NEW HORIZON
COLLEGE OF ENGINEERING

Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

50 Golden JUBILEE
YEARS OF EXCELLENCE IN EDUCATION

Name	HALVI SAI ANUSHA				
USN	1NH18CS071	Year	2020-2021		
Program	B.E - CSE	Sem	5	Sec	A
Course	ANALYSIS AND DESIGN OF ALGORITHM	Course Code	20CSL57		

NEW HORIZON COLLEGE OF ENGINEERING
Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

Golden JUBILEE
YEARS OF EXCELLENCE IN EDUCATION

Laboratory Certificate

This is to certify that

*Ms./Mr. ....**HALVI SAI ANUSHA**....................................*

has satisfactorily completed the experiments prescribed by New Horizon college of Engineering, Bangalore Affiliated to Visvesvaraya Technological University

*in ....**Analysis and Design of Algorithms**........... Laboratory Course for the...**5th**......semester of*

Computer Science and Engineering Program.

Student Name: Halvi Sai Anusha

USN: 1NH18CS071

Sem/ Sec:5TH SEM 'A' SEC

Course Code: 20CSE57

Marks Obtained
Max. Marks

Signature of Student

Signature of the Faculty In-charge

Head of the Department

NEW HORIZON COLLEGE OF ENGINEERING
Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

Golden Jubilee
YEARS OF EXCELLENCE IN EDUCATION

LABORATORY PERFORMANCE EVALUATION SHEET

Name of Student: Halvi Sai Anusha
USN: 1NH18CS071
Lab Course: ANALYSIS AND DESIGN OF ALGORITHMS LAB
Course Code: 20CSL57
Sem/Sec: 5a
Session: AFTERNOON SESSION

CIE- PART A- Record and Performance (Max Marks: 10)

SN	Date of Evaluation	Name of Experiment/ Program	1	2	3	4	Total	Faculty Signature
1	09/10/2020	Write a program to find GCD of two numbers using two different algorithms.						
2	09/10/2020	Write a program to implement Sieve of Eratosthenes to generate prime numbers between a given range						
3	16/10/2020	Write a Python program to implement String matching using Brute force						
4	16/10/2020	Write a program to implement Merge sort						
5	23/10/2020	Write a python program to implement Quick sort.						
6	23/10/2020	Write a program to obtain minimum cost spanning tree using Prim's algorithm.						

7	**13/11/2020**	Write a program to obtain minimum cost spanning tree using Kruskal’s algorithm.						
8	**13/11/2020**	Write a program to find the shortest path using Djkshatra’s algorithm.						
9	**20/11/2020**	Write a program to compute binomial co-efficient.						
10	**20/11/2020**	Write a program to obtain shortest path using Floyd’s algorithm						
11	**04/12/2020**	Write a program to compute transitive closure using Warshall’s algorithm.						
12	**04/12/2020**	Write a program to implement Breadth first search.						
13	**04/12/2020**	Write a program to implement depth first search.						
14	**04/12/2020**	Write a program to implement Topological sorting						
15	**11/12/2020**	To write a program to implement Sum of Subset problem using backtracking						
16	**11/12/2020**	To write a program to implement N Queens problem using backtracking						

1. **Conduction of Experiment/ Writing the Program:03 Marks**
2. **Specimen Calculation / Results: 02 Marks**
3. **Record Writing: 3 Marks**
4. **Viva Voce: 2 Marks**

NEW HORIZON COLLEGE OF ENGINEERING
Autonomous College, Affiliated to VTU | Approved by AICTE New Delhi & UGC
Accredited by NAAC with 'A' Grade & Accredited by NBA

Golden JUBILEE
YEARS OF EXCELLENCE IN EDUCATION

CIE- PART B- Lab Test (Max Marks: 50)

	Date of Lab Test	Procedure and Write Up (15 Marks)	Conduction and Results (25 Marks)	Viva Voce (10 Marks)	Total (50 Marks)	Faculty Signature
Test 1						
Test 2						
					AVG Marks (out of 15 marks)	

CIE- Marks Obtained

CIE-Part A Record and Performance (10 Marks)	CIE-Part B Lab Test (Scaled to 15 Marks)	Total (25 Marks)	Faculty Signature

PROGRAM NO.1

AIM: To Write a program to find GCD of two numbers using two different algorithms.

ALGORITHM:

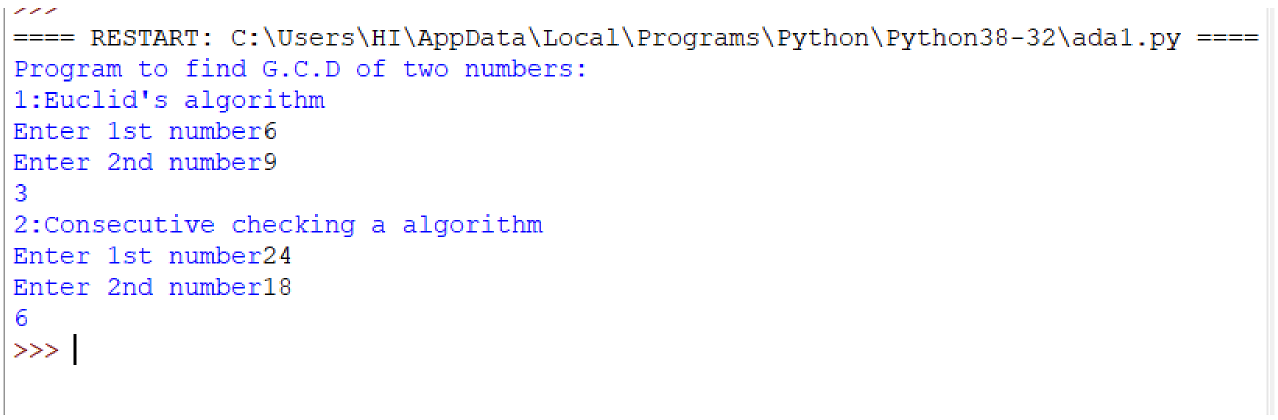
```
Algorithm Euclid(m,n):
While n!=0 do
 r<-m mod n
m<-n
n<-r
Return m
Algorithm Consecutive(m,n):
1)Find min(m, n)
2)Do step 1 until m%s==0 and n%s==0
```

PROGRAM:

```
def gcde(m,n):
    if m<n:
        (m,n)=(n,m)
    while n!=0:
            r=m%n
            m=n
            n=r
    return m


def gcdc(m,n):
    t=min(m,n)
    while t>0:
        if m%t==0 and n%t==0:
            return t
        t=t-1
```

```
print("Program to find G.C.D of two numbers:")
print("1:Euclid's algorithm")
m=int(input("Enter 1st number"))
n=int(input("Enter 2nd number"))
print(gcde(m,n))
print("2:Consecutive checking a algorithm")
m=int(input("Enter 1st number"))
n=int(input("Enter 2nd number"))
print(gcdc(m,n))
```

OUTPUT:



```
==== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\ada1.py ====
Program to find G.C.D of two numbers:
1:Euclid's algorithm
Enter 1st number6
Enter 2nd number9
3
2:Consecutive checking a algorithm
Enter 1st number24
Enter 2nd number18
6
>>> 
```

PROGRAM NO.2:

AIM: Write a program to implement Sieve of Eratosthenes to generate prime numbers between a given range

ALGORITHM:

1)Create a list of consecutive integers from 2 to n: (2, 3, 4, …, n).

2)Initially, let i equal 2, the first prime number.

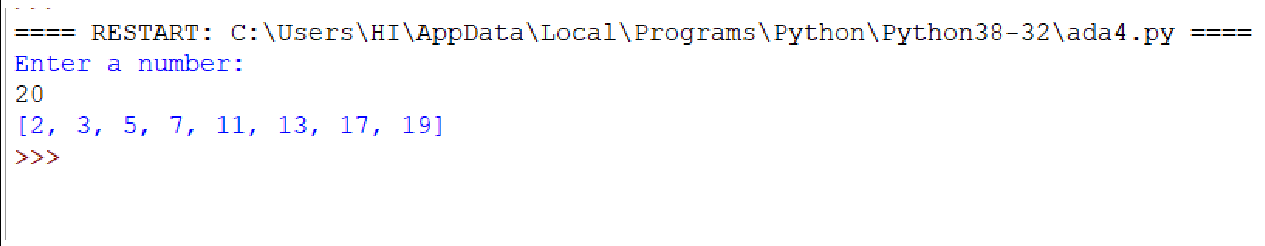
3)Starting from i2(square), count up in increments of p and mark each of these numbers greater than or equal to i2 itself in the list. These numbers will be i(i+1), i(i+2), i(i+3), etc.

4)Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let i now equal this number (which is the next prime), and repeat from step 3.

PROGRAM:

```
import math
print("Enter a number:")
number=int(input())
primes=[]
for i in range(2,number+1):
    primes.append(i)
i=2
while(i<=int(math.sqrt(number))):
    if i in primes:
        for j in range(i*2,number+1,i):
            if j in primes:
                primes.remove(j)
    i=i+1
print(primes)
```

OUTPUT:



```
==== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\ada4.py ====
Enter a number:
20
[2, 3, 5, 7, 11, 13, 17, 19]
>>>
```

PROGRAM NO 3:

AIM : Write a Python program to implement String matching using Brute force

ALGORITHM:

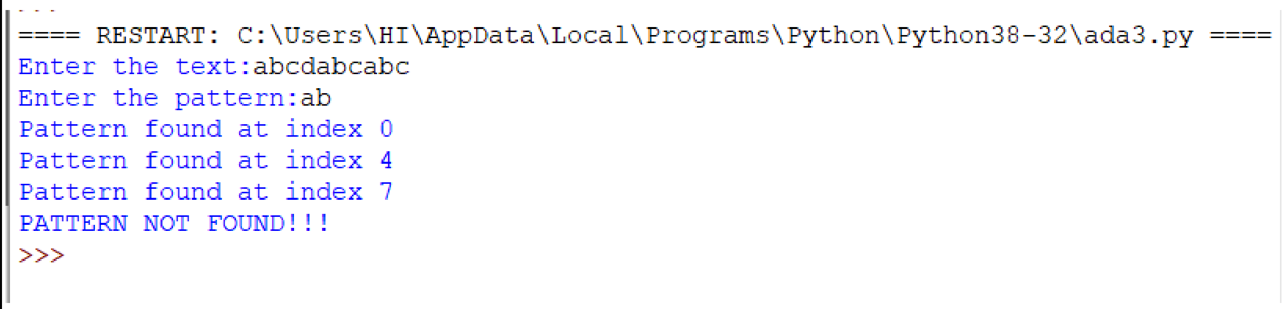
```
Algorithm BruteString(T[0..n-1],P[0…m-1]):
for i<-0 to n-m do
j<-0
while(j<m and P[j]=T[I+j])do
        j=j+1
        if j=m return I
return -1
```

PROGRAM:

```
#Python programfor naive pattern
#searching algorithm
def search(pat,txt):
   M=len(pat)
   N=len(txt)
   #Aloop tp slide pat[] one by one
   for i in range (N-M+1):
      j=0

      #for current index i,check
      #for pattern match
      while(j<M):
         if(txt[i+j]!=pat[j]):
            break
         j+=1

      if(j==M):
         print("Pattern found at index",i)
   else:
      print("PATTERN NOT FOUND!!!")
```

```
if __name__=='__main__':
    txt=input("Enter the text:")
    pat=input("Enter the pattern:")
    search(pat,txt)
```

OUTPUT:



```
==== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\ada3.py ====
Enter the text:abcdabcabc
Enter the pattern:ab
Pattern found at index 0
Pattern found at index 4
Pattern found at index 7
PATTERN NOT FOUND!!!
>>>
```

PROGRAM NO 4:

AIM : To write a program to implement Merge sort

ALGORITHM:

1)Copy the first half of the array into a new array.

2)Copy the second half of the array to another array.

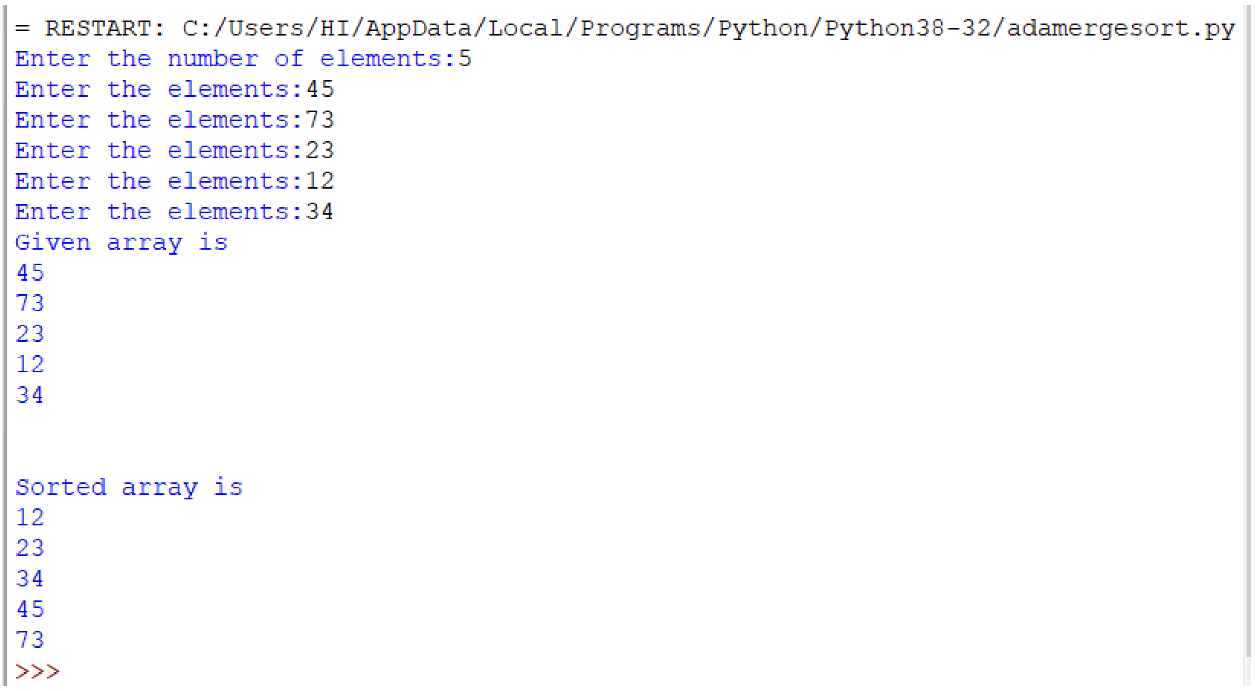
3)Continue dividing both the arrays until there are single elements left.

4)Then call Merge function which merges the arrays in ascending order.

PROGRAM:

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r- m

    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0 , n1):
        L[i] = arr[l + i]

    for j in range(0 , n2):
        R[j] = arr[m + 1 + j]

    i = 0     # Initial index of first subarray
    j = 0     # Initial index of second subarray
    k = l     # Initial index of merged subarray

    while i < n1 and j < n2 :
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # Copy the remaining elements of L[], if there
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if there
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr,l,r):
    if l < r:
        m = (l+(r-1))//2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)



n=int(input("Enter the number of elements:"))
a=[]
for i in range(0,n):
    ele=int(input("Enter the elements:"))
    a.append(ele)
#arr = [12, 11, 13, 5, 6, 7]
n = len(a)
print ("Given array is")
for i in range(n):
    print ("%d" %a[i]),


mergeSort(a,0,n-1)
print ("\n\nSorted array is")
for i in range(n):
    print ("%d" %a[i]),
```

OUTPUT:



```
= RESTART: C:/Users/HI/AppData/Local/Programs/Python/Python38-32/adamergesort.py
Enter the number of elements:5
Enter the elements:45
Enter the elements:73
Enter the elements:23
Enter the elements:12
Enter the elements:34
Given array is
45
73
23
12
34


Sorted array is
12
23
34
45
73
>>>
```

PROGRAM NO 5:

AIM : To write a python program to implement Quick sort.

AGORITHM:

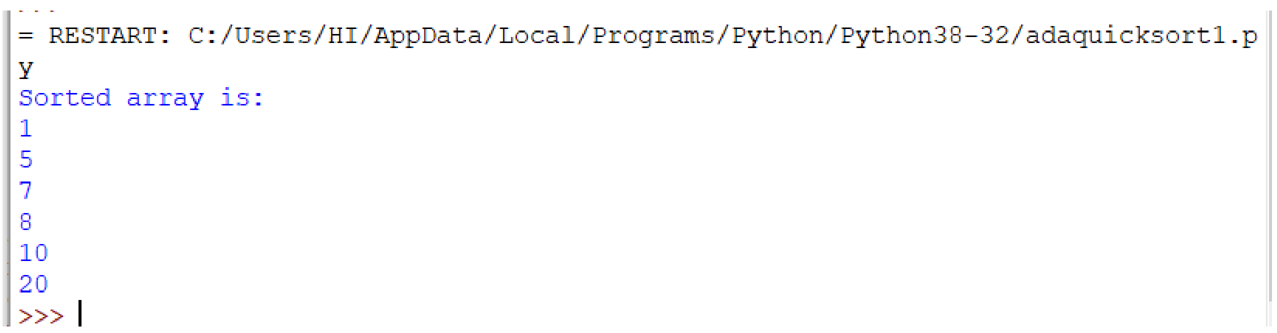
1)Pick a pivot element
2)Fix a higher bound and lower bound
3)Compare this pivot element with the value at higher bound, if the pivot is greater then swap.
4)If the pivot is less than higher bound then compare with lower bound and swap accordingly.

PROGRAM:

```
def partition(arr, low, high):
    i = (low-1)         # index of smaller element
    pivot = arr[high]     # pivot

    for j in range(low, high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:

            # increment index of smaller element
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low  --> Starting index,
# high  --> Ending index

# Function to do Quick sort


def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)


# Driver code to test above
```

```
arr = [20, 7, 8, 10, 1, 5]
n = len(arr)
quickSort(arr, 0, n-1)
print("Sorted array is:")
for i in range(n):
    print("%d" % arr[I]),
```

OUTPUT:



```
= RESTART: C:/Users/HI/AppData/Local/Programs/Python/Python38-32/adaquicksort1.p
y
Sorted array is:
1
5
7
8
10
20
>>> 
```

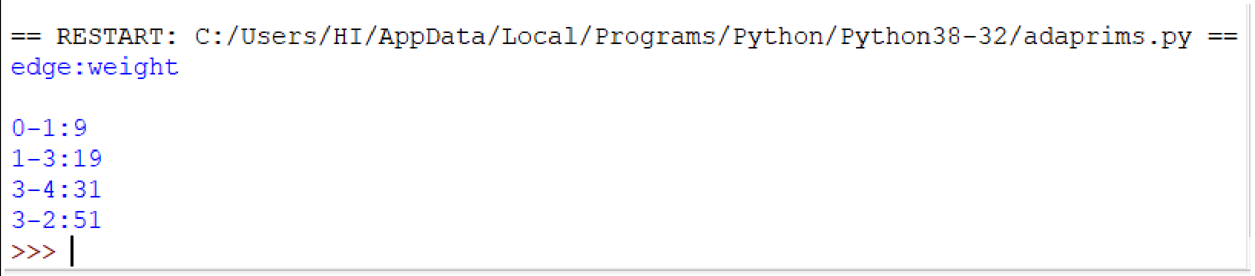
PROGRAM NO 6:
AIM : To write a program to obtain minimum cost spanning tree using Prim’s algorithm.

ALGORITHM:
1)Start from vertex 1 and let T<-phi(T will contain all edges in the spanning tree).
2)Next edge to be included in T is the minimum cost edge (u,v) such that u is in the tree and v is not.

PROGRAM :

```
INF=9999999
v=5
G=[[0,9,75,0,0],
 [9,0,95,19,42],
 [75,95,0,51,66],
 [0,19,51,0,31],
 [0,42,66,31,0]]
selected=[0,0,0,0,0]
no_edge=0
selected[0]=True
print("edge:weight\n")
while(no_edge<v-1):
 minimum=INF
 x=0
 y=0
 for i in range(v):
    if selected[i]:
        for j in range(v):
            if((not selected[j]) and (G[i][j])):
                if minimum > G[i][j]:
                    minimum=G[i][j]
                    x=i
                    y=j
 print(str(x)+"-"+str(y)+":"+str(G[x][y]))
 selected[y]=True
 no_edge+=1
```

OUTPUT:



```
== RESTART: C:/Users/HI/AppData/Local/Programs/Python/Python38-32/adaprims.py ==
edge:weight

0-1:9
1-3:19
3-4:31
3-2:51
>>> 
```

PROGRAM NO 7:

AIM : To write a program to obtain minimum cost spanning tree using Kruskal’s algorithm.

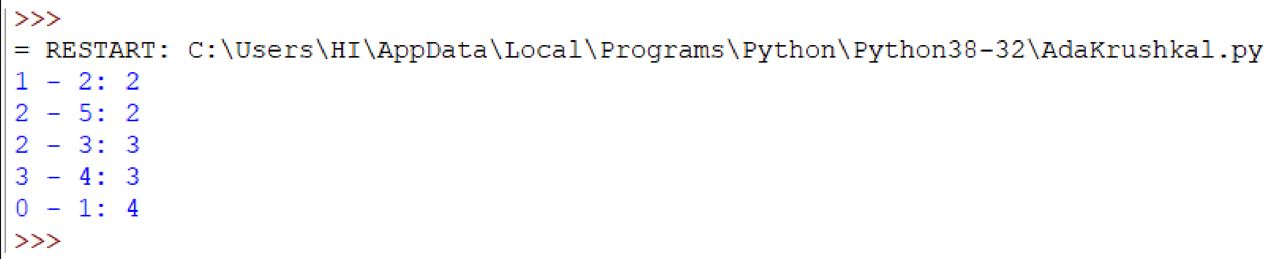
ALGORITHM:

1)Don’t consider if T is a tree or not in the intermediate state.
2)As long as the including of a new edge will not create a cycle, we include the minimum edge.

PROGRAM:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    # Search function

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def apply_union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    #  Applying Kruskal algorithm
    def kruskal_algo(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.apply_union(parent, rank, x, y)
        for u, v, weight in result:
            print("%d - %d: %d" % (u, v, weight))


g = Graph(6)
g.add_edge(0, 1, 4)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 2)
g.add_edge(1, 0, 4)
g.add_edge(2, 0, 4)
g.add_edge(2, 1, 2)
g.add_edge(2, 3, 3)
g.add_edge(2, 5, 2)
g.add_edge(2, 4, 4)
g.add_edge(3, 2, 3)
g.add_edge(3, 4, 3)
g.add_edge(4, 2, 4)
g.add_edge(4, 3, 3)
g.add_edge(5, 2, 2)
g.add_edge(5, 4, 3)
g.kruskal_algo()
```

OUTPUT:



```
>>>
= RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\AdaKrushkal.py
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4
>>>
```

PROGRAM NO 8:
AIM : To write a program to find the shortest path using Djkshatra’s algorithm.

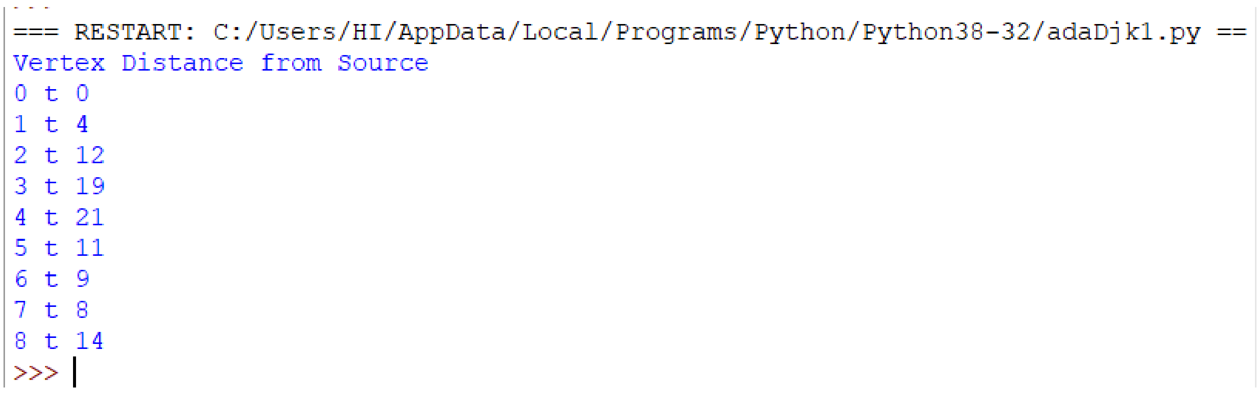
ALGORITHM:

1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalised. Initially, this set is empty.
2)Assign a distance value to all vertices in the input graph. Initialise all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3)While sptSet doesn’t include all vertices, Pick a vertex u which is not there in sptSet and has minimum distance value, include u to sptSet, update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

PROGRAM:

```
import sys
class Graph():
   def __init__(self,vertices):
      self.V=vertices
      self.graph=[[0 for column in range(vertices)]for row in range(vertices)]
   def printSolution(self,dist):
      print("Vertex Distance from Source")
      for node in range(self.V):
         print(node,"t",dist[node])
   def minDistance(self,dist,sptSet):
      min=sys.maxsize
      for v in range(self.V):
         if dist[v]<min and sptSet[v]==False:
            min=dist[v]
            min_index=v
      return min_index
   def dijkstra(self,src):
      dist=[sys.maxsize] * self.V
      dist[src]=0
      sptSet=[False] * self.V
      for cout in range(self.V):
         u=self.minDistance(dist,sptSet)
         sptSet[u]=True
         for v in range(self.V):
            if self.graph[u][v]>0 and sptSet[v]==False and dist[v]>dist[u] + self.graph[u][v]:
               dist[v]=dist[u] + self.graph[u][v]
      self.printSolution(dist)
g=Graph(9)


g.graph=[[0,4,0,0,0,0,0,8,0],[4,0,8,0,0,0,0,11,0],[0,8,0,7,0,4,0,0,2],
[0,0,7,0,9,14,0,0,0],[0,0,0,9,0,10,0,0,0],[0,0,4,14,10,0,2,0,0],
[0,0,0,0,0,2,0,1,6],[8,11,0,0,0,0,1,0,7],[0,0,2,0,0,0,6,7,0]];
g.dijkstra(0)
```

OUTPUT:



```
=== RESTART: C:/Users/HI/AppData/Local/Programs/Python/Python38-32/adaDjk1.py ==
Vertex Distance from Source
0 t 0
1 t 4
2 t 12
3 t 19
4 t 21
5 t 11
6 t 9
7 t 8
8 t 14
>>> 
```

PROGRAM NO 9:
AIM : To write a program to compute binomial co-efficient.

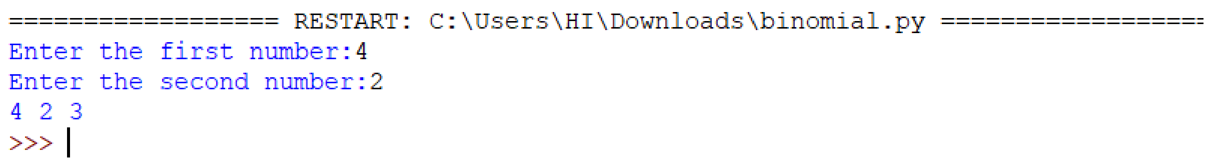
ALGORITHM:

```
1)The value of C(n,k) can be recursively calculated using the formula:
        C(n,k)=C(n-1,k-1)+C(n-1,k)
        C(n,0)=C(n,n)=1
```

PROGRAM:

```
def binomial(n,k):
   C=[[0 for x in range(k+1)]for x in range (n+1)]
   for i in range(n+1):
      for j in range(min(i,k)+1):
         if j==0 or j==1:
            C[i][j]=1
         else:
            C[i][j]=C[i-1][j-1]+C[i-1][j]
   return C[n][k]
n=int(input("Enter the first number:"))
k=int(input("Enter the second number:"))
print(n,k,binomial(n,k))
```

OUTPUT:



```
================== RESTART: C:\Users\HI\Downloads\binomial.py ==================
Enter the first number:4
Enter the second number:2
4 2 3
>>> |
```

PROGRAM NO 10:
AIM: To write a program to obtain shortest path using Floyd’s algorithm

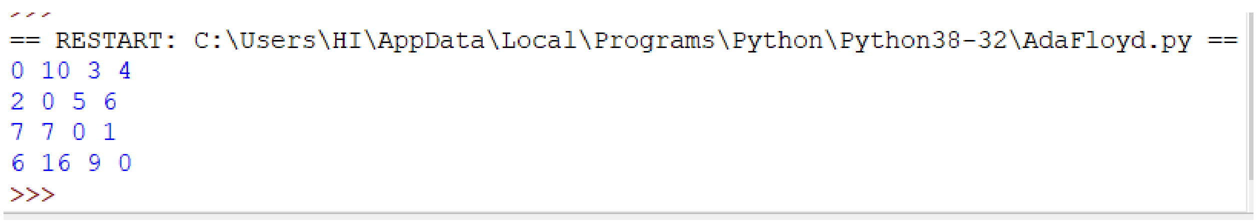
ALGORITHM:

1)If k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

PROGRAM:

```
def printSolution(cost,N):
    for i in range(N):
        for j in range(N):
            if(cost[i][j] == M):
                print("INF", end=" ")
            else:
                print(cost[i][j],end=" ")
        print(" ")


def floydWarshall(adjMatrix,N):
    cost = adjMatrix.copy()
    for k in range(N):
        for v in range(N):
            for u in range(N):
                if cost[v][k] != float('inf') and cost[k][u] !=float('inf') and (cost[v][k] +cost[k][u] <
cost[v][u]):
                    cost[v][u] = cost[v][k] + cost[k][u]
            if cost[v][u]< 0 :
                print("Negative Weight Cycle Found")
                return
    printSolution(cost, N)
if __name__ == '__main__':
    N = 4
    M = float('inf')
    adjMatrix = [
        [0, M, 3, M],
        [2, 0, M, M],
        [M, 7, 0, 1],
        [6, M, M, 0]
    ]
    floydWarshall(adjMatrix, N)
```

OUTPUT:



```
== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\AdaFloyd.py ==
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
>>>
```

PROGRAM NO 11:

AIM: To write a program to compute transitive closure using Warshall’s algorithm.

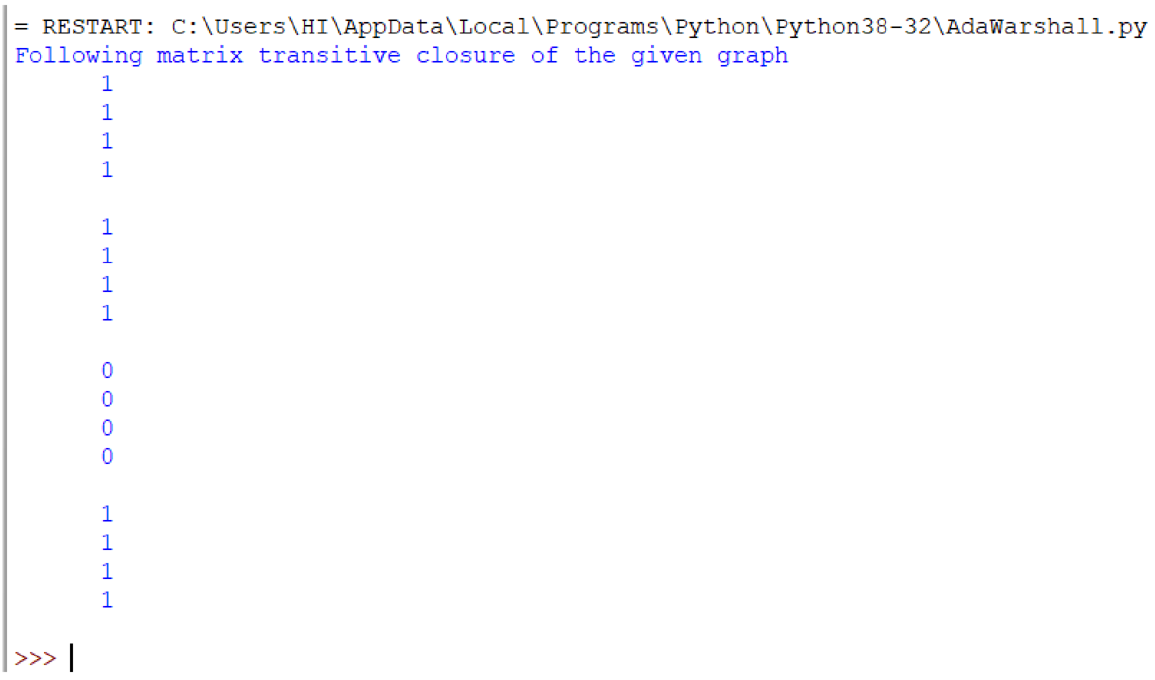
ALGORITHM:

1)We find if there exists a path between two vertices or not.
2)Then we use transitive closure method to find paths.

PROGRAM:

```
class Graph:
    def _init_(self, vertices):
        self.V = vertices
    def printSolution(self, reach):
        print("Following matrix transitive closure of the given graph")
        for i in range(self.V):
            for j in range(self.V):
                print("%7d\t"%(reach[i][j]))
            print("")
    def transitiveClosure(self,graph):
        reach = [i[:] for i in graph]
        for k in range(self.V):
            for i in range(self.V):
                for j in range(self.V):
                    reach[i][j] = reach[i][j] or (reach[i][k] and reach[k][j])
        self.printSolution(reach)
g = Graph(4)
graph = [
    [0, 1, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 0, 0],
    [1, 0, 1, 0]
]
g.transitiveClosure(graph)
```

OUTPUT:



```
= RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\AdaWarshall.py
Following matrix transitive closure of the given graph
        1
        1
        1
        1

        1
        1
        1
        1

        0
        0
        0
        0

        1
        1
        1
        1

>>> 
```

PROGRAM NO 12:

AIM : To write a program to implement Breadth first search.

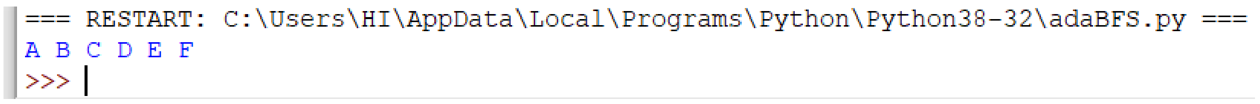
ALGORITHM:

1)Graphs may contain cycles .
2)To avoid processing a node more than once, we use a boolean visited array.
3) For simplicity, it is assumed that all vertices are reachable from the starting vertex.

PROGRAM:

```
graph={
   'A':['B','C'],
      'B':['A','D','E'],
      'C':['A','E'],
      'D':['B','E','F'],
      'E':['B','C','D','F'],
      'F':['D','E']
}
visited=[]
queue=[]
def bfs(visited,graph,node):
   visited.append(node)
   queue.append(node)
   while queue:
      s=queue.pop(0)
      print(s)

      for neighbour in graph[s]:
         if neighbour not in visited:
            visited.append(neighbour)
            queue.append(neighbour)
bfs(visited,graph,’A')
```

OUTPUT:



```
=== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\adaBFS.py ===
A B C D E F
>>> |
```

PROGRAM NO 13:

AIM : To write a program to implement depth first search.

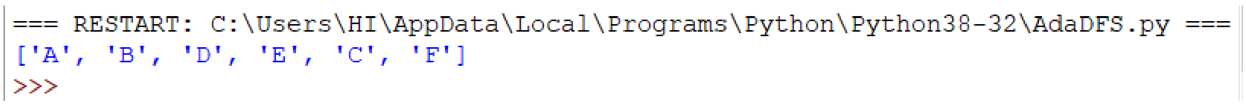
ALGORITHM:

1. Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes.
2.Repeat until all the nodes are visited, or the node to be searched is found.

PROGRAM:

```
graph={
    'A':['B','C'],
        'B':['A','D','E'],
        'C':['A','E'],
        'D':['B','E','F'],
        'E':['B','C','D','F'],
        'F':['D','E']
}
def dfs(graph,node,visited):
    if node not in visited:
        visited.append(node)
        for k in graph[node]:
            dfs(graph,k,visited)
    return visited
visited=dfs(graph,'A',[])
print(visited)
```

OUTPUT:



```
=== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\AdaDFS.py ===
['A', 'B', 'D', 'E', 'C', 'F']
>>>
```

PROGRAM NO 14:
AIM : To write a program to implement Topological sorting

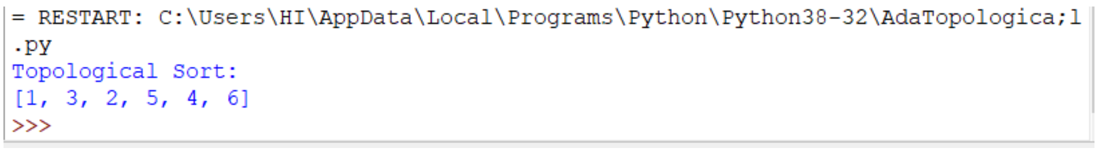
ALGORITHM:
1) In topological sorting, we use a temporary stack.
2)We first recursively call topological sorting for all its adjacent vertices, then push it to a stack.
3)A vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in the stack.

PROGRAM:

```
from collections import defaultdict
class Graph:
    def __init__(self, directed=False):
        self.graph = defaultdict(list)
        self.directed = directed
    def addEdge(self, frm, to):
        self.graph[frm].append(to)
        if self.directed is False:
            self.graph[to].append(frm)
        else:
            self.graph[to] = self.graph[to]
    def topoSortvisit(self, s, visited, sortlist):
        visited[s] = True
        for i in self.graph[s]:
            if not visited[i]:
                self.topoSortvisit(i, visited, sortlist)
        sortlist.insert(0, s)
    def topoSort(self):
        visited = {i: False for i in self.graph}
        sortlist = []

        for v in self.graph:
            if not visited[v]:
                self.topoSortvisit(v, visited, sortlist)
        print(sortlist)
if __name__ == '__main__':
```

```
g = Graph(directed=True)
g.addEdge(1, 2)
g.addEdge(1, 3)
g.addEdge(2, 4)
g.addEdge(2, 5)
g.addEdge(3, 4)
g.addEdge(3, 6)
g.addEdge(4, 6)


print("Topological Sort:")
g.topoSort()
```

OUTPUT:



```
= RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\AdaTopologica;l
.py
Topological Sort:
[1, 3, 2, 5, 4, 6]
>>>
```

PROGRAM NO 15:
AIM : To write a program to implement Subset Sum problem using backtracking

ALGORITHM:

1)Given a set A = {a1,…,an} of n positive integers, and a value d for which the algorithm has to find all subsets of A whose sum is equal to d.

2)Start with an empty set.

3)Add the next element from the set a to the set.

4)If the subset is having sum d, then stop with that subset as solution.

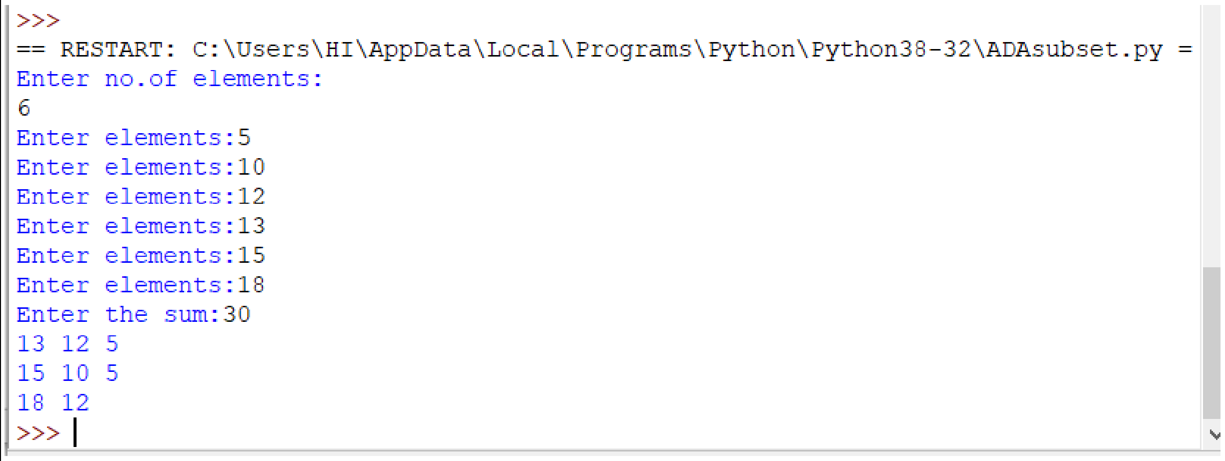
5)If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.

6)If the subset is feasible(sum of subset<d) then go to step 2
7)If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

PROGRAM:

```
def printAllSubsetRec(arr,n,v,sum):
    if(sum==0):
        for value in v:
            print(value,end=" ")
        print()
        return
    if(n==0):
        return
    printAllSubsetRec(arr,n-1,v,sum)
    v1=[]+v
    v1.append(arr[n-1])
    printAllSubsetRec(arr,n-1,v1,sum-arr[n-1])


def printAllSubsets(arr,n,sum):
    v=[]
    printAllSubsetRec(arr,n,v,sum)


arr=[]
num=int(input("Enter no.of elements:\n"))
for i in range(0,num):
    ele=int(input("Enter elements:"))
    arr.append(ele)
sum=int(input("Enter the sum:"))
n=len(arr)
printAllSubsets(arr,n,sum)
```

OUTPUT:



```
>>>
== RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\ADAsubset.py =
Enter no.of elements:
6
Enter elements:5
Enter elements:10
Enter elements:12
Enter elements:13
Enter elements:15
Enter elements:18
Enter the sum:30
13 12 5
15 10 5
18 12
>>>
```

PROGRAM NO 16:
AIM : To write a program to implement N Queens problem using backtracking

ALGORITHM:

1.Consider a 4X4 chess board.

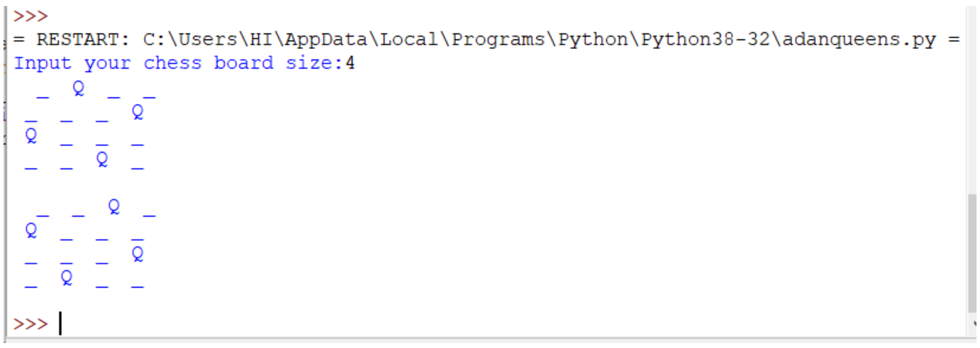
2.Let the four queens be Q1,Q2,Q3,Q4.

3.Place all the queens in the chess board such that no two queens are under attack.
4.Draw state space tree applying bounding function

PROGRAM:

```
def backtrack(Q,r):
    if r==len(Q):
        print_chess_board(Q)
    else :
        for j in range(len(Q)):
            legal = True
            for i in range(r):
                if((Q[i]==j)or(Q[i]==j+r-i)or(Q[i]==j-r+i)):
                    legal = False
            if legal :
                    Q[r]=j
                    backtrack(Q,r+1)


def print_chess_board(d):
    chess_board=' '
    for i in range(len(d)):
         for j in range(len(d)):
             if j==d[i]:
                 chess_board += " Q "
             else:
                 chess_board += " _ "
         chess_board += " \n"
    print(chess_board)


def create_chess_board(size):
    mat=[]
    for i in range(size):
        mat.append(0)
    return mat
if __name__ =='__main__':
    size=input("Input your chess board size:")
    backtrack(create_chess_board(int(size)),0)
```

OUTPUT:



```
>>>
= RESTART: C:\Users\HI\AppData\Local\Programs\Python\Python38-32\adanqueens.py =
Input your chess board size:4
 _  Q  _  _
_  _  _  Q
Q  _  _  _
_  _  Q  _

 _  _  Q  _
Q  _  _  _
_  _  _  Q
_  Q  _  _

>>> |
```