1).MERGESORT

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MAX 10000
int A[MAX], B[MAX];
int main(){int n, i, j;
int low, high, mid;
clock_t s, e;
double cpu_exe_t;
printf("\n Please enter the size
of the array: ");scanf("%d", &n);
low = 0;high = n-1;
for(i=0; i<n; i++){
A[i] = rand() % 100;}
printf("\n The array ele are: \n");
for(i=0; i<n; i++){
printf("%d\t", A[i]);}s=clock();
//for(j=0;i<10000;j++) //Delay loops
for(i=0; i<1000; i++)
for(j=0; j<1000; j++)
Mergesort(low, high);e=clock();
cpu_exe_t = (double)(e-s)/CLK_TCK;
printf("\nThe sorted array is: \n");
for(i=0; i<n; i++){
printf("%d\t",A[i]);}
printf("\n CPU execution time is
%lf", cpu_exe_t);return 0;}
Mergesort(int low, int high){
int mid;if(low < high){
mid = (low + high) / 2;
Mergesort (low, mid);
Mergesort (mid+1, high);}
}Merge(int low, int mid, int high)
{int i=low, j=mid+1, k=low;
while(i<=mid && j<=high)
{if(A[i] <= A[j]){
B[k] = A[i];i++;}else{
B[k] = A[j];j++;}
k++;}while(i <= mid){
B[k] = A[i];k++;
i++;}while(j <= high)
{B[k] = A[j];
k++;j++;}
for(i=low; i<=high; i++)
{A[i] = B[i];}}
```

■ ALGORITHM

```
Meregesort(A[0..n-1])
if n>1
copy A[0..[n/2]-1] to B[0..[n/2] - 1]
copy A[[n/2]..n - 1] to C[0..[n/2] - 1]
Mergesort(B[0..(n/2] - 1])
Mergesort(C[0..[n/2] - 1])
Merge(B, C, A)
Merge(B(0..p-1), C[0..q-1], A[0..p+q-1)
i<-0; j<-0; k<-0
while i < p and j < q do
if B[i]<= C[il
A[k]<-B[i]; i<-i + 1
else A[k]<-C[j]; j<-j+1
k<-k+1
if i= p
copy C[j..q-1]to A[k..p+q-1]
else copy B[i..p-1] to A[k..p+q-1]
```

2) QUICKSORT

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20000
int A[MAX];
void Quicksort(int low, int high);
int_Partition ( int low, int high);
void swap(int *p, int *q);
int main(){int n, i, j;
int low, high;clock_t s, e;
double cpu_exe_t;
printf("\nEnter size of array: \n");
scanf("%d", &n);for(i=0; i<n; i++)
{A[i]=rand()%100;}
printf("\nThe array ele are: \n");
for(i=0; i<n; i++){
printf("%d\t", A[i]);}
s= clock();
//for(j=0; j<1000; j++)
for(i=0; i<10000; i++){
low =0;high = n-1;
Quicksort(low,high);}
e=clock();
cpu_exe_t=(double)(e-s)/CLK_TCK;
printf("\n sorted array is: \n");
for(i=0; i<n; i++){
printf("%d\t", A[i]);}
printf("\n CPU execution
time is %f", cpu_exe_t);return 0;}
int Partition (int low, int high)
{int i, j;int pivot=A[low];
i=low; j=high+1;
while(i<j){do{++i;
}while(A[i]<=pivot);do
{--j;}while(A[j]>pivot);
if(i<j){swap(&A[i], &A[j]);
/*t=A[i];A[i]=A[j];
A[j]=t;*/}}
swap(&A[low], &A[j]);
/*t=A[low];A[low]=A[j];
A[j]=t;*/return j;}
void Quicksort (int low, int high)
{int j;if(low<high){
j=Partition(low, high);
Quicksort(low, j-1);
Quicksort(j+1, high);}}
void swap( int *a, int *b){
int t;t =*a;*a=*b;*b=t;}
```

■ ALGORITHM

```
Quicksort(A[L..r])
if l <r
s<-Partition(A(l.r])
Quicksort(A[l..s-1])
Quicksort(A[s+1..r])
PARTITION
Partition(A[l..r])
p<-A[l]
i<-l; j<-r+1
repeat
repeat i <- i+1 until A[i]>=p
repeat j <-j-1 until A[i]<=p
swap(A[i], A[j])
until i>=j
swap(A[i], A[j])
swap(A[l), A[j])
return j
```

3)INSERTION

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MAX 1000
int A[MAX];
void insertion_sort(int n){
int i=0, j=0, key=0;
for(i=1; i<n; i++){key=A[i];
j=i-1;while(((j>=0 && key<A[j])
){A[j+1]= A[j];j=j-1;
}A[j+1]=key;}}int main()
{clock_t s,e;
double cpu_exe_t;
int i=0, j=0, n;
printf("How many nos :");
scanf("%d", &n);
for(i=0; i<n;i++ )
A[i]= rand()%100;
printf("The array ele are: ");
for(i=0; i<n; i++){
printf("%d\t",A[i]);}s=clock();
for(j=0; j<1000; j++)
for(i=0; i<1000; i++)
insertion_sort(n);e=clock();
cpu_exe_t = (double)(e-s)/CLK_TCK;
printf("\n Order of the sorted ele is:");
for(i=0; i<n; i++)
printf("%d\t", A[i]);
printf("\n CPU execution time is:
%1f \n",cpu_exe_t);return 0;}
```

■ ALGORITHM

```
InsertionSort(A[0..n - 1])
for i<-1 to n-1 do
v <- A[i]
j <- i-1
while j ≥ 0 and A[j] > v do
A[j + 1]<-A[j]
j←j－1
A[j+1]<-v
```

4)HEAPSORT

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MAX 10000
void exchange(int *p, int *q){
int t;t=*p;*p =*q;*q=t;}
void HeapSort(int *A, int n){
int i;for(i=n/2;i>=1; i--){
Heapify(A,n,i);}
for(i = n; i>=2; i--){
exchange(&A[i], &A[1]);
Heapify(A, i-1, 1);}}
void Heapify(int *A, int n, int i){
int largest, l, r;largest = i;
l=2*i; r=2*i+1;
if(l <= n && A[l] > A[largest]){
largest = l;}
if(r <= n && A[r] > A[largest]){
largest = r;}if(largest != i){
exchange(&A[largest], &A[i]);
Heapify(A, n, largest);}}
int main(){
int i, n, A[MAX], k , j;
srand(1);time_t start, end;
double cpu_exe_time;
printf("Heap Sort..\n");
printf("Enter the value of n:");
scanf("%d", &n);
for(k=0; k<n;k++){
A[k] = rand() % 100 + 1;}
printf("\nArray b4 sorting: \n");
for(i = 1; i<=n; i++){
printf("%d ", A[i]);}
start=clock();
for(int i=0; i<10000; i++){
HeapSort(A,n);}
end = clock();
printf("\nArray after sort: \n");
for(i = 1; i<=n; i++){
printf("%d ", A[i]);}
cpu_exe_time=(double)
(end-start)/CLK_TCK;
printf("\nExecution time heap
sort is %lf", cpu_exe_time);}
```

■ ALGORITHM

```
HeapBottomUp(H[1..n])
for i<-[n /2] down to 1 do
k<-i; v<-H[k]
heap<-false
while not heap and 2 * k <= n do
j<-2 * k
if j<n
if H[j]<H[j+1]j<-j+1
if v>= H[I]
heap <-true
else H[k]<-H[I]; k<-j
H[k]<-v
```

5)DIJKSTRAS

```
#include<stdio.h>
#define infinity 999
void dij(int n,int v,int cost
[10][10],int dist[100]){
int i,u,count,w,flag[10],min;
for(i=1;i<=n;i++)
flag[i]=0,dist[i]=cost[v][i];
count=2;while(count<=n){
min=99;for(w=1;w<=n;w++)
if(dist[w]<min && !flag[w])
min=dist[w],u=w;flag[u]=1;
count++;for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w])
&& !flag[w])
dist[w]=dist[u]+cost[u][w];
}}void main(){
int v,n,i,j,cost[10][10],dist[10];
printf("\nEnter no of Nodes:\n");
scanf("%d",&n);
printf("\nEnter cost 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]=infinity;
}}printf("\nEnter source matrix:");
scanf("%d",&v);dij(n,v,cost,dist);
printf("\nShortest path :\n");
for(i=1;i<=n;i++)if(i!=v)
printf("%d->%d,cost=%d\n"
,v,i,dist[i]);}
```

■ ALGORITHM

```
Dijkstra(G, s)
Initialize(Q)
for every vertex v in V do
dv<-∞; pv<-null
Insert(Q,v,dv)
ds<-0; Decrease(Q,s,ds)
Vt<-∅
for i<-0 to |V|-1 do
u*<-DeleteMin(Q)
Vt<-VtU(u*)
for every vertex u in V - Vt
if du*+w(u*, u)<du
du<-du*+w(u*, u); pu<-u*
Decrease(Q,u,du)
```

6) PRIMS

```
#include<stdio.h>
#include<stdlib.h>
int a,b,u,v,n,i,j,ne=1;int
visited[10]={0},min,mincost=0,cost
[10][10];void main(){printf("\nEnter no
nodes:");scanf("%d",&n);
printf("\nEnter the adj mat:\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 Min cost=%d\n",mincost);}
```

■ ALGORITHM

```
Prim(G)
Vt<-{vo}
Et<-∅
for i<-1 to |V|<-1 do
find a minimum-weight edge e* =
(v*, u*)
 among all the edges (v, u)
such that v is in Vt and u is in V - Vt
Vt<-Vt U (u*)
Et<-EtU (e*)
return Et
```

7) FLOYDS

```
#include <stdio.h>
#include <stdlib.h>
int min(int,int);
void printM(int D[10][10],int n)
{int i,j;
for(i=1;i<=n;i++){
for(j=1;j<=n;j++){
printf("%d\t",D[i][j]);}
printf("\n");}}
void floyds(int D[10][10],int n)
{ int i,j,k;for(k=1;k<=n;k++){
printf("With %d as intermediate Vertx",k);
printf("Cost Mat now: \n");
for(i=1;i<=n;i++){for(j=1;j<=n;j++)
{if(i==j)D[i][j]=0;
else{D[i][j]= min(D[i][j],
(D[i][k]+D[k][j]));}}
}printM(D,n);}}
int min(int a,int b){
return(a<b)?a:b;}
int main(){
int D[10][10],w,n,e,u,v,i,j;
printf("Enter value of vertices ");
scanf("%d",&n);
printf("\n Enter cost of mat:\n");
for(i=1;i<=n;i++){
for(j=1;j<=n;j++)
{scanf("%d",&D[i][j]);}
}printf("\n initial Cost Mat:\n");
printM(D,n);floyds(D,n);
printf("\n The final Cost Mat:\n");
printM(D,n);
printf("\n The Shortst paths are:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{if(i!=j)printf("\n <%d,%d> ====>%d",i,j,D[i]
[j]);
}return 0;}
```

■ ALGORITHM

```
Floyd(W[1..n, 1..n])
D<-W
for k<-1 to n do
for i<- 1 to n do
for j<-1 to n do
D[i,j]<-min{D[i,j], D[i,k]+D[k,j]}
return D
```

8)KNAPSACK

```
#include<stdio.h>
#define MAX 200
int V[MAX][MAX] = {0};
int res [200]={0};
int count = 0;
int max (int a, int b)
{return (a>b)? a:b;}
int knapSack(int W, int
wt[], int val[], int n)
{int i, j;for(i=0; i<=n; i++)
{for(j=0; j<=W; j++){
if(i==0 | | j==0){
V[i][j]=0;}
else if (wt [i-1] <= j){
V[i][j] = max(val [i-1] + V[i-1]
[j-wt[i-1]], V[i-1][j]);}else
{V[i][j] = V[i-1][j];}}
int k, m;for(k=0; k<=n; k++)
{for(m=0; m<=W; m++){
printf("%d ", V[k][m]);}
printf("\n");}printf("\n");}
i=n;j=W;
while(i>0 && j>0){
if(V[i][j]!= V[i-1][j]){
res[count++] = i;
j = j-wt[i-1];i--;}
else i--;}return V[n][W];}
int main(){
int i, n, W, optsoln;
int val[20], wt[20];
printf("\nEnter no of items:\n");
scanf("%d", &n);
printf("\nEnter wght of items:\n");
for(i=0; i<n; i++)scanf("%d", &wt[i]);
printf("\nEnter values:\n");
for(i=0; i<n; i++)scanf("%d", &val[i]);
printf("\nEnter the knapsack
capacity: ");scanf("%d", &W);
optsoln=knapSack(W, wt, val, n);
printf("\n optimal sol is: %d",optsoln);
printf("\nThe optimal subset\n");
printf("Items included in knapsack are: ");
for(i=count-1; i>=0; i--)
printf("\t%d", res[i]);printf("\n");
return 0;}
```

■ ALGORITHM

```
MFKnapsack(i,j)
if V[i,j]<0
if j<Weights[i]
value<-MFKnapsack(i-1,j),
  Values[i]+MFKnapsack(i-1,j-Weights[i]))
V[i,j]<-value
return V[i,j]
```

9) SUBSETS

```
#include <stdio.h>
#include <stdlib.h>
#define true 1#define false 0
#define max 50
int inc[max],w[max],sum,n;
int prom(int i,int wt,int t){
return(((wt+t)>=sum)&&((wt==
sum)||(wt+w[i+1]<=sum)));}
void sumset(int i,int wt,int t){
int j;if(prom(i,wt,t)){
if(wt==sum){
printf("\n{\t");
for(j=0;j<=i;j++)if(inc[j])
printf("%d\t",w[j]);printf("}\n");
}else{
inc[i+1]=true;
sumset(i+1,wt+w[i+1],t-w[i+1]);
inc[i+1]=false;
sumset(i+1,wt,t-w[i+1]);
}}}
int main(int argc,char *argv[]){
int i,j,tmp,t=0;
printf("Enter how many nos to read");
scanf("%d",&n);
printf("Enter the value for all
nos");for(i=0;i<n;i++){
scanf("%d",&w[i]);t+=w[i];}
printf("Enter sum");
scanf("%d",&sum);
for(i=0;i<=n;i++)
for(j=0;j<n-1;j++)
if(w[j]>w[j+1]){
tmp=w[j]; w[j]=w[j+1];
w[j+1]=tmp;}
printf(" Given %d nos as I\p\n",n);
for(i=0;i<=n;i++)
printf("%d\t",w[i]);if(t<sum)
printf("not possible");else{
for(i=0;i<n;i++)inc[i]=0;
printf("\nAnd sol is as bellow \n");
sumset(-1,0,t);}return 0;}
```

■ ALGORITHM

```
Backtrack(X[1..i])
ifX[1..i] is a solution write X[1..i]
else
for each element x∈Si+1
consistent with X[1..i] and
the constraints do
X[i+1]<-x
Backtrack(X[1..i+1])
```

10) QUEENS

```
#include<stdio.h>
#include<math.h>
int a[30], count=0;
int place(int pos)
{int i;for(i=1; i<pos; i++)
{if((a[i]==a[pos]) ||((abs
(a[i]-a[pos])==abs(i-pos))))
return 0;}return 1;}
void printsol(int n){
int i, j;count ++;
printf("\n\nSol #%d\n\n",
count);for(i=1; i<=n; i++)
{for(j=1; j<=n; j++){
if(a[i]==j)printf("Q\t");
else printf("*\t");}
printf("\n");}}
void queen (int n){
int k=1;a[k]=0;
while(k!=0){
a[k]=a[k]+1;while(a[k]<=n
&& !place(k))
a[k]++;if(a[k]<=n){
if(k==n)printsol(n);
else{k++;a[k]=0;}}
elsek--;}}void main()
{int n;
printf("Enter no of quen\n");
scanf("%d", &n);queen(n);
printf("\nTotal no of soln
 =%d", count);}
```

■ ALGORITHM