QUICKSORT

```
#include<iostream.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
#define MAX 500
void qsort(int[],int,int);
int part(int[],int,int);
void main()
{
clrscr();
int a[MAX],i,n;
clock_t s,e,z;
s=clock();
cout<<"Enter the value of n:";
cin>>n;
for(i=0;i<n;i++)
a[i]=rand()%100;
cout<<"Before sorting:"<<"\t"<<"\n";
for(i=0;i<n;i++)
cout<<a[i]<<"\t";
qsort(a,0,n-1);
cout<<"After sort:"<<"\t"<<"\n";
for(i=0;i<n;i++)
cout<<a[i]<<"\t";
e=clock();
z=e-s;
cout<<"Time Taken:"<<z/CLOCKS_PER_SEC<<"sec"<<"\n"<<"\t";
getch();
}
void qsort(int a[],int low,int high)
```

```
{
int j;
if(low<high)
{
j=part(a,low,high);
qsort(a,j+1,high);
qsort(a,low,j-1);
}
}
int part(int a[],int low,int high)
{
int pivot,i,j,temp;
pivot=a[low];
i=low+1;
j=high;
while(1)
{
while(pivot>a[i]&&i<high)
i++;
while(pivot<a[j])
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
else
{
temp=a[j];
a[j]=a[low];
a[low]=temp;
return j; } } }
```

OUTPUT:

```
Enter the value of n:3

Before sorting:
46      30      82

After sort:
30      46      82

Time Taken:1.813187sec
```

TOPOLOGY

```
#include<iostream.h>
#include<conio.h>
int a[10][10],n,indegree[10];
void find_indegree()
{
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 topology()
{
int i,u,v,t[10],s[10],top=-1,k=0;
find_indegree();
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]=-1;
s[++top]=u;
}
}
}
cout<<"The Topological Sequence is :"<<"\n";
for(i=0;i<n;i++)
cout<<t[i]<<"\t";
}
void main()
{
int i,j;
cout<<"Enter the Number of Nodes:";
cin>>n;
cout<<"Enter the Adajency Matrix:"<<"\n";
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
cin>>a[i][j];
}
topology();
getch();
}
```

OUTPUT:

```
Enter the Number of Nodes: 3
Enter the Adajency Matrix:
3 4 5
2 5 6
1 2 4
The Topological   sequence is:

1400 4  0
```

TRAVELLING SALESMAN PROBLEM

```
#include<iostream.h>
#include<conio.h>
int ary[10][10],completed[10],n,cost=0;
void takeinput()
{
clrscr();
int i,j;
cout<<"Enter the Number of Villages:";
cin>>n;
cout<<"\n Enter the Cost Matrix:\n";
for(i=0;i<n;i++)
{
cout<<"Enter Element of Row:"<<i+1<<"\n";
for(j=0;j<n;j++)
cin>>ary[i][j];
completed[i]=0;
}
cout<<"\n\n The Cost list is:";
for(i=0;i<n;i++)
{
cout<<"\n";
for(j=0;j<n;j++)
cout<<"\t"<<ary[i][j];
}
}
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!=99)
cost+=kmin;
return nc;
}
void mincost(int city)
{
int i,ncity;
completed[city]=1;
cout<<city+1<<"-->";
ncity=least(city);
if(ncity==999)
{
ncity=0;
cout<<ncity+1;
cost+=ary[city][ncity];
return;
}
mincost(ncity);
}
int main()
{
takeinput();
cout<<"\n The Path is:\n";
mincost(0);
 cout<<"\n Minimum Cost is:"<<cost;
getch();
return 0;
}
```

OUTPUT:

```
Enter the Number of Villages:3
 Enter the Cost Matrix:
Enter Element of Row:1
2
3
4
Enter Element of Row:2
3
2
1
Enter Element of Row:3
4
2
3

 The Cost list is:
        2       3       4
        3       2       1
        4       2       3

 The Path is:

1-->2-->3-->1

 Minimum Cost is:10
```

PRIMS ALGORITHM

```
#include<iostream.h>
#include<conio.h>
int ne=1,min_cost=0;
void main()
{
clrscr();
int n,i,j,min,cost[20][20],a,b,u,v,source,visited[20];
cout<<"enter the number of matrices:\n";
cin>>n;
cout<<"enter the cost of matrix:\n";
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
cin>>cost[i][j];
if(cost[i][j]==0)
cost[i][j]=999;
}
}
for(i=0;i<n;i++)
visited[i]=0;
cout<<"enter the root node:";
cin>>source;
visited[source]=1;
cout<<"\n minimum cost spanning tree:\n";
while(ne<n)
{
min=999;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(cost[i][j]<min)
if(visited[i]!=0)
{
min=cost[i][j];
a=v=i;
b=v=j;
}
}
}
if(visited[v]==0||visited[v]==0)
{
cout<<"\n edge"<<ne++<<"\t("<<a<<"--->"<<b<<")"<<min;
min_cost=min_cost+min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
cout<<"\n minimum cost="<<min_cost<<"\n";
getch();
}
```

OUTPUT:

```
Enter the number of matrices:
5
Enter the cost of matrix:
0 4 5 6 1
0 3 7 2 1
4 3 2 0 2
2 4 0 4 1
4 3 1 4 2
Enter the root node:1

 Minimum cost Spanning tree:

 Edge1  (1--->5)1
 Edge2  (5--->3)1
 Edge3  (3--->2)3
 Edge4  (2--->4)2
 Minimum cost=7
```

DIJKSTRA’S ALGORITHM

```
#include<iostream.h>
void dij(int,int[20][20],int[20],int[20],int);
void main()
{
int i,j,n,visited[20],source,cost[20][20],d[20];
cout<<"enter no. of vertices:";
cin>>n;
cout<<"enter the cost adjacency matrix\n";
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
cin>>cost[i][j];
}
}
cout<<"\nEnter the source node:";
cin>>source;
dij(source,cost,visited,d,n);
for(i=1;i<=n;i++)
if(i!=source)
cout<<"\n shortest path from"<<source<<"to"<<i<<"is"<<d[i];}
void dij(int source,int cost[20][20],int visited[20],int d[20],int n)
{
int i,j,min,u,w;
for(i=1;i<=n;i++)
{
visited[i]=0;
d[i]=cost[source][i];
}
visited[source]=1;
d[source]=0;
for(j=1;j<=n;j++)
{
min=999;
for(i=1;i<=n;i++)
{
if(!visited[i])
{
if(d[i]<min)
{
min=d[i];
u=i;
}
}
}
visited[u]=1;
for(w=1;w<=n;w++)
{
if(cost[u][w]!=999 &&visited[w]==0)
{
if(d[w]>cost[u][w]+d[u])
d[w]=cost[u][w]+d[u];
}
}
}
}
```

OUTPUT:

```
Enter no. of vertices: 5

Enter the cost Adjacency matrix:

0      5          12        17      999
999   0          999      8        7
999   9999   0          9        999
999   999     999      0        999
999   999     999      999    0

Enter the source node: 1

 shortest  path from1 to 2 is 5
 shortest  path from1 to 3 is 12
 shortest  path from1 to 4 is 13
 shortest  path from1 to 5 is 12
```

N -QUEEN PROBLEM

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void queen(int,int);
int place(int,int);
int x[15],count=1;
void main()
{
int i,j,n;
clrscr();
cout<<"\n\t\t queen problem";
cout<<"\n\t\t **************";
cout<<"Enter the number of queens";
cin>>n;
queen(1,n);
getch();
}
void queen(int k,int n)
{
int i,j,m;
for(i=1;i<=n;i++)
{
if(place(k,i))
{
x[k]=i;
if(k==n)
{
cout<<"\n\t feasible solution:"<<count++;
cout<<"\n\t ******************:";
for(j=1;j<=n;j++)
{
cout<<"\n\t\t Row"<<j<<"--column"<<x[j]<<":\t|";
for(i=1;i<=n;i++)
{
if(i==x[j])
cout<<"Q|";
else
cout<<".|";
}
}
getch();
}
else
queen(k+1,n);
}
}
}
int place(int k,int i)
{
int j;
for(j=1;j<=k-1;j++)
if((x[j]==i)||(abs(x[j]-i)==abs(j-k)))
return 0;
return 1;
}
```

OUTPUT:

```
                              Queen problem
                             *************
Enter the number of queen: 4

    Feasible solution: 1
    ****************

        Row1--column2: |.|Q|.|.|
        Row2--column4: |.|.|.|Q|
        Row3--column1: |Q|.|.|.|
        Row4--column3: |.|.|Q|.|

    Feasible solution: 2
    ****************

        Row1--column3: |.|.|Q|.|
        Row2--column1: |Q|.|.|.|
        Row3--column4: |.|.|.|Q|
        Row4--column2: |.|Q|.|.|
```

4-Queens Problem

```
queens(N, Queens) :-
length(Queens, N),
board(Queens, Board, 0, N, _, _),
queens(Board, 0, Queens).

board([], [], N, N, _, _).
board([_|Queens], [Col-Vars|Board], Col0, N, [_|VR], VC) :-
Col is Col0+1,
functor(Vars, f, N),
constraints(N, Vars, VR, VC),
board(Queens, Board, Col, N, VR, [_|VC]).

constraints(0, _, _, _) :- !.
constraints(N, Row, [R|Rs], [C|Cs]) :-
arg(N, Row, R-C),
M is N-1,
constraints(M, Row, Rs, Cs).

queens([], _, []).
queens([C|Cs], Row0, [Col|Solution]) :-
Row is Row0+1,
select(Col-Vars, [C|Cs], Board),
arg(Row, Vars, Row-Row),
queens(Board, Row, Solution).
```

OUTPUT:

```
?- queens(4, Queens).
Queens = [2, 4, 1, 3]
```

ALPHA-BETA PRUNING

```
:- dynamic true/1, does/2.

role(player).

init(step(0)).
init(cell(1, 1, 1)). init(cell(1, 2, 2)). init(cell(1, 3, 3)).
init(cell(2, 1, 7)). init(cell(2, 2, 8)). init(cell(2, 3, 4)).
init(cell(3, 1, 6)). init(cell(3, 2, 5)). init(cell(3, 3, b)).

legal(player, move(Row, Col)) :-
    true(cell(U, Col, b)),
    (succ(Row, U) ; pred(Row, U)).

legal(player, move(Row, Col)) :-
    true(cell(Row, V, b)),
    (succ(Col, V) ; pred(Col, V)).

next(step(X)) :-
    true(step(Y)),
    X is Y + 1.

next(cell(X, Y, b)) :-
    does(player, move(X, Y)).

next(cell(U, Y, Z)) :-
    does(player, move(X, Y)),
    true(cell(U, Y, b)),
    true(cell(X, Y, Z)),
    Z \= b.

next(cell(X, V, Z)) :-
    does(player, move(X, Y)),
    true(cell(X, V, b)),
    true(cell(X, Y, Z)),
    Z \= b.
```

```
next(cell(U, V, Z)) :-
    true(cell(U, V, Z)),
    does(player, move(X, Y)),
    (X \= U ; Y \= V),
    true(cell(X1, Y1, b)),
    (X1 \= U ; Y1 \= V).

goal(player, 100) :-
    inorder.
goal(player, 0)   :-
    \+inorder.

terminal :- inorder.
terminal :- true(step(4)).

inorder :- true(cell(1, 1, 1)), true(cell(1, 2, 2)), true(cell(1, 3, 3)),
           true(cell(2, 1, 8)), true(cell(2, 2, b)), true(cell(2, 3, 4)),
           true(cell(3, 1, 7)), true(cell(3, 2, 6)), true(cell(3, 3, 5)).

succ(1, 2).
succ(2, 3).
pred(2, 1).
pred(3, 2).

%% Heuristic using Manhattan distance, also called taxicab geometry
heuristic(State, [goal(player, Value)]) :-
    maplist(taxicab_dist, State, Distances),
    sum_list(Distances, TotalDistances),
    Value is 100 - TotalDistances.

taxicab_dist(step(_), 0).
taxicab_dist(cell(Row, Col, Tile), Distance) :-
    member(cell(RowDest, ColDest, Tile),
        [cell(1, 1, 1), cell(1, 2, 2), cell(1, 3, 3),
         cell(2, 1, 8), cell(2, 2, b), cell(2, 3, 4),
         cell(3, 1, 7), cell(3, 2, 6), cell(3, 3, 5)]),
    abs(Row - RowDest, Y),
    abs(Col - ColDest, X),
    Distance is X + Y.
```

OUTPUT:

```
      time(solve_dfs(Path)).
      Path = [
       does(player,move(3,2)),
       does(player,move(3,1)),
       does(player,move(2,1)),
       does(player,move(2,2))
       ]
```

2 one:

```
     time(solve_bfs(Path)).
      Path = [
does(player,move(3,2)),
does(player,move(3,1)),
does(player,move(2,1)),
does(player,move(2,2))
]
```