3) Topological Sorting( can be applied only in Directed acyclic graphs)

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int n; /\*Number of vertices in the graph\*/

int adj[MAX][MAX]; /\*Adjacency Matrix\*/

void create\_graph();

int queue[MAX], front = -1,rear = -1;

void insert\_queue(int v);

int delete\_queue();

int isEmpty\_queue();

int indegree(int v);

int main()

{

int i,v,count,topo\_order[MAX],indeg[MAX];

create\_graph();

/\*Find the indegree of each vertex\*/

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

{

indeg[i] = indegree(i);

if( indeg[i] == 0 )

insert\_queue(i);

}

count = 0;

while( !isEmpty\_queue( ) && count < n )

{

v = delete\_queue();

topo\_order[++count] = v; /\*Add vertex v to topo\_order array\*/

/\*Delete all edges going fron vertex v \*/

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

{

if(adj[v][i] == 1)

{

adj[v][i] = 0;

indeg[i] = indeg[i]-1;

if(indeg[i] == 0)

insert\_queue(i);

}

}

}

if( count < n )

{

printf("\nNo topological ordering possible, graph contains cycle\n");

exit(1);

}

printf("\nVertices in topological order are :\n");

for(i=1; i<=count; i++)

printf( "%d ",topo\_order[i] );

printf("\n");

return 0;

}/\*End of main()\*/

void insert\_queue(int vertex)

{

if (rear == MAX-1)

printf("\nQueue Overflow\n");

else

{

if (front == -1) /\*If queue is initially empty \*/

front = 0;

rear = rear+1;

queue[rear] = vertex ;

}

}/\*End of insert\_queue()\*/

int isEmpty\_queue()

{

if(front == -1 || front > rear )

return 1;

else

return 0;

}/\*End of isEmpty\_queue()\*/

int delete\_queue()

{

int del\_item;

if (front == -1 || front > rear)

{

printf("\nQueue Underflow\n");

exit(1);

}

else

{

del\_item = queue[front];

front = front+1;

return del\_item;

}

}/\*End of delete\_queue() \*/

int indegree(int v)

{

int i,in\_deg = 0;

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

if(adj[i][v] == 1)

in\_deg++;

return in\_deg;

}/\*End of indegree() \*/

void create\_graph()

{

int i,max\_edges,origin,destin;

printf("\nEnter number of vertices : ");

scanf("%d",&n);

max\_edges = n\*(n-1);

for(i=1; i<=max\_edges; i++)

{

printf("\nEnter edge %d(-1 -1 to quit): ",i);

scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))

break;

if( origin >= n || destin >= n || origin<0 || destin<0)

{

printf("\nInvalid edge!\n");

i--;

}

else

adj[origin][destin] = 1;

}

}

Output:

