O/P:-

```
Enter number of vertices :4
Enter edge 1 (-1 -1 to quit) : 0 1
Enter edge 2 (-1 -1 to quit) : 0 3
Ender edge 3 (-1 -1 to quit) : 1 2
enter edge 4 (-1 -1 to quit) : 2 3
enter edge 5 (-1 -1 to quit) : -1 -1

Vertices in topological order are :
        0 1 2 3
```

Experiment - 6

* W.A.P to obtain the topological ordering of Vertices in a digraph.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int n;
int adj [MAX] [MAX]
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 ();
  for (i=0; i<n; i++)
   { indeg [i] = indegree (i);
       if (indeg [i] == 0)
           insert_queue (i);
   }
```

Scanned with CamScanner

```
count = 0;
while ( ! is empty_queue () && count <n)
 {
   v = delete_queue ( );
 topo_order [++count ] = 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 ("\n Vertices in topological order are : \n");
  for ( i= 1 ; i<= count ; i++){
   printf ("%d", topo_order [i] );    }
    printf ("\n");
  return 0;
  }
Void insert_queue (int Vertex )
 { if ( rear == MAX -1)
          printf ("\n Queue overflow");
```

```
else {
    if (front == -1)
        front = 0 ;
        rear = rear +1;
        queue [rear ] = vaten;  }}

int is empty_queue () {
 if ( front == -1 || front >rear )
      return 1;
   else
      return 0; }

int delete _queue () {
 int del-item ;
 if ( front ==-1 || front >rear)
 { printf ("\n Queue Underflow \n");
      exit(1);
  }
 else { del_item = queue [front ];
          front = front +1;
          return del_item ; }
 }
 int indegree (int v)
 { int i, in_deg =0;
     for (i=0 ;i<n ; i++)
```

Date

Expt. No.

```
    if ( adj [i][v] = = 1)
            in_deg++;
return in_deg ; }

Void Create _graph ()
{ int i, max_edges , origin , dest ;
 printf ("\n Enter number of Vertices:");
  Scanf ("%d", &n);
  max-edges  = n*n(n-1)
  for ( i = 1 , i< = max_edges ; i++ )
 {   printf ("\n Enter edge %d (-1, -1 to quit) :", i);
     Scanf ("%d %d ", &origin , &dest);
 if  ((origin = = -1) && (dest = = -1))
        break;
 if (( origin >= n || dest >= n || origin<0||
    ~~origin <~~  dest <0 )
  {
   printf ("\n Invalid edge !\n");
  }
  else
        adj [origin ][dest ] = 1;
  }
 }.
```