QUICK SORT

```
#include <iostream.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
#define MAX 500
void qsort(int [],int,int);
int partition(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 Sorting:"<<"\t"<<"\n";
for(i=0;i<n;i++)
cout<<a[i]<<"\t";
e=clock();
z=e-s;
cout<<"\n"<<"Time taken:"<<z/ CLOCKS_PER_SEC<<"sec";
getch();
}
void qsort(int a[],int low,int high)
{
int j;
if(low<high)
{
j=partition(a,low,high);
qsort(a,low,j-1);
qsort(a,j+1,high);
}
}

int partition(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:5
Before Sorting:
40 30 82 90 56
After Sorting:
30 40 56 82 90
Time taken:2.362637
```

TOPOLOGICAL ORDERING OF VERTICES

```
#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+=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;
if(indegree[v]==0)
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 no.of nodes:"<<"\n";
cin>>n;
cout<<"Enter the adjacency matrix:"<<"\n";
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
cin>>a[i][j];
}
topology();
getch();
}
```

OUTPUT

```
Enter the no of nodes:5
Enter the adjacency matrix:
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
The topological sequence is:
2 1 3 4 5
```

TRAVELLING SALESMAN PROBLEM

```
#include <iostream.h>
#include <conio.h>
int a[10][10],completed[10],n,cost=0;
void takeinput()
{
clrscr();
int i,j;
cout<<"Enter the no of villagers:";
cin>>n;
cout<<"\nEnter cost matrix\n";
for(i=0;i<n;i++)
{
cout<<"Enter the elements of row:"<<i+1<<"\n";
for(j=0;j<n;j++)
cin>>a[i][j];
completed[i]=0;
}
cout<<"\nThe Cost List is:";
for(i=0;i<n;i++)
{
cout<<"\n";
for(j=0;j<n;j++)
cout<<"\t"<<a[i][j];
}
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i<n;i++)
{
if((a[c][i]!=0)&&(completed[i]==0))
if((a[c][i]+a[i][c]<min))
{
min=a[i][0]+a[c][i];
kmin=a[c][i];
nc=i;
}
}
if(min!=00)
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+=a[city][ncity];
return;
}
mincost(ncity);
}
int main()
{
takeinput();
cout<<"\nThe Path is:";
mincost(0);
cout<<"\nThe Minimum cost is:"<<cost;
getch();
return 0;
}
```

OUTPUT

```
Enter the no of villagers:2
Enter cost matrix
Enter the elements of row1:
1
2
Enter the elements of row2:
2
3
The cost List is:
1   2
2   3
The path is:1→2→1
The minimum cost is:5
```

PRIM'S 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<<"\nMinimum 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=u=i;
b=v=j;
}
}
}
if(visited[u]==0 || visited[v]==0)
{
cout<<"\nEdge"<<ne++<<"\t("<<a<<"--->"<<b<<")"<<min;
min_cost=min_cost+min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
cout<<"\nMinimum Cost="<<min_cost<<"\n";
getch();
}
```

OUTPUT

```
Enter the number of matrices:4
Enter the cost of matirx:
0 1 4 5
1 0 2 3
4 2 0 7
5 3 7 0
Enter the root node:2
Minimum cost Spanning Tree:
Edge(1→2)1
Edge(2→3)2
Edge(2→4)3
Minimum Cost=6
```

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 the no.of matirces:3
Enter the adjacency matrix:
0 5 12 17 999
999 0 999 8 7
999 999 0 9 999
999 999 999 0 999
999 999 999 999 0
Enter the source node:1
Shortest path from 1 to 2 is 5
Shortest path from 1 to 3 is 12
Shortest path from 1 to 4 is 13
Shortest path from 1 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 N QUEEN PROBLEM";
cout<<"\nEnter the no of queen:\n";
cin>>n;
queen(1,n);
getch(); }
void queen(int k,int n)
{
int j,i,m;
for(i=1;i<=n;i++)
{
if(place(k,i))
{
x[k]=i;
if(k==n)
{
cout<<"\n\t Feasible Solution:"<<count++;
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

```
Enter the no.of queen:4
Feasible Solution:1
   Row1–Column2:|.|Q|.|.|
   Row2–Column4:|.|.|.|Q|
   Row3–Column1:|Q|.|.|.|
   Row4–Column3:|.|.|Q|.|
```

4-QUEEN 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

```
Queen = [2, 4, 1, 3] .
```

TIC TAC TOE

```
win(Board, Player) :- rowwin(Board, Player).
win(Board, Player) :- colwin(Board, Player).
win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,_,_,_,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,Player,Player,Player,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,_,_,_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,_,_,Player,_,_,Player,_,_].
colwin(Board, Player) :- Board = [_,Player,_,_,Player,_,_,Player,_].
colwin(Board, Player) :- Board = [_,_,Player,_,_,Player,_,_,Player].

diagwin(Board, Player) :- Board = [Player,_,_,_,Player,_,_,_,Player].
diagwin(Board, Player) :- Board = [_,_,Player,_,Player,_,Player,_,_].
other(x,o).
other(o,x).

game(Board, Player) :- win(Board, Player), !, write([player, Player, wins]).
game(Board, Player) :-
  other(Player,Otherplayer),
  move(Board,Player,Newboard),
  !,
  display(Newboard),
  game(Newboard,Otherplayer).

move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).
move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).
move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).
move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).
move([A,B,C,D,b,F,G,H,I], Player, [A,B,C,D,Player,F,G,H,I]).
move([A,B,C,D,E,b,G,H,I], Player, [A,B,C,D,E,Player,G,H,I]).
move([A,B,C,D,E,F,b,H,I], Player, [A,B,C,D,E,F,Player,H,I]).
move([A,B,C,D,E,F,G,b,I], Player, [A,B,C,D,E,F,G,Player,I]).
move([A,B,C,D,E,F,G,H,b], Player, [A,B,C,D,E,F,G,H,Player]).

display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
 write([G,H,I]),nl,nl.
```

```
selfgame :- game([b,b,b,b,b,b,b,b,b],x).

x_can_win_in_one(Board) :- move(Board, x, Newboard), win(Newboard, x).

orespond(Board,Newboard) :-
  move(Board, o, Newboard),
  win(Newboard, o),
  !.
orespond(Board,Newboard) :-
  move(Board, o, Newboard),
  not(x_can_win_in_one(Newboard)).
orespond(Board,Newboard) :-
  move(Board, o, Newboard).
orespond(Board,Newboard) :-
  not(member(b,Board)),
  !,
  write('Cats game!'), nl,
  Newboard = Board.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).
xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).
xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).
xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).
xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).
xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).
xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).
xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).
xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).
xmove(Board, _, Board) :- write('Illegal move.'), nl.

playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).

explain :-
  write('You play X by entering integer positions followed by a period.'),
  nl,
  display([1,2,3,4,5,6,7,8,9]).

playfrom(Board) :- win(Board, x), write('You win!').
playfrom(Board) :- win(Board, o), write('I win!').
playfrom(Board) :- read(N),
  xmove(Board, N, Newboard),
  display(Newboard),
  orespond(Newboard, Newnewboard),
  display(Newnewboard),
  playfrom(Newnewboard).
```

OUTPUT

```
 playo.
You play X by entering integer positions followed by a period.
[1,2,3]
[4,5,6]
[7,8,9]

|: 5.
[b,b,b]
[b,x,b]
[b,b,b]

[o,b,b]
[b,x,b]
[b,b,b]

|: 2.
[o,x,b]
[b,x,b]
[b,b,b]

[o,x,b]
[b,x,b]
[b,o,b]
```

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))
]
```