DESIGN & ANALYSIS OF ALGORITHM :

PROGRAM 1 : Design ,develop, and execute a program in c to sort a given set of elements using Quicksort method and determine the time required to sort the elements .Repeat the experiments for different values of n, the number of elements in the list to be sorted and plot a graph of time taken verses n.

ALGORITHM:

```
Algorithm Quicksort ( p, q )
//sorts the elements q [ p ] ,...,a [ q ] which reside in the global array a
[ 1 : n ] into ascending order; a [ n + 1 ] is considered to be defined and
must be ≥ all elements in a [ 1 : n ].
{
   if ( p < q ) then // if there are more than one element
   {
     // divide P into two subproblems
        j := Partition ( a, p, q+1 );
         // j is the position of the partitioning element
     // solve the subproblems.
        Quicksort( p, j-1 );
        Quicksort( j+1, q );
     // There is no need for combining solutions.
    }
}

Algorithm partition ( a, m, p )
// within a[ m ], a[ m + 1 ],....,a[ p - 1 ] the elements are rearranged in
such a manner that if initially t = a [ m ], then after completion a [ q ] =
t for some q between m
 and p - 1, a [ k ] ≤ t for m ≤ k < q, and a [ k ] ≥ t  for q < k < p, q is
returned.Set a[ q ] = infinity.
{
  v := a [ m ]; i := m; j := p;
  repeat
  {
       repeat
          i := i + 1;
       until ( a [ i ] ≥ v );

       repeat
           j := j - 1;
       until( a [ j ] ≥ v );


       if ( i < j ) then interchange ( a , i, j );

  } until( i ≥ j );

  a [ m ] := a [ j ]; a [ j ] := v; return j;
}


Algorithm interchange ( a, i, j )
// Exchange a [ i ] with a [ j ]
```

```
{
  p := a [ i ];
  a [ i ] := a [ j ]; a [ j ] := p;
}
```

CODE:

```
#include    < stdio.h >
#include    "quickSort1.h"
#include    < time.h >


int a[ 1001 ];

void interchange ( int a[ ], int i, int j )
{
      int t;
      t = a[ i ] ;
      a [ i ] = a[ j ] ;
      a [ j ]= t;
}

int partition ( int a[ ], int m, int p )
{
      int v = a[ m ];
      int i = m;
      int j = p;
      do
      {
while ( a[ ++ i ] < v );
while ( a[ -- j ]> v );
            if( i < j )
                  interchange ( a, i, j );
      }while( i < = j );
      a [ m ] = a [ j ]; a[ j ] = v;
      return j;
}

Void qSort(int p,int q )
{
      int j;
      if( p < q )
      {
            j = partition ( a, p, q +1);
            qSort( p, j -1);
            qSort( j +1, q );
      }
}

int main ( void )
{
      registerint i;
      int n, step =10;
      double duration;
      /* times for n = 0, 10, ..., 100, 200, ..., 1000 */
      Printf ( "\tn\trepetitions\ttime\n\n" );
      for( n = 0; n < = 1000; n + = step )
```

```
        {
              /*get time for size n */
              long repetitions = 0;
              clock_t start = clock ( );
              do{
                    repetitions ++;
                    /* initilize with worst-case data */
                    for( i = 0; i < n; ++ i )
                          a [ i ]= rand( )% 1000;//a[i] = n - i;
                    qSort(0, n - 1)    ;
                }while( clock( )- start < 1000);
                          /* repete until enough time has elapsed */
              Duration = ( ( double )( clock ( )- start ) ) / CLOCKS_PER_SEC;

              Duration / = repetitions;
              printf( "%9d %14d %15f\n\n", n, repetitions, duration );
              if( n = = 100) step =100;
        }
        getchar( );
        return0;
  }
```

PROGRAM 2 : Design, develop, and execute a program in c to sort a given set of elements using merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

ALGORITHM :

```
Algorithm MergeSort ( low, high )
// a [ low : high ] is a global array to be sorted.
// Small( p ) is true if there is only one element to sort.In this case the list is already sorted.
{
          if ( low < high ) then // If there are more than one element
          {
                  // Divide P into subproblems.
                         // Find where to split the set.
                                mid := [ ( low + high ) / 2 ] ;
                 // Solve the subproblems.
                         MergeSort ( low, mid ) ;
                         MergeSort ( mid + 1, high ) ;
                 //combine ( low, mid, high ) ;
                        Merge ( low, mid, high ) ;
           }
}

Algorithm Merge ( low, mid, high )
// a [ low : high ] is a global array containing two sorted subsets in a [ low : mid ] and in a [ mid + 1 : high ] .The goal is to merge these two sets into a single set residing in a [ low : high ] . b [ ] is an auxiliary global array.
{
            h := low ; i := low ; j := mid + 1 ;
            while ( h ≤ mid ) and ( j ≤ high ) ) do
           {
                   if ( a [ h ] ≤ a [ j ] )  then
                   {
                             b [ i ] := a [ h ] ; h := h + 1 ;
                   }
                    else
                  {
                           b [ i ] := a [ j ] ; j := j + 1 ;
                  }
                  i := i + 1 ;
         }
         if ( h > mid ) then
                 for k := j to high do
                {
                           b [ i ] := a [ k ] ; i := i + 1 ;
                }
         else
                for k := h to mid do
                {
                           b [ i ] := a [ k ] ; i := i + 1 ;
                 }
```

```
for k := low to high do a [ k ] : = b [ k ] ;
```

PROGRAM 3: Design , develop, and execute a program to

a) Obtain the Topological ordering of vertices in a given digraph

ALGORITHM :

```
ALGORITHM DFS ( u, n, a )
//purpose:To  obtain  the  sequence  of  jobs  to  be  executed  resulting  in
topological order
//Input:
//  u-From where the DFS  traversal start
//  n-The number of vertices in the graph
//  a-agjacency matrix of the given graph
//Global variables:
//  s-to know what are the nodes visited and what are the nodes that are not
visited
//  j-index variables to store vertices(only those nodes which are dead ends
or those nodes whose nodes are completly explored.
//  res-an array which holds the order in which the vertices are popped.
//Output:
//  res-indicates the vertices in the reverse order that are to be exicuted.

step 1 : [ visit the vertex u ]
         s [ u ] <- 1
step 2 : [ traverse deeper in to the graph till we get dead end or till all
vertices are visited ]
         for v <- 0 to n - 1 do
            if ( a [ u ] [ v ] ) = 1 and s [ v ] = 0 ) then
                  DFS ( v, n, a )
            end if
         end for
step 3 : [ store the dead vertex or which is completly explored ]
         j <- j + 1
           res [ j ] <- u;
step 4 : [ finished ]
          return

ALGORITHM DFS ( u, n, a )
//purpose:To  obtain  the  sequence  of  jobs  to  be  executed  resulting  in
topological order
//Input:
//  n-The number of vertices in the graph
//  a-agjacency matrix of the given graph
//Global variables:
//  s-to know what are the nodes visited and what are the nodes that are not
visited
//  j-index variables to store vertices(only those nodes which are dead ends
or those nodes whose nodes are completly explored.
//  res-an array which holds the order in which the vertices are popped.
//Output:
//  res-indicates the vertices in the reverse order that are to be exicuted.

Step 1: [Initialization to indicate that no vertex has been visited]
        For I <- 0 to n – 1 do
                  S[i] <- 0
            End for
            J <- 0; // An index to store the vertices which are dead ends and
which  are completely explored .
```

```
Step 2: [Process each vertex in the graph]
            For u <- 0 to n - 1 do
                  If ( s[u] = 0 ) call DFS ( u, n, a );
            End for
```

```
Step 3: [Output the topological sequence by printing in the reverse order of
popped sequence]
            For I <- n - 1 down to 0
                  Print res[i]
            End for
```

```
Step 4: [Finished]
      Return;
```

CODE :

```
#include    < stdio.h >
#define     N 7
Int adj[ ][ N +1 ] = {
      {0, 1, 2, 3, 4, 5, 6, 7},
      {1, 0, 1, 1, 0, 0, 0, 0},
      {2, 0, 0, 0, 0, 1, 0, 1},
      {3, 0, 0, 0, 0, 0, 1, 0},     //FIG :1.1
      {4, 1, 1, 1, 0, 0, 1, 1},
      {5, 0, 0, 0, 0, 0, 0, 0},
      {6, 0, 0, 0, 0, 0, 0, 0},
      {7, 0, 0, 0, 0, 1, 1, 0}
                  };

int visited [ N +1 ] = { 0 };
int count = 0;
int dVertex[ N +1 ];

void dfs( int v )
{
      int w;
      visited [ v ] = 1;
      for( w = 1; w < = N; ++ w )
      {
            if ( adj[ v ][ w ] && ! visited[ w ] )
            {
                  dfs( w );
            }
      }
      dVertex [ ++ count ] = v;
}

int main ( void )
{
      int i, j;
      int flag = 0;
      printf( "Adjacency matrix:\n\n" );
      for( i = 0; i < = N; ++ i )
```

```
        {
              for( j = 0; j < = N; ++ j )
                    printf( " %d\t ", adj[ i ][ j ]);
              printf ( "\n\n" );
        }
        for( i = 1; i < = N; ++ i )
              if( ! visited[ i ])
                    dfs( i );
        printf( "\n\nPopped Sequence:\n\n" );
        for( i = 1; i < = count ; ++ i )
              printf( " %d\t ", dVertex[ i ]);
        printf( " \n\n\nTopological Order:\n\n " ) ;
        for( i = count; i > =1 ; -- i )
        printf( " %d\t ", dVertex [ i ] );
        getchar( );
        return0;
  }
```

```
ALGORITHM topological_sort ( a, n, s )
//purpose  :  To  obtain  the  sequence  of  jobs  to  be  excecuted  resulting  in
topological order
//Input : a - adjacency marix of the given graph
//        n - the number of vertices in the graph
//Output : s - indicates the jobs that are to be executed in the order
           for j <- 0 to n - 1 do           //obtain indegree of each vertex
                    sum <- 0
                        for i <- 0 to n - 1 do
                                sum <- sum + a [ i ] [ j ]
                  end for
               indegree [ j ] <- sum
               end for

             top <- -1                   // place independent jobs of the stack
             for i <- 0 to n - 1 do
                    if ( indegree [ i ] = 0 )
                                  top <- top + 1
                                  s [ top ] <- i
                      end if
           end for

             while top ! = 1           //find the topological sequence
                   u <- s [ top ]      //delete the job from the stack
                     top <- top - 1
                     Add u to soution vector T
                              for each vector v adjacent to u
                                    decrement indgree[ v ] by one
                                    if ( indegree [ v ] = 0 )
                                            top <- top + 1
                                           s [ top ] <- v
                                  end if
                            end for
            end while

        write T                         //output the topological sequence
            return
```

CODE:

```c
#include    < stdio.h >

#define     N 7
int adj[ ][ N + 1 ] = {
      {0, 1, 2, 3, 4, 5, 6, 7},
      {1, 0, 1, 1, 0, 0, 0, 0},
      {2, 0, 0, 0, 0, 1, 0, 1},
      {3, 0, 0, 0, 0, 0, 1, 0},     //FIG :1.1
      {4, 1, 1, 1, 0, 0, 1, 1},
      {5, 0, 0, 0, 0, 0, 0, 0},
      {6, 0, 0, 0, 0, 0, 0, 0},
      {7, 0, 0, 0, 0, 1, 1, 0}
                          };

int indegre [ N +1 ] ;
int top = -1 ;
int s [ N +1 ] ;

void push ( int item )
{
        s [ ++ top ]= item;
}

int pop ( void )
{
      return s[ top -- ];
}

void findIndegre ( void )
{
      int i, j;
      int sum;
      for( i = 1; i < = N; ++ i )
      {
            Sum = 0;
            for( j = 1; j < = N;++ j )
                  sum + = adj [ j ][ i ];
            indegre [ i ]= sum;
      }
}

void topoSort( void )
{
      int u, v, t[ 10 ];
      int i, k;

      for( i = 1; i < = N; ++ i )
            if ( ! indegre[ i ] )
                  push( i );

      k =0;
      while( top ! = -1)
      {
            u = pop ( );
            t[ ++ k ]= u;
            for( v = 1; v < = N; ++ v )
            {
                  if( adj[ u ][ v ]= = 1)
                  {
                        indegre[ v ]--;
                        if( ! indegre[ v ] )
                              push( v );
                  }
            }
      }
      printf ( "\n\nThe topological sequence is:\n\n" );
      for( i = 1; i < = N;++ i )
            printf( " %d\t ", t[ i ] );

      printf( "\n\n" );
}

int main ( void )
```

```
{
      int i, j;
      printf( " Adjacency matrix:\n\n " );
      for( i = 0; i < = N; ++ i )
      {
            for( j = 0; j < = N ; ++ j )
                  printf( " %d\t ", adj[ i ][ j ]);
            printf( "\n\n" );
      }
      findIndegre( );

      topoSort( );
      getchar( );
      return 0;
}
```

b) Compute the transitive closure of a given directed graph using Warshall's algorithm.

ALGORITHM:

```
ALGORITHM Warshall ( A [ 1...n,  1....n ] )

// Implement Warshall's algorithm for computing the transitive closure
// Input : The agjacency matrix A of a digraph with n vertices
// Output : The transitive closure of the digraph
R ( 0 ) <- A
for k <- 1 to n do
    for i <- 1 to n do
        for j <- 1 to n do
           R ( k ) [ i, j ] <- R ( k - i ) [ i, j ] or R ( k - i ) [ i, k ]
and R ( k - 1 )[ k, j ]
return R ( n )
```

CODE :

```
#include < stdio.h >
#define N 6
int a[ ][ N ] = {
      {0, 1, 2, 3, 4, 5},
      {1, 0, 1, 1, 0, 0},
      {2, 0, 0, 0, 1, 0}, // vertices = 6    FIG : 1.2
      {3, 0, 0, 0, 1, 1},
      {4, 0, 0, 0, 0, 1},
      {5, 0, 1, 0, 0, 0}
};
int p [ N ][ N ];

void warshall( void )
{
      int i, j, k ;
      for( i = 1; i < = N - 1; ++ i )
            for( j = 1; j < = N - 1 ; ++ j )
                  p [ i ] [ j ] = a [ i ] [ j ];
      for( k = 1; k < = N - 1; ++ k )
            for( i = 1; i < = N - 1; ++ i )
                  for( j = 1; j < = N - 1; ++ j )
                        if( ( p [ i ][ k ]= = 1 ) && ( p[ k ][ j ]= = 1) )
                              p[ i ][ j ] = 1;
}

int main ( void )
{
      int i, j;
      printf( "\nAdjacency matrix:\n\n" );
      for( i = 1; i < =  N - 1; ++ i )
      {
            for( j = 1; j < = N - 1; ++ j )
                  printf( "%d " , a[ i ][ j ]);
            printf( "\n\n" );
      }
```

```
        warshall( );
        printf( "\n\nTransitive Closure Path Matrix:\n\n" );
        for( i = 1; i < =  N - 1; ++ i )
        {
              for( j = 1; j < = N - 1; ++ j )
                    printf( "%d ", p[ i ][ j ] );
              printf( "\n\n" );
        }
        getch( );
        return 0;
}
```

PROGRAM 4 : Design, develop and execute a program in c to Implement 0/1 Knapsack problem using Dynamic programming.

ALGORITHM :

```
ALGORITHM DKnap ( p, w, x, n, m )
{
      // pair [ ]   is an array of PW's
      b [ 0 ] := 1 ; pair [ 1 ].p := pair [ 1 ].w := 0.0 ;  // s0
      t := 1 ; h := 1 ; // start and end of s0
      b [ 1 ] := next := 2 ; // next free spot in pair [ ]
      for i := 1 to n - 1  do
      {  // generate si.
         k := t ;
         u := Largest ( pair, w, t, h, i, m ) ;
         for j := t to u do
         {  /Type equation here./ generate si - 1 and merge
              pp := pair [ j ].p + p[ i ] ; ww := pair [ j ]. w + w [ i ] ;
                         // ( pp, ww ) is the next element is si - 1i
                while ( ( k ≤ h ) and ( pair [ k ] . w ≤ ww ) ) do
            {
                         pair [ next ] . p := pair [ k ] . p ;
                         pair [ next ] . w := pair [ k ] . w ;
                         next := next + 1 ; k := k + 1 ;
                  }
                  if ( ( k ≤ h ) and ( pair [ k ] . w = ww ) )  then
                  {
                 if pp < pair [ k ] . p then pp := pair [ k ] . p ;
                 k := k + 1 ;
                  }
                  if pp > pair [ next - 1 ] . p then
            {
                          pair [ next ] . p := pp ; pair [ next ] . w := ww ;
                          next := next + 1 ;
                  }
            while ( ( k ≤ h ) and ( pair [ k ] . p ≤ pair [ next - 1 ] . p )
)  do
                k := k + 1 ;
         }
       // Merge in remaining terms from si - 1.
       while ( k ≤ h ) do
       {
                pair [ next ] . p := pair [ k ] . p ; pair [ next ] . w :=
pair [ k ]. w ;
                  next := next + 1 ; k := k + 1 ;
         }
         // Initialize for si + 1
         t := h + 1 ; h := next - 1 ; b [ i + 1 ] := next ;
   }
   TraceBack( p, w, pair, x, m, n ) ;
}
```

CODE:

```
#include    < stdio.h >

typedef struct profitWeightPair
{
      int p;
      int w;
}PW;
PW pair[ 100 ];
int b[ 50 ];
int x[ ] = { };

int comparePairs( PW x, PW y )
{
if( x.p = = y.p) return 0;
if( x.p < y.p ) return-1;
else return 1;
}

Int binSearch ( int low, int high, PW key )
{
int mid;
while( low < = high )
      {
            mid =( low + high ) / 2;
            switch( comparePairs( pair[ mid ], key ) )

            {
                  Case -1 : low = mid + 1; break;
                  Case 0 : return1;
                  Case 1 : high = mid -1;
            }
      }
return 0;
}

int compare ( int x, int y )
{
if( x = = y ) return 0;
if( x < y )return -1;
else return 1;
}

Void traceBack ( int p[ ],int w[ ],int m, int n )
{
      int i, j, set;
      int count = 0;
      int start, end;

      PW key = pair [ b [ n + 1] - 1];

      for( i = n + 1; i > =1; -- i )
      {
for( j = i; j > =1; -- j )
{
```

```
start = b[j - 1];
end =  b[ j ] - 1;

if( binSearch ( start, end, key ) )
{
                              Set = j -1;
                              Count++;
}
}
if( count > 0 ) x [ set ] = 1;

key.p - = p[ set ];
key.w - = w [ set ];

set = count = 0;
}
printf( "\n\n" );
printf( "Selected items:\n\n" );
for( i = 1; i < = n; ++ i )
printf( "x[%d]\t",  i );
printf( "\n\n" );
      for( i = 1; i < = n; ++ i )
          printf( "%d\t", x[ i ]);

}

int largest ( int w[ ], int t, int h, int i, int m )
{
int q;
while( t < = h )
      {
            q = ( t +  h )/2;
            switch( compare ( pair[ q ].w + w[ i ], m ))
            {
                  case-1 : t = q + 1; break ;
                  case 0 : return q ;
                  case 1 : h = q -1;
            }
      }
      return t -1;
}

void printPairs( int t, int h )
{
      int i;
      for( i = t; i < = h;++ i )
            printf("( %d, %d ) ", pair[ i ].p, pair[ i ].w );
      printf( "\n" );
}

Void DKnap( int p[ ], int w[ ], int n, int m )
{
int i, j, t, h, u, k, next;
int pp,ww ;

      //pair[] is an array of PW's.
      b [ 0 ] = 1;
```

```
        pair [ 1 ].p = pair[ 1 ].w = 0;
        t = h = 1;  // Start and end of S0
        b[ 1 ] = next = 2 ;  //Next free spot in pair[]
        for( i = 1; i < = n; ++ i )
        {
              k = t;
              u = largest ( w, t, h, i, m );
              for( j = t; j < = u;++ j )
              {   // Generate S1i-1 and merge
                    pp = pair[ j ].p + p[ i ];
                    ww = pair[ j ].w + w[ i ];
                    // ( pp, ww ) is the next element in s1i-1
                    while( ( k < = h ) && ( pair[ k ].w < = ww) )
                    {
                          pair[ next ].p = pair[ k ].p;
                          pair[ next ].w = pair[ k ].w;
                          next += 1 ;
                          k += 1;
                    }
                    if(( k < = h ) && ( pair[ k ].w = = ww) )
                    {
                          if( pp < pair[ k ].p )
                                pp = pair[ k ].p;
                          k + = 1;
                    }
                    if( pp > pair[ next -1 ].p )
                    {
                          pair[ next ].p = pp;
                          pair[ next ].w = ww;
                          next + = 1;
                    }
                    while(( k < = h ) && ( pair[ k ].p < = pair[ next -1 ].p ))
                    {
                          k + = 1;
                    }
              }
              //Merge in remaining terms from Si-1
              while( k < = h )
              {
                    pair[ next ].p = pair[ k ].p;
                    pair[ next ].w = pair[ k ].w;
                    next + = 1;
                    k + = 1;
              }
              //Initilize for Si+1
              t = h + 1;
              h = next -1;
              b[ i +1 ] = next;
              //printf( "\n\n" );
              //printPairs( t,h );
        }
        traceBack( p, w, m, n );
        printf( "\n\nOptimal solution is: ( %d )", pair[ h ].p );

  }

  int main ( void )
```

```
{
      int i;

            int p[ ] = { -1, 1, 2, 5 };
            int w[ ] = { -1, 2, 3, 4 };
            int x[ ] = { };
            int n = 3;
            int m = 6;
      /*
            int p[ ] = { -1, 20, 30, 66, 40, 60 };
            int w[ ] = { -1, 10, 20, 30, 40, 50 };
            int x[ ] = { };
            int n = 5;
            int m = 100;
      */
      /*intp[ ] = { -1, 10, 40, 30, 50 };
            int w[ ] = { -1, 5, 4, 6, 3 };
            int x[ ] = { } ;
            int n = 4;
            int m = 10;
      */
      /*    intp[ ] = { -1, 3, 4, 5, 8, 10 };
            int w[ ] = { -1, 2, 3, 4, 5, 9 };
            int x[ ] = { };
            int n = 5;
            int m = 20;
      */
/*    intp[ ] = { -1, 15, 10, 9, 5 };
            int w[ ] = { -1, 1, 5, 3, 4 };
            int x[ ] = {};
            int n = 4;
            int m = 8;
      */
      /*
            int p[ ] = { -1, 12, 10, 20, 15 };
            int w[ ] = { -1, 2, 1, 3, 2 };
            int n = 4;
            int m = 5;
      */

      printf( "\nn: %d\tm: %d\n" , n, m );
      printf( "\nProfits: " );
      for( i = 1; i < = n; ++ i ) printf( "%3d" , p[ i ] );
      printf(" \n\nWeights: ");
      for( i = 1; i < = n ; ++ i ) printf ( "%3d",  w[ i ] );
            DKnap( p, w, n, m );

            fflush( stdin );
            getchar( );

            return 0;

}
```

```
}
```

CODE :

```
#include    < stdio.h >
#include    " merge.h "
#include    < time.h >

int a[ 1001 ];

void merge ( int low, int mid, int high )
{
int b[ 1001 ];
      int h = low;
      int i = low;
      int j = mid + 1;
      int k;

      while( ( h < = mid ) && ( j < = high ) )
      {
            if( a [ h ]< = a[ j ] ) b[ i ++] = a[ h ++ ];
            else b[ i ++ ] = a [j ++ ];
      }
      if( h > mid )
      {
            for( k = j; k < = high; ++ k ) b [ i ++ ] = a[ k ];
      }
      else
      {
            for( k = h; k < = mid; ++ k ) b[ i ++ ]= a [ k ] ;
      }
      for( k = low; k < = high; ++ k ) a[ k ] = b[ k ];
}

Void mergeSort( int low, int high )
{
      int mid;

      if( low < high )
      {
            mid =( low + high ) / 2;

            mergeSort ( low, mid );
            mergeSort ( mid + 1, high );
            merge ( low, mid, high );
      }
}


int main ( void )
{
      registerint i;
      int n, step = 10;
      double duration;

      /* times for n = 0, 10, ..., 100, 200, ..., 1000 */
```

```
        printf( "\tn\trepetitions\ttime\n\n" );
        for( n = 0; n < = 1000; n + = step )
        {
              /*get time for size n */
              long repetitions = 0;
              clock_t start = clock ( );
              do{
                    repetitions ++ ;
                    /* initilize with worst-case data */
                    for( i = 0; i < n; ++ i )
                          a[ i ]= n - i;

                    mergeSort ( 0, n -1 );
              }while( clock( )- start < 1000 );
                          /* repete until enough time has elapsed */
              duration = (( double )( clock ( )- start ) ) / CLOCKS_PER_SEC;
              duration / = repetitions ;
              printf("%9d %14d %15f\n\n", n, repetitions, duration );
              if( n = =100) step =100;
        }

        getchar ( );
        return0;
  }
```

ALGORITHM : LKnapsack

the following recurrence:

$$F(i, j) = \begin{cases} \max\{F(i-1, j), v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0, \\ F(i-1, j) & \text{if } j - w_i < 0. \end{cases}$$

It is convenient to define the initial conditions as follows:

$F(0, j) = 0$ for $j \geq 0$ and $F(i, 0) = 0$ for $i \geq 0$.

CODE :

```
#include    < stdio.h >
//#include< conio.h >
int v [ 10 ][ 10 ], w[ 10 ], p[ 10 ];
int max (int a,int b )
{
return ( a > b ? a : b );
}

Void traceBack ( int x[ ],int n ,int c )
{
      int i, j;
      for( i = 0; i < = n ; ++ i )
      x [ i ] = 0;
    i = n;
    j = c;
while( i ! = 0 &&  j ! = 0)
{
if( v[ i ][ j ]! = v [ i - 1 ][ j ] )
{
x[ i ] = 1;
            j = j - w [ i ];
}
        i = i - 1;
}
printf( "\n\nSelected Items:\n\n" );
for( i = 1; i < = n; ++ i )
      printf( "x[%d]\t" , i );
printf( "\n\n" );
for( i = 1; i < = n; ++ i )
      printf( " %d\t ", x[ i ] );
}

int knap ( int n, int c )
{
int i, j;
for( i = 0; i < = n; ++ i )
{
      for( j = 0; j < = c ; ++ j )
{
if( i = = 0 || j = = 0)
v [ i ][ j ] = 0;
elseif ( j < w[ i ])
v[ i ][ j ]= v[ i - 1][ j ];
else
v [ I ][ j ] = max ( v[ i -1 ][ j ], p[ i ] + v[ i -1 ][ j - w[ i ] ] );
}
}
printf( "\n\nThe Matrix is:\n" );
for( i = 0; i < = n ; ++ i )
{
printf( "\n\n" );
for( j = 0; j < = c; ++ j)
      printf( "%d\t", v[ i ][ j ] );
}
return v[ n ][ c ];
}

int main ( void )
{
int i, j, c, n, opt, x[ 10 ];
printf( "\nEnter the Number of Items: ");
scanf( " %d ", &n );
printf( "\nEnter the Weights of %d Items:\n", n );
for( i = 1; i < = n; ++ i )
      scanf( "%d ", & w[ i ] );
printf( "\nEnter the Profits of %d Items:\n", n );
for( i = 1; i < = n; ++ i )
      scanf( "%d ",& p[ i ] );
printf( " \nEnter the Capacity of Knapsack: ?\b" );
      scanf( " %d ", &c );
opt = knap ( n, c );
printf( "\n\n\nOptimal Solution: %d\n" , opt );
      traceBack( x, n , c );
      printf( "\n\n" );
return 0;
}
```

PROGRAM 5: Design,develop and execute a program in c to Find Minimum Cost spannning Tree of a given undirected graph using prim's algorithm.

ALGORITHM:

```
Algorithm Prim (  E, cost, n, t )
//E is the set of edges in G. Cos [ 1:n, 1:n ] is the cost adjancey matrix of
an n vertex graph such that cost [ i, j ] is either a positive real number or
infinity if no edge( i, j ) exists .A minimum spanning tree is computed and
stored as a set of edges in the array t[ 1:n-1, 1:2 ].( t[ i, 1 ],t[ i, 2 ])
is an edge in the minimun-cost spanning tree .The final cost is returned.
{
      Let ( k, l ) be an edge of minimum cost in E;
      min cost := cost[ k, l ];
      t[ 1, 1 ] := k; t[ 1, 2 ] := l;
      for i := 1 to n do   //Initialize near.
      if ( cost [ i, l ] < cost [ i, k ] ) then near [ i ] := l;
                     else near [ i ] := k;
     near [ k ] := near [ l ] := 0;
     for i := 2 to n-1 do
     {   // Find n-2 additional edges for t.
             Let j be an index such that near [ j ] != 0 and
            cost [ j, near [ j ] ] is minimum;
            t [ i, 1 ] := j; t [ i, 2 ] := near[j];
            mincost :=  mincost + cost [ j, near [ j ] ];
            near [ j ] := 0;
            for k := 1 to do  //Update near[].
            if ( ( near [ k ] != 0 ) and ( cost [ k, near [ ] ] > cost [ k, j
] ) )
            then near [ k ] := j;
     }
     return mincost;
}
```

CODE:

```
#include    < stdio.h >
/*
#define     N 7
int cost[ ][ N + 1 ] = {
      { 0,  1,  2,  3,  4,  5,  6,  7 },
      { 1,  0, 28,  ∞,  ∞,  ∞, 10,  ∞ },
      { 2, 28,  0, 16,  ∞,  ∞,  ∞, 14 },
      { 3,  ∞, 16,  0, 12,  ∞,  ∞,  ∞ },  FIG :1.3
      { 4,  ∞,  ∞, 12,  0, 22,  ∞, 18 },
      { 5,  ∞,  ∞,  ∞, 22,  0, 25, 24 },
      { 6, 10,  ∞,  ∞,  ∞, 25,  ∞,  ∞ },
      { 7,  ∞, 14,  ∞, 18, 24,  ∞,  ∞ }
                        };
*/
/*
#define     N 8
int cost[][N + 1] = {
      { 0,  1,  2,  3,  4,  5,  6,  7,  8 },
      { 1,  0, 11, 13,  ∞,  2,  ∞,  ∞,  ∞ },
      { 2, 11,  0, 15,  8, 12,  ∞,  6,  ∞ },     FIG : 1.4
      { 3, 13, 15,  0,  ∞,  ∞,  ∞,  ∞,  ∞ },
      { 4,  ∞,  8,  ∞,  0, 14,  ∞, 10, 17 },
      { 5,  2, 12,  ∞, 14,  0,  ∞,  ∞,  8 },
      { 6,  ∞,  ∞,  ∞,  ∞,  ∞,  0, 21,  7 },
      { 7,  ∞,  6,  ∞, 10,  ∞, 21,  0, 11 },
      { 8,  ∞,  ∞,  ∞, 17,  5,  7, 11,  0 }
                        };
*/

#define     N 12
int cost[ ][N +1]={
{ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12},
{ 1,  0,  3,  5,  4,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞},
{ 2,  3,  0,  ∞,  ∞,  3,  6,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞},
{ 3,  5,  ∞,  0,  2,  ∞,  ∞,  4,  ∞,  ∞,  ∞,  ∞,  ∞},
{ 4,  4,  ∞,  2,  0,  1,  ∞,  ∞,  5,  ∞,  ∞,  ∞,  ∞},  FIG:1.5
{ 5,  ∞,  3,  ∞,  1,  0,  2,  ∞,  ∞,  4,  ∞,  ∞,  ∞},
{ 6,  ∞,  6,  ∞,  ∞,  2,  0,  ∞,  ∞,  ∞,  5,  ∞,  ∞},
{ 7,  ∞,  ∞,  4,  ∞,  ∞,  ∞,  0,  3,  ∞,  ∞,  6,  ∞},
{ 8,  ∞,  ∞,  ∞,  5,  ∞,  ∞,  3,  0,  6,  ∞,  7,  ∞},
{ 9,  ∞,  ∞,  ∞,  ∞,  4,  ∞,  ∞,  6,  0,  3,  ∞,  5},
{10,  ∞,  ∞,  ∞,  ∞,  ∞,  5,  ∞,  ∞,  3,  0,  ∞,  9},
{11,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  6,  7,  ∞,  ∞,  0,  8},
{12,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  5,  9,  8,  0}
                        };

int t[ N + 1 ][ N + 1 ];

int find_j( int near[ ] )
{
            int j, u, min = ∞;
            for( j = 1; j < = N; ++ j )
            {
                  if( near[ j ]! = 0)
                  {
                        if( cost[ j ][ near[ j ] ] < min )
                        {
                              Min = cost[ j ][ near[ j ] ];
                              u = j;
                        }
                  }
            }
      return u;
}

int primes ( void )
{
      int i, j, k, min;
      int minCost, near[ N + 1 ];

      minCost = 0;
      for( i = 2; i < = N; ++ i )
            near[ i ] = 1;
      t[ 1 ][ 1 ]= 1;
      near[ 1 ] = 0;
      for( i = 1; i < = N - 1; ++ i )
      {
            j = find_j( near );

            t[ i ][ 1 ] = j;
            t[ i ][ 2 ] = near[ j ];
            minCost + = cost[ j ][ near[ j ] ];

            near[ j ] = 0;

            for( k = 1; k < = N; ++ k )
            {
                  if( ( near[ k ]! = 0 )&&( cost[ k ][ near[ k ] ] > cost[ k
][ j ] ) )
                  {
                        near[ k ] = j;
                  }
            }
      }
      Return minCost;
}

int main ( void )
{
      int i, j;
      printf( "\n\n\n" );
      printf( " Adjacency matrix:\n\n " );
      for( i = 0; i < = N ; ++ i )
      {
            for( j = 0; j < = N; ++ j ) printf( "%d\t ", cost[ i ][ j ]);
            printf( "\n\n" );
      }
      printf( " \t\t\tMinimum Cost = %d\n ",  primes ( ) );
      printf( "\n\t\t\tMinimum Spanning Tree\n" );
      for( i = 1; i < N; ++ i )
            printf( "\n\t\t\t%6d - %d\n", t[ i ][ 1 ], t[ i ][ 2 ] );
      printf( "\n\n\n" );

      return 0;
}
```

PROGRAM 6 : Design ,Develop and execute a program in c to find Minimum cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

ALGORITHM:

```
Algorithm Kruskal ( E, cost, n, t )
//E is the set of edges is G . G has n vertices. cost [ u, v ] is the cost of
edge ( u, v ).t is the set
// of edges in the minimum-cost spanning tree . The final cost is returned.
{
    Construct a heap out of the edge costs using Heapify ;
                   for i := 1 to n do parent [ i ] := -1 ;
                   //Each vertex is in a different set.
      i := 0 ; mincost := 0.0 ;
      while ( ( i < n - 1 ) and ( heap not empty ) ) do
      {
           Delete a minimum cost edge( u, v ) from the heap and reheapify
using Adjust;
             j := Find ( u ) ; k := Find ( v ) ;
             if ( j ! = k ) then
             {
                                i := i + 1 ;
                  t [ i, 1 ] := u ; t [ i, 2 ] := v ;
                  mincost :=  mincost + cost [ u, v ] ;
                  Union ( j, k ) ;
              }
      }
      if ( i ! = n - 1 ) then write ( "NO spanning tree" ) ;
      else return mincost ;
}
```

CODE:

```
#include    < stdio.h >

typedef struct set_of_edges
{
      int k;
      int l;
      int cost;
}EDGE;

#define     N 7
#define NO_OF_EDGES 9
int cost[ ][N + 1 ] = {
      { 0,  1,  2,  3,  4,  5,  6,  7 },
      { 1,  0, 28,  ∞,  ∞,  ∞, 10,  ∞ },
      { 2, 28,  0, 16,  ∞,  ∞,  ∞, 14 },
      { 3,  ∞, 16,  0, 12,  ∞,  ∞,  ∞ },    FIG :1.6
      { 4,  ∞,  ∞, 12,  0, 22,  ∞, 18 },
      { 5,  ∞,  ∞,  ∞, 22,  0, 25, 24 },
      { 6, 10,  ∞,  ∞,  ∞, 25,  ∞,  ∞ },
      { 7,  ∞, 14,  ∞, 18, 24,  ∞,  ∞ }
                          };
```

```
EDGE heap[ ] = {
      { 0,  0,  0},
      { 6,  1, 10}, { 2,  1, 28 }, { 3, 2, 16 },
      { 7,  2, 14}, { 4,  3, 12 }, { 5, 4, 22 },
      { 6,  5, 25}, { 7,  5, 24 }, { 7, 4, 18 }
};

/*
#define     N 12
#define NO_OF_EDGES 19
int cost[][N + 1] =   {
{ 0 , 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12},
{ 1 , 0,  3,  5,  4,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞},
{ 2 , 3,  0,  ∞,  ∞,  3,  6,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞},
{ 3 , 5,  ∞,  0,  2,  ∞,  ∞,  4,  ∞,  ∞,  ∞,  ∞,  ∞},
{ 4 , 4,  ∞,  2,  0,  1,  ∞,  ∞,  5,  ∞,  ∞,  ∞,  ∞},
{ 5 , ∞,  3,  ∞,  1,  0,  2,  ∞,  ∞,  4,  ∞,  ∞,  ∞},   FIG : 1.7
{ 6 , ∞,  6,  ∞,  ∞,  2,  0,  ∞,  ∞,  ∞,  5,  ∞,  ∞},
{ 7 , ∞,  ∞,  4,  ∞,  ∞,  ∞,  0,  3,  ∞,  ∞,  6,  ∞},
{ 8 , ∞,  ∞,  ∞,  5,  ∞,  ∞,  3,  0,  6,  ∞,  7,  ∞},
{ 9 , ∞,  ∞,  ∞,  ∞,  4,  ∞,  ∞,  6,  0,  3,  ∞,  5},
{10 , ∞,  ∞,  ∞,  ∞,  ∞,  5,  ∞,  ∞,  3,  0,  ∞,  9},
{11 , ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  6,  7,  ∞,  ∞,  0,  8},
{12 , ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  ∞,  5,  9,  8,  0}
                        };
EDGE heap[] =   {
      { 0, 0, 0 },
      { 2, 1, 3 }, { 3, 1, 5 }, { 4, 1, 4}, { 6, 2, 6},
      { 5, 2, 3 }, { 4, 3, 2 }, { 7, 3, 4}, { 5, 4, 1},
      { 8, 4, 5 }, { 6, 5, 2 }, { 9, 5, 4}, {10, 6, 5},
      { 8, 7, 3 }, { 11,7, 6 }, { 9, 8, 6}, {11, 8, 7},
      {10, 9, 3 }, { 12,9, 5 }, {12, 11,8}
};
*/
/*
#define     N 8
#define NO_OF_EDGES 14
int cost[][N + 1] = {
      { 0,   1,  2,  3,  4,  5,  6,  7,  8 },
      { 1,   0, 11, 13,  ∞,  2,  ∞,  ∞,  ∞ },
      { 2,  11,  0, 15,  8, 12,  ∞,  6,  ∞ },
      { 3,  13, 15,  0,  ∞,  ∞,  ∞,  ∞,  ∞ },  FIG : 1.8
      { 4,   ∞,  8,  ∞,  0, 14,  ∞, 10, 17 },
      { 5,   2, 12,  ∞, 14,  0,  ∞,  ∞,  5 },
      { 6,   ∞,  ∞,  ∞,  ∞,  ∞,  0, 21,  7 },
      { 7,   ∞,  6,  ∞, 10,  ∞, 21,  0, 11 },
      { 8,   ∞,  ∞,  ∞, 17,  5,  7, 11,  0 }
                        };
EDGE heap[] =   {
      {0, 0, 0 },
      {1, 2, 11}, {1, 3, 13}, {1, 5, 2}, {2, 3, 15},
      {2, 5, 12}, {2, 4,  8}, {2, 7, 6}, {4, 5, 14},
      {4, 8, 17}, {4, 7, 10}, {5, 8, 5}, {6, 7, 21},
      {6, 8,  7}, {7, 8, 11}
};
*/
```

```
/*
#define     N 6
#define NO_OF_EDGES 4
int cost[][N + 1] = {
      {0,  1,  2,  3,  4, 5, 6 },
      {1,  0, 10,  ∞,  ∞, ∞, ∞ },
      {2, 10,  0, 20,  ∞, ∞, ∞ },
      {3,  ∞, 20,  0, 30, ∞, ∞ },
      {4,  ∞, ∞, 30,  0,  ∞,  ∞ },
      {5,  ∞, ∞,  ∞,  ∞,  0, 40 },
      {6,  ∞, ∞,  ∞,  ∞, 40,  0 }
                        };
EDGE heap[] =   {
      {0, 0, 0},  {1, 2, 10}, {2, 3, 20}, {3, 4, 30}, {4, 6, 40}
};
*/
int t[ N + 1 ][ N + 1 ];
int parent [ N + 1 ];
int flag = 0;

void adjust (int i, int n )//O(log(n))
{
      int j = 2 * i;
      EDGE item = heap[ i ];
      while( j < = n )
      {
            if( ( j < n )&&( heap[ j ].cost > heap[ j +1 ].cost ) )
                  ++j ;
            if( item .cost < = heap[ j ].cost ) break;
            heap[j / 2] = heap[ j ];
            j = 2 * j;
      }
      heap[j / 2 ] = item;
}

void heapify( int n )
{
      int i;
      for( i = n /2; i > = 1; -- i )
            adjust( i, n );
}

EDGE delMin( int n )
{
      EDGE x;

      if( n = =0) printf( "\n\t\t\tHeap is empty\n" );
      else
      {
            x = heap[ 1 ]; heap[ 1 ] = heap[ n ];
            adjust( 1, n - 1);
            return  x;
      }
}

Void simpleUnion( int i, int j )
{
      parent[ i ] = j;
}

int find ( int i )
{
      while( parent[ i ]> = 0) i = parent[ i ];
      return i;
}

int kruskal( void )
{
      EDGE e;
      int i, x;
      int u, v;
      int j, k;
      int minCost;
      heapify( NO_OF_EDGES );// Construct a heap out of the edge costs using
Heapify; O(|E|)
      for( i = 1; i < = N ; ++ i )// O(n)
            parent[i]=-1;
      i =0; x = NO_OF_EDGES +1; minCost = 0;
      while(( i < N - 1) && ( x ! = 0)) // O(|E|log(|E|))
      {
            e = delMin ( -- x );
            u = e.k ; v = e.l;
            j = find ( u );
            k = find ( v );
            if( j ! = k )
            {
                  i + = 1;
                  t[ i ][ 1 ] = u;
                  t[ i ][ 2 ] = v;
                  minCost + = cost[ u ][ v ];
                  simpleUnion( j, k );
            }
      }
      if( i ! = N -1){
            printf( "\n\n\t\t\t\"No spanning tree\"\n\n" );
            flag = 1;
      }
      else
            return minCost;
}

int main ( void )
{

      int i, j;
      printf( "\n\n\n" );
      printf( "Adjacency matrix:\n\n" );
      for( i = 0; i < = N; ++ i )
      {
            for( j = 0; j < = N ; ++ j )
                  printf( "%d\t" , cost[ i ][ j ] );
            printf( "\n\n" );
      }
      printf( "\n\t\t\tMinimum Cost = %d\n", kruskal( ) );
      if( ! flag )
      {
            printf( "\n\n\t\t\tMinimum Spanning Tree\n\n" );
            for( i = 1; i < N ; ++ i )
                  printf( "\n\t\t\t%6d - %d\n", t[ i ][ 1 ], t[ i ][ 2 ] );
      }
      printf( "\n\n\n" );
      return 0;
}
```

PROGRAM 7 : Design ,develop and execute a program in c to a) Print all nodes reachable from a given starting node in a digraph using BFS method.

ALGORITHM:

```
 Algorithm BFS ( v )
// A breadth first search of G is carried out beginning at a vertex v . For
any node i, visited [ i ] = 1
//if i has already beenn visited . The graph G and array visited [ ] are
global; visited [ ] is initialized to zero.
{

          u := v ;  // q is a queue of unexplored vertices.
           visited [ v ] := 1 ;
           repeat
            {
                   for all vertices w adjacent from u do
                   {
               if ( visited [ w ] = 0 ) then
                {
                          Add w to q; //w is unexplored.
                          visited [ w ] := 1 ;
                 }
       }
        if q is empty the return; // No unexplored vertex.
              Delete the next element, u, from q ; //get first unexplored
vertex.
      } until ( false ) ;
}
```

CODE:

```
#include    < stdio.h >
#include    < stdlib.h >
/*
#define     N 10
int adj [][ N + 1 ] = {
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 },
      { 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0  },
      { 2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0  },
      { 3, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0  },
      { 4, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0  },
      { 5, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0  },
      { 6, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0  },     FIG : 1.9
      { 7, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1  },
      { 8, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0  },
      { 9, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1  },
      { 10, 0, 0, 0, 0, 0, 0, 1, 0, 1,0  }
                        };
*/
/*#define   N 8
int adj[ ][N + 1] = {
      { 0, 1, 2, 3, 4, 5, 6, 7, 8 },
      { 1, 0, 1, 1, 0, 0, 0, 0, 0 },
      { 2, 1, 0, 0, 1, 1, 0, 0, 0 },
      { 3, 1, 0, 0, 0, 0, 1, 1, 0 },
      { 4, 0, 1, 0, 0, 0, 0, 0, 1 },    FIG : 2
      { 5, 0, 1, 0, 0, 0, 0, 0, 1 },
      { 6, 0, 0, 1, 0, 0, 0, 0, 1 },
      { 7, 0, 0, 1, 0, 0, 0, 0, 1 },
      { 8, 0, 0, 0, 1, 1, 1, 1, 0 }
                        };*/

int visited[ N + 1] = { 0 };
int queue[ N ];
int rear = -1;
int front = 0;
int max = 20;

void queueFull( void )
{
      printf( "\n\nERROR: Queue is Full\n\n" );
}

int queueEmpty( void )
{
      if( front > rear )
            return 1;
      else
            return 0;
}

void addq( int item )
{
      queue[ ++ rear ] = item;
}

int deleteq( void )
{
      int temp;
      temp = queue[ front ++];
      if( front > rear )
      {
            front = 0;
            rear = - 1;
      }
      return temp;
}

void bfs(int v )
{
      int u, w;
      int i = 1;
      u = v;
      visited[ v ] = 1;

      do {
            for( w = 1; w < = N ; ++ w )
            {
                  if( adj[ u ][ w ] && ! visited[ w ] )
```

```
                        {
                                addq( w );
                                visited[ w ] = 1;
                        }
                }
                printf("%d, ", u );
                if( queueEmpty( ) ) return;
                u = deleteq( );
        }while( 1 );

}

int main ( void )
{
        int i, j;
        printf( "Adjacency matrix:\n\n" );
        for( i = 0; i < = N; ++ i )
        {
                for( j = 0; j < = N ; ++ j )
                        printf( " %d\t ", adj[ i ][ j ] );
                printf( "\n\n" );
        }
        printf( "\n\nBFS Traversals\n\n" );
        for( i = 1; i < = N; ++ i )
        {
                if( ! visited[ i ])
                {
                        bfs( i );
                        printf( "\n\n" );
                }
        }
        printf( "\n" );
        return 0;
}
```

7 b) Check whether the given graph is connected or not using DFS method.

CODE:

```
#include    < stdio.h >

#define     N 10
int adj[ ][N +1]={
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10},
      { 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0},
      { 2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
      { 3, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0},      FIG : 2.1
      { 4, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
      { 5, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0},
      { 6, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0},
      { 7, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1},
      { 8, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0},
      { 9, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1},
      {10, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0}
                        };

/*
#define     N 12
int adj[ ][N + 1] =   {
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 },
      { 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0 , 0 ,  0 },
      { 2, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0 , 0 ,  0 },
      { 3, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0 , 0 ,  0 },
      { 4, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0 , 0 ,  0 },
      { 5, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0 , 0 ,  0 },
      { 6, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1 , 0 ,  0 },  FIG : 2.2
      { 7, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0 , 1 ,  0 },
      { 8, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0 , 1 ,  0 },
      { 9, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1 , 0 ,  1 },
      {10, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0 , 0 ,  1 },
      {11, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0 , 0 ,  1 },
      {12, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 , 1 ,  0 }

      };
*/
/*
#define N 3
intadj[][N] = {
      { 0, 1, 2 },
      { 1, 0, 1 }, // vertices = 2
      { 2, 0, 0 }
                        };
*/
int visited[ N  + 1] = { 0 };

void dfs( int v )
{
      int w;
      visited [ v ] = 1;
      printf( "%d  ", v );
      for( w = 1; w < = N ; ++ w )
```

```
        {
                if( adj[ v ][ w ] && ! visited[ w ] )
                {
                        dfs( w );
                }
        }
}

int main ( void )
{
        int i, j;
        int flag = 0;
        printf( " Adjacency matrix:\n\n " );
        for( i = 0; i < = N; ++ i )
        {
                for( j = 0; j < = N ; ++ j )
                        printf(" %d\t ",adj[ i ][ j ] );
                printf( " \n\n " );
        }
        printf( "\n\nDFS Traversal..\n" );
        for( i = 1; i < = N ; ++ i )
        {
                if( ! visited[ i ])
                {
                        printf( " \n\n " );
                        dfs( i );
                }
        }
        Return 0;
}
```

PROGRAM 8 : Design, develop and execute a program in c to find a given set S = { s1,s2,......sn } of n positive integers whose sum is equal to a given positive integer .For example ,if S={ 1,2,5,6,8 } and d=9 there are two solutions { 1,2,6) and { 1,8 }. A suitable message is to be displayed if the given problem instance dosent have a solution.

ALGORITHM :

```
ALGORITHM SumOfSub( s, k, r )
//Find all subsets of w [ 1 : n ] that sum to m . The values of x [ j ] ,
// 1 ≤ j < k , have already been determined . s =  ∑ w [j] * x [ j ]
// and r = ∑ w [ j ] . The w [ j ]'s are in nondecreasing order.
// It is assumed that w [ 1 ] ≤ m  and   w [ j ] ≥ m .
{
       // Generate left child . Note : s + w [ k ] ≤ m  since Bk - 1 is true.
       x [ k ] := 1 ;
       if ( s + w [ k ] = m ) then write ( x [ 1 : k ] ) ;  //subset found
           //There is no recursive call here as w [ j ] > 0 , 1 ≤ j ≤ n .
     else if( s + w [ k ] + w [ k + 1 ] ≤  m )
         then SumOfSub( s + w [ k ] , k + 1 , r - w [ k ] ) ;
     // Generate right child and evaluate Bk.
     if ( ( s + r - w [ k ] ≥  m ) and ( s + w [ k + 1 ]  ≤  m ) )  then
       {
            x [ k ] := 0 ;
            SumOfSub  ( s, k + 1, r - w [ k ] ) ;
       }
}
```

CODE :

```
#include    < stdio.h >

int n = 6;
int m = 30;
int w[ ]= {0, 5, 10, 12, 13, 15, 18 };
int x[ 20 ];
int howManySol;

void sumOfSub( int s, int k, int r )
{
      int i = 0;
      x[ k ] = 1;
      if( s + w[ k ]= = m )
      {
            printf( "\nSolution: %d\n\n", howManySol + = 1 );
            printf( "{ " );
            while( i < = k )
            {
                  if( x[ i ]! = 0)
                        printf( " %d ", w[ i ] );
                  ++i;
            }
            printf( "} = %d\n\n", m );
      }elseif( s + w[ k ] + w[ k + 1 ]< = m )
```

```
            sumOfSub( s + w[ k ], k + 1, r - w[ k ] );
      if(( s + r - w[ k ] > = m ) && ( s + w [k + 1] < = m ))
      {
            x[ k ] = 0;
            sumOfSub( s, k +1, r - w[ k ]);
      }
}

int main  ( void )
{
      sumOfSub ( 0, 1, 73 );
      if( ! howManySol ) printf( "\n\n No solution exist\n\n" ) ;
      return 0;
}
```

PROGRAM 9 : Floyd's Algorithm.

ALGORITHM :

```
Algorithm ALLPaths ( cost, A, n )
// cost [ 1 : n, 1 : n ] is the cost of adjacency matrix of graph with n
vertices; A [ i, j ] is the cost of a shotest
// path from vertex i to vertex j. cost [ i, j ] = 0.0, for 1 ≤ i ≤ n.
{
      for i := 1 to n do
              for j := 1 to n do
                    A [ i, j ] := cost [ i, j ] ; // Copy cost into A.
      for k := 1 to n do
              for i := 1 to n do
                    for j := 1 to n do
                          A [ i, j ] := min ( A [ i, j ], A [ i, k ] + A [ k,
j ] ) ;
}
```

CODE:

```
#include    < stdio.h >

#define N 5
int cost[ ][ N + 1 ] = {
      { 0, 1, 2, 3, 4, 5 },
      { 1, 0, 5, ∞, 2, ∞ },    FIG : 2.3
      { 2, ∞, 0, 2, ∞, ∞ },
      { 3, 3, ∞, 0, ∞, 7 },
      { 4, ∞, ∞, 4, 0, 1 },
      { 5, 1, 3, ∞, ∞, 0 }
};

int a [ N + 1 ][ N + 1 ];

int min ( int x, int y )
{
      if( x < y ) return x;
      else return y;
}

Void allPaths( void )
{
      int i, j, k;

      printf( "\nMatrix A ^ %d\n\n", 0 );
      for( i = 1; i < = N ; ++ i ){
            for( j = 1; j < = N ; ++ j )
                  printf("%d\t", a[ i ][ j ] = cost[ i ][ j ] );
            printf( "\n\n" );
      }
      for( k = 1; k < = N ; ++ k )
      {
            printf( " \nMatrix A ^ %d\n\n", k );
            for( i = 1; i < = N ; ++ i )
            {
                  for( j = 1; j < = N ; ++ j )
```

```
                        printf( " %d\t ", a[ i ][ j ] = min ( a[ i ][ j ], a[
i ][ k ]+ a[ k ][ j ] ) ) ;
                  printf( " \n\n " ) ;
            }
            printf( "\n\n" );
      }
}

int main ( void )
{
      allPaths( );
      return 0;
}
```

PROGRAM 9 : Floyd’s Algorithm.

ALGORITHM :

```
Algorithm ALLPaths ( cost, A, n )
// cost [ 1 : n, 1 : n ] is the cost of adjacency matrix of graph with n
vertices; A [ i, j ] is the cost of a shotest
// path from vertex i to vertex j. cost [ i, j ] = 0.0, for 1 ≤ i ≤ n.
{
      for i := 1 to n do
             for j := 1 to n do
                   A [ i, j ] := cost [ i, j ] ; // Copy cost into A.
      for k := 1 to n do
             for i := 1 to n do
                   for j := 1 to n do
                         A [ i, j ] := min ( A [ i, j ], A [ i, k ] + A [ k,
j ] ) ;
}
```

CODE:

```
#include    < stdio.h >

#define N 5
int cost[ ][ N + 1 ] = {
      { 0, 1, 2, 3, 4, 5 },
      { 1, 0, 5, ∞, 2, ∞ },   FIG : 2.3
      { 2, ∞, 0, 2, ∞, ∞ },
      { 3, 3, ∞, 0, ∞, 7 },
      { 4, ∞, ∞, 4, 0, 1 },
      { 5, 1, 3, ∞, ∞, 0 }
};

int a [ N + 1 ][ N + 1 ];

int min ( int x, int y )
{
      if( x < y ) return x;
      else return y;
}

Void allPaths( void )
{
      int i, j, k;

      printf( "\nMatrix A ^ %d\n\n", 0 );
      for( i = 1; i < = N ; ++ i ){
            for( j = 1; j < = N ; ++ j )
                  printf("%d\t", a[ i ][ j ] = cost[ i ][ j ] );
            printf( "\n\n" );
      }
      for( k = 1; k < = N ; ++ k )
      {
            printf( " \nMatrix A ^ %d\n\n", k );
            for( i = 1; i < = N ; ++ i )
            {
                  for( j = 1; j < = N ; ++ j )
```

```
                        printf( " %d\t ", a[ i ][ j ] = min ( a[ i ][ j ], a[
i ][ k ]+ a[ k ][ j ] ) ) ;
                  printf( " \n\n " ) ;
            }
            printf( "\n\n" );
      }
}

int main ( void )
{
      allPaths( );
      return 0;
}
```

PROGRAM 9 : Floyd's Algorithm.

ALGORITHM :

```
Algorithm ALLPaths ( cost, A, n )
// cost [ 1 : n, 1 : n ] is the cost of adjacency matrix of graph with n
vertices; A [ i, j ] is the cost of a shotest
// path from vertex i to vertex j. cost [ i, j ] = 0.0, for 1 ≤ i ≤ n.
{
      for i := 1 to n do
             for j := 1 to n do
                   A [ i, j ] := cost [ i, j ] ; // Copy cost into A.
      for k := 1 to n do
             for i := 1 to n do
                   for j := 1 to n do
                         A [ i, j ] := min ( A [ i, j ], A [ i, k ] + A [ k,
j ] ) ;
}
```

CODE:

```
#include    < stdio.h >

#define N 5
int cost[ ][ N + 1 ] = {
      { 0, 1, 2, 3, 4, 5 },
      { 1, 0, 5, ∞, 2, ∞ },   FIG : 2.3
      { 2, ∞, 0, 2, ∞, ∞ },
      { 3, 3, ∞, 0, ∞, 7 },
      { 4, ∞, ∞, 4, 0, 1 },
      { 5, 1, 3, ∞, ∞, 0 }
};

int a [ N + 1 ][ N + 1 ];

int min ( int x, int y )
{
      if( x < y ) return x;
      else return y;
}

Void allPaths( void )
{
      int i, j, k;

      printf( "\nMatrix A ^ %d\n\n", 0 );
      for( i = 1; i < = N ; ++ i ){
            for( j = 1; j < = N ; ++ j )
                  printf("%d\t", a[ i ][ j ] = cost[ i ][ j ] );
            printf( "\n\n" );
      }
      for( k = 1; k < = N ; ++ k )
      {
            printf( " \nMatrix A ^ %d\n\n", k );
            for( i = 1; i < = N ; ++ i )
            {
                  for( j = 1; j < = N ; ++ j )
```

```
                        printf( " %d\t ", a[ i ][ j ] = min ( a[ i ][ j ], a[
i ][ k ]+ a[ k ][ j ] ) ) ;
                  printf( " \n\n " ) ;
            }
            printf( "\n\n" );
      }
}

int main ( void )
{
      allPaths( );
      return 0;
}
```

PROGRAM 9 : Floyd's Algorithm.

ALGORITHM :

```
Algorithm ALLPaths ( cost, A, n )
// cost [ 1 : n, 1 : n ] is the cost of adjacency matrix of graph with n
vertices; A [ i, j ] is the cost of a shotest
// path from vertex i to vertex j. cost [ i, j ] = 0.0, for 1 ≤ i ≤ n.
{
      for i := 1 to n do
             for j := 1 to n do
                   A [ i, j ] := cost [ i, j ] ; // Copy cost into A.
      for k := 1 to n do
             for i := 1 to n do
                   for j := 1 to n do
                         A [ i, j ] := min ( A [ i, j ], A [ i, k ] + A [ k,
j ] ) ;
}
```

CODE:

```
#include    < stdio.h >

#define N 5
int cost[ ][ N + 1 ] = {
      { 0, 1, 2, 3, 4, 5 },
      { 1, 0, 5, ∞, 2, ∞ },    FIG : 2.3
      { 2, ∞, 0, 2, ∞, ∞ },
      { 3, 3, ∞, 0, ∞, 7 },
      { 4, ∞, ∞, 4, 0, 1 },
      { 5, 1, 3, ∞, ∞, 0 }
};

int a [ N + 1 ][ N + 1 ];

int min ( int x, int y )
{
      if( x < y ) return x;
      else return y;
}

Void allPaths( void )
{
      int i, j, k;

      printf( "\nMatrix A ^ %d\n\n", 0 );
      for( i = 1; i < = N ; ++ i ){
            for( j = 1; j < = N ; ++ j )
                  printf("%d\t", a[ i ][ j ] = cost[ i ][ j ] );
            printf( "\n\n" );
      }
      for( k = 1; k < = N ; ++ k )
      {
            printf( " \nMatrix A ^ %d\n\n", k );
            for( i = 1; i < = N ; ++ i )
            {
                  for( j = 1; j < = N ; ++ j )
```

```
                        printf( " %d\t ", a[ i ][ j ] = min ( a[ i ][ j ], a[
i ][ k ]+ a[ k ][ j ] ) ) ;
                  printf( " \n\n " ) ;
            }
            printf( "\n\n" );
      }
}

int main ( void )
{
      allPaths( );
      return 0;
}
```