijkstra(int n,int cost[100][100],int visited[100],int p[100],int distance[100],int source,int dest)
{
    int i,u,v,mincost,j;
    for(i=0;i<n;i++)
    {
        distance[i]=cost[source][i];
        p[i]=source;
        visited[i]=0;
    }
    visited[source]=1;
    for(i=0;i<n;i++)
    {
        u=-1;
        mincost=infinity;
        for(j=0;j<n;j++)
            if((visited[j]==0)&&(distance[j]<mincost))
            {
                mincost=distance[j];
                u=j;
            }
        if(u==-1)
            return;
        if(u==dest)
            return;
        visited[u]=1;
        for(v=0;v<n;v++)
            if((visited[v]==0)&&((distance[u]+cost[u][v]<distance[v])))
            {
                distance[v]=cost[u][v]+distance[u];
                p[v]=u;
            }
    }
}

void main()
{
    int n,a[100][100],visited[100],p[100],distance[100],source,dest,i,j;
    clrscr();
    printf("enter the no of verticesin graph\n");
    scanf("%d",&n);
    printf("enter adjacency matrix\n 999 for no edge\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    printf("enter source node\n");
    scanf("%d",&source);
    for(dest=0;dest<n;dest++)
    {
        dijkstra(n,a,visited,p,distance,source,dest);
        if(distance[dest]==infinity)
            printf("\n node=%d is not reachable from %d",dest,source);
        else
        {

```

```

        printf("path from %d to %d is \n",dest,source);
        i=dest;
        while(i!=source)
        {
            printf("%d<- ",i);
            i=p[i];
        }
        printf("%d\n",i);
        printf("\n distance =%d",distance[dest]);
    }
}
getch();
}

```

KRUSKAL'S ALG

```
#include<stdio.h>
#include<conio.h>
#define infinity 999
int find(int v,int s[])
{
    while(s[v]!=v)
        v=s[v];
    return v;
}
void krushkal(int n,int c[100][100])
{
    int t[100][2],u,v,s[100];
    int sum=0;
    int count=0;
    int i,j,k=0,min;
    for(i=0;i<n;i++)
        s[i]=i;
    while(count<n-1)
    {
        min=infinity;
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
            {
                if(c[i][j]!=0 && c[i][j]<min)
                {
                    min=c[i][j];
                    u=i;
                    v=j;
                }
            }
        if(min==999)break;
        i=find(u,s);
        j=find(v,s);
        if(i!=j)
        {
            t[k][0]=u;
            t[k][1]=v;
            count++;
            k++;
            sum=sum+min;
            s[j]=i;
        }
        c[u][v]=999;
        c[v][u]=999;
    }
    if(count==(n-1))
    {
        printf("Spanning tree exists\n");
        printf("Edges are\n");
        for(i=0;i<n-1;i++)
            printf("%d->%d\n",t[i][0],t[i][1]);
        printf("sum=%d\n",sum);
    }
    else
        printf("Spanning tree does not exist\n");
}
```

```
void main()
{
    int n,c[100][100],i,j;
    clrscr();
    printf("Enter the number of nodes\n");
    scanf("%d",&n);
    printf("Enter the cost adjacency matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    krushkal(n,c);
    getch();
}
```

BFS

```
#include<stdio.h>
#include<stdlib.h>
void bfs(int a[100][100],int n,int source)
{
    int f,r,q[100],u,v,i,s[100];
    for(i=0;i<n;i++)
    {
        s[i]=0;
    }
    f=0;
    r=0;
    q[r]=source;
    s[source]=1;
    while(f<=r)
    {
        u=q[f++];
        for(v=1;v<n;v++)
        {
            if(a[u][v]==1 && s[v]==0)
            {
                s[v]=1;
                q[++r]=v;
            }
        }
    }
    for(i=0;i<n;i++)
    {
        if(s[i]==0)
        {
            printf("Vertex %d is not reachable\n",i);
        }
        else
        {
            printf("Vertex %d is reachable\n",i);
        }
    }
}
void main()
{
    int n,adj[100][100],i,j,source;
    clrscr();
    printf("Enter the number of nodes\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&adj[i][j]);
        }
    }
    printf("Enter the source vertex\n");
    scanf("%d",&source);
    bfs(adj,n,source);
    getch();
}
```

DFS

```
#include<stdio.h>
#include<conio.h>
void read(int n,int cost[10][10])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&cost[i][j]);
        }
    }
}
void dfs(int n,int cost[20][20],int u,int s[]){
    int v;
    s[u]=1;
    for(v=0;v<n;v++)
    {
        if(cost[u][v]==1 && s[v]==0)
            dfs(n,cost,v,s);
    }
}
int connectivity(int n,int cost[20][20]){
    int i,j,flag,s[10];
    for(j=0;j<n;j++)
    {
        for(i=0;i<n;i++)
            s[i]=0;
        dfs(n,cost,j,s);
        flag=0;
        for(i=0;i<n;i++)
            if(s[i]==0)
                flag=1;
        if(flag==0)
            return 0;
    }
    return 0;
}
void main(){
    int n,cost[20][20],flag,i,j;
    clrscr();
    printf("Enter the number of nodes\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            scanf("%d",&cost[i][j]);
    }
    flag=connectivity(n,cost);
    if(flag==1)
        printf("graph is connected\n");
    else
        printf("graph is not connected\n");
    getch();
}
```

SUBSET

```
#include<stdio.h>
#include<conio.h>
int count,w[10],d,x[10];
void subset(int cs,int k,int r)
{
    int i;
    x[k]=1;
    if(cs+w[k]==d)
    {
        printf("Subset soluton=%d\n",++count);
        for(i=0;i<=k;i++)
        {
            if(x[i]==1)
                printf("%d\t",w[i]);
        }
        printf("\n");
    }
    else if(cs+w[k]+w[k+1]<=d)
        subset(cs+w[k],k+1,r-w[k]);
    if((cs+r-w[k]>=d) && (cs+w[k+1]<=d))
    {
        x[k]=0;
        subset(cs,k+1,r-w[k]);
    }
    getch();
}

void main()
{
    int sum=0,i,n;
    clrscr();
    printf("Enter no of elements\n");
    scanf("%d",&n);
    printf("Enter the elements in ascending order\n");
    for(i=0;i<n;i++)
        scanf("%d",&w[i]);
    printf("Enter the required sum\n");
    scanf("%d",&d);
    for(i=0;i<n;i++)
        sum+=w[i];
    if(sum<d)
    {
        printf("no solution exist\n");
        return;
    }
    printf("solution is \n");
    count=0;
    subset(0,0,sum);
    getch();
}
```


TRAVELLING SALES PERSON PROBLEM (FROM INTERNET)

```
#include<stdio.h>
int ary[10][10],completed[10],n,cost=0;
void takeInput()
{
    int i,j;
    printf("Enter the number of villages: ");
    scanf("%d",&n);
    printf("\nEnter the Cost Matrix\n");
    for(i=0;i < n;i++)
    {
        printf("\nEnter Elements of Row: %d\n",i+1);
        for( j=0;j < n;j++)
            scanf("%d",&ary[i][j]);
        completed[i]=0;
    }
    printf("\n\nThe cost list is:");
    for( i=0;i < n;i++)
    {
        printf("\n");
        for(j=0;j < n;j++)
            printf("\t%d",ary[i][j]);
    }
}

void mincost(int city)
{
    int i,ncity;
    completed[city]=1;
    printf("%d--->",city+1);
    ncity=least(city);
    if(ncity==999)
    {
        ncity=0;
        printf("%d",ncity+1);
        cost+=ary[city][ncity];
        return;
    }
    mincost(ncity);
}

int least(int c)
{
    int i,nc=999;
    int min=999,kmin;
    for(i=0;i < n;i++)
    {
        if((ary[c][i]!=0)&&(completed[i]==0))
            if(ary[c][i]+ary[i][c] < min)
            {
                min=ary[i][0]+ary[c][i];
                kmin=ary[c][i];
                nc=i;
            }
    }
    if(min!=999)
        cost+=kmin;
    return nc;
}
```

```
int main()
{
    takeInput();
    printf("\n\nThe Path is:\n");
    mincost(0); //passing 0 because starting vertex
    printf("\n\nMinimum cost is %d\n ",cost);
    return 0;
}
```

PRIM'S ALG

```
#include<stdio.h>
#include<conio.h>
#define infinity 999
void prim(int cost[100][100],int n,int visited[100],int p[100],int d[100])
{
    int mincost,count=0,i,j,v,source,u,t[100][2],k,sum;
    k=0;
    sum=0;
    mincost=infinity;
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        if(cost[i][j]<mincost)
        {
            mincost=cost[i][j];
            source=i;
        }
    for(i=0;i<n;i++)
    {
        visited[i]=0;
        p[i]=source;
        d[i]=cost[source][i];
    }
    visited[source]=1;
    for(i=0;i<n;i++)
    {
        u=-1;
        mincost=infinity;
        for(j=0;j<n;j++)
        {
            if((visited[j]==0)&&(d[j]<mincost))
            {
                mincost=d[j];
                u=j;
            }
        }
        if(u==-1)break;
        visited[u]=1;
        t[k][0]=p[u];
        t[k][1]=u;
        count++;
        k++;
        sum=sum+mincost;
        for(v=0;v<n;v++)
        if((visited[v]==0)&&(cost[u][v]<d[v]))
        {
            d[v]=cost[u][v];
            p[v]=u;
        }
    }
    if(count==n-1)
    {
        printf("spanning tree exists\n edges of the spanning tree is \n");
        for(i=0;i<n-1;i++)
            printf("%d->%d\n",t[i][0],t[i][1]);
        printf("Sum =%d\n",sum);
    }
}
```

```
        else
            printf("Spanning tree does not exist\n");
    }
void main()
{
    int n,i,j,cost[100][100],visited[100],p[100],d[100];
    clrscr();
    printf("Enter the number of vertices int graph\n");
    scanf("%d",&n);
    printf("Enter the cost adjacency matrix\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&cost[i][j]);
    prim(cost,n,visited,p,d);
    getch();
}
```

FLOYD'S ALG

```
#include<stdio.h>
#include<conio.h>
#define infinity 99
int min(int a,int b)
{
    return(a<b)?a:b;
}
void floyd(int n,int a[10][10],int d[10][10])
{
    int i,j,k;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            d[i][j]=a[i][j];
        }
    }
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
            }
        }
    }
}
void main()
{
    int i,j,k,a[10][10],d[10][10],n;
    clrscr();
    printf("Enter the no of nodes\n");
    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]);
        }
    }
    printf("distance matrix\n");
    floyd(n,a,d);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",d[i][j]);
        }
        printf("\n\n");
    }
    getch();
}
```

N QUEEN'S

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define TRUE 1
#define FALSE 0
void print_solution(int n,int x[])
{
    char c[10][10];
    int i,j;
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    c[i][j]='x';
    for(i=1;i<=n;i++)
    c[i][x[i]]='Q';
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("%c\t",c[i][j]);
        }
        printf("\n");
    }
}
int place(int x[],int k)
{
    int i;
    for(i=1;i<k;i++)
    {
        if((x[i]==x[k]) || abs(i-k)==abs(x[i]-x[k]))
            return FALSE;
    }
    return TRUE;
}
void nqueens(int n)
{
    int x[10],count=0,k=1;
    x[k]=0;
    while(k!=0)
    {
        x[k]+=1;
        while((x[k]<=n) && (!place(x,k)))
        {
            x[k]+=1;
        }
        if(x[k]<=n)
        {
            if(k==n)
            {
                count++;
                printf("Solution is %d\n",count);
                print_solution(n,x);
            }
            else
            {
                k++;
                x[k]=0;
            }
        }
    }
}
```

```
        }
    }
    else
        k--;
}
if(n==2 || n==3)
{
    printf("There is no solution\n");
}
}
void main()
{
    int n;
    clrscr();
    printf("Enter the no of queens\n");
    scanf("%d",&n);
    nqueens(n);
    getch();
}
```

HORSPPOOL'S ALG

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define MAX 500
int t[MAX];
void shifttable(char p[]) {
    int i,j,m;
    m=strlen(p);
    for (i=0;i<MAX;i++)
        t[i]=m;
    for (j=0;j<m-1;j++)
        t[p[j]]=m-1-j;
}
int horspool(char src[],char p[]) {
    int i,j,k,m,n;
    n=strlen(src);
    m=strlen(p);
    printf("\nLength of text=%d",n);
    printf("\n Length of pattern=%d",m);
    i=m-1;
    while(i<n) {
        k=0;
        while((k<m)&&(p[m-1-k]==src[i-k]))
            k++;
        if(k==m)
            return(i-m+1); else
            i+=t[src[i]];
    }
    return -1;
}
void main() {
    char src[100],p[100];
    int pos;
    clrscr();
    printf("Enter the text in which pattern is to be searched:\n");
    gets(src);
    printf("Enter the pattern to be searched:\n");
    gets(p);
    shifttable(p);
    pos=horspool(src,p);
    if(pos>=0)
        printf("\n The desired pattern was found starting from position %d",pos+1); else
        printf("\n The pattern was not found in the given text\n");
    getch();
}
```