ACHARYA INSTITUTE OF GRADUATE STUDIES

(NAAC Re -Accredited ‘A’ and Affiliated to Bengaluru City University)

Soladevanahalli, Bengaluru-560107

DEPARTMENT OF COMPUTER APPLICATION

DESIGN AND ANALYSIS OF ALGORITHMS

LAB MANUAL-NEP SYLLABUS

SEMESTER : IV

COURSE CODE : CA-C19L

SYLLABUS

1. Write a program to implement linear search algorithm Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

2. Write a program to implement binary search algorithm. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

3. Write a program to solve towers of honai problem and execute it for different number of disks

4. Write a Program to Sort a given set of numbers using selection sort algorithm. 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. The elements can be read from a file or can be generated using the random number generator.

5. Write a program to find the value of an (where a and n are integers) using both brute-force based algorithm and divide and conquer based algorithm

6. Write a Program to Sort a given set of elements using quick sort algorithm. 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.

7. Write a Program to find the binomial co-efficient C(n, k), [where n and k are integers and n > k] using brute force based algorithm and also dynamic programming based algorithm

8. Write a Program to implement Floyd’s algorithm and find the lengths of the shortest paths from every pairs of vertices in a given weighted graph

9. Write a program to evaluate a polynomial using brute-force based algorithm and using Horner’s rule and compare their performances

10. Write a Program to solve the string matching problem using Boyer-Moore approach.

11. Write a Program to solve the string matching problem using KMP algorithm

12. Write a program to implement BFS traversal algorithm

13. Write a program to find the minimum spanning tree of a given graph using Prim’s algorithm

14. Write a Program to obtain the topological ordering of vertices in a given digraph. Compute the transitive closure of a given directed graph using Warshall's algorithm.

15. Write a Program to Find a subset of a given set S = {s1,s2, ,sn} of n positive integers whose sum is equal to a given positive integer d. 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 doesn't have a solution.

INDEX

PROGRAM NO -1

OBJECTIVE:

To implement linear search algorithm, Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

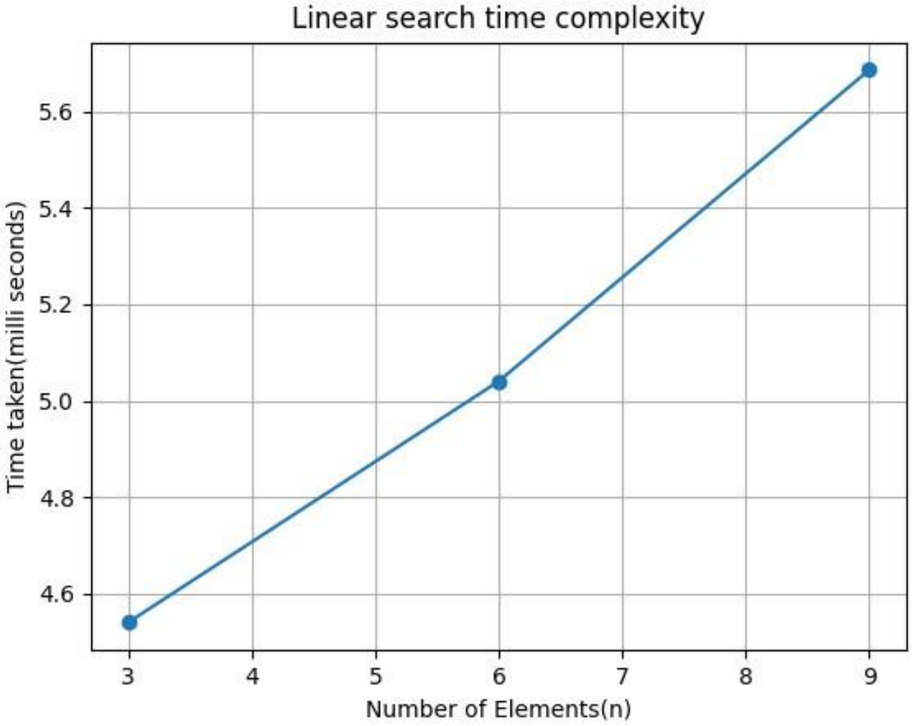
SOURCE CODE:

```
import time
import matplotlib.pyplot as plt

#Function to perform linear search
def Linear_search(arr, n, key):
    for i in range(n):
        if arr[i]==key:
            return i
    return -1

def linear_search(r):
    results=[]
    for _ in range(r):
        n=int(input("Enter the number of elements:"))
        arr=list(map(int, input("\n Enter the elements of array:").split()))
        key=int(input("\nEnter the key element to be searched"))

        #Repeat the search operation multiple times to amplify the time taken
        repeat=10000
        result=-1
        start=time.time()
        for _ in range(repeat):
            result=Linear_search(arr, n, key)
        end=time.time()
        if result !=-1:
            print(f"key {key} found at position {result}")
        else:
            print(f"key {key} not found")
        time_taken=(end - start)* 1000  #in milli seconds
        print(f"Time to search a key element={time_taken} milli seconds")

        #Record Number of elements and time taken
        results.append((n,time_taken))
    return results
#Function to plot results
```

```
def plot_results(results):
    #extract data
    n_values=[result[0] for result in results]
    times=[result[1] for result in results]

    #create Plot
    plt.figure()
    plt.plot(n_values,times,'o-')
    plt.xlabel('Number of Elements(n)')
    plt.ylabel('Time taken(milli seconds)')
    plt.title('Linear search time complexity')
    plt.grid(True)
    plt.show()

#main function
r=int(input("Enter the number of runs:"))
results=linear_search(r)
plot_results(results)
```

Output:

```
Enter the number of runs:3
Enter the number of elements:9
Enter the elements of array:5 8 1 4 6 9 7 2 5
Enter the key element to be searched6
key 6 found at position 4
Time to search a key element=5.686283111572266 milli seconds

Enter the number of elements:6
Enter the elements of array:8 3 4 7 9 5
Enter the key element to be searched9
key 9 found at position 4
Time to search a key element=5.040168762207031 milli seconds

Enter the number of elements:3
Enter the elements of array:5 4 6
Enter the key element to be searched6
key 6 found at position 2
Time to search a key element=4.539728164672852 milli seconds
```



PROGRAM NO -2

OBJECTIVE:

To implement binary search algorithm. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

PROGRAM NO -3

OBJECTIVE:

To solve towers of honai problem and execute it for different number of disks

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
def toh(n,source, temp, dest):
    global count
    if n>0:
        toh(n-1,source,dest, temp)
        print(f'Move Disk {n} {source}->{dest}')
        count+=1
        toh(n-1,temp, source, dest)
#main code
source='S'
temp='T'
dest='D'
count=0
n=int(input("Enter the number of disks:"))
print("Sequence is:")
toh(n,source,temp,dest)
print("The number of Moves:",count)
```

Output:

RUN 1:

```
Enter the number of disks:2
Sequence is:
Move Disk 1 S->T
Move Disk 2 S->D
Move Disk 1 T->D
The number of Moves: 3
```

RUN 2:

```
Enter the number of disks:3
Sequence is:
Move Disk 1 S->D
Move Disk 2 S->T
Move Disk 1 D->T
Move Disk 3 S->D
Move Disk 1 T->S
Move Disk 2 T->D
Move Disk 1 S->D
The number of Moves: 7
```

PROGRAM NO -4

OBJECTIVE:

To Sort a given set of numbers using selection sort algorithm. 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. The elements can be read from a file or can be generated using the random number generator.

PROCEDURE:

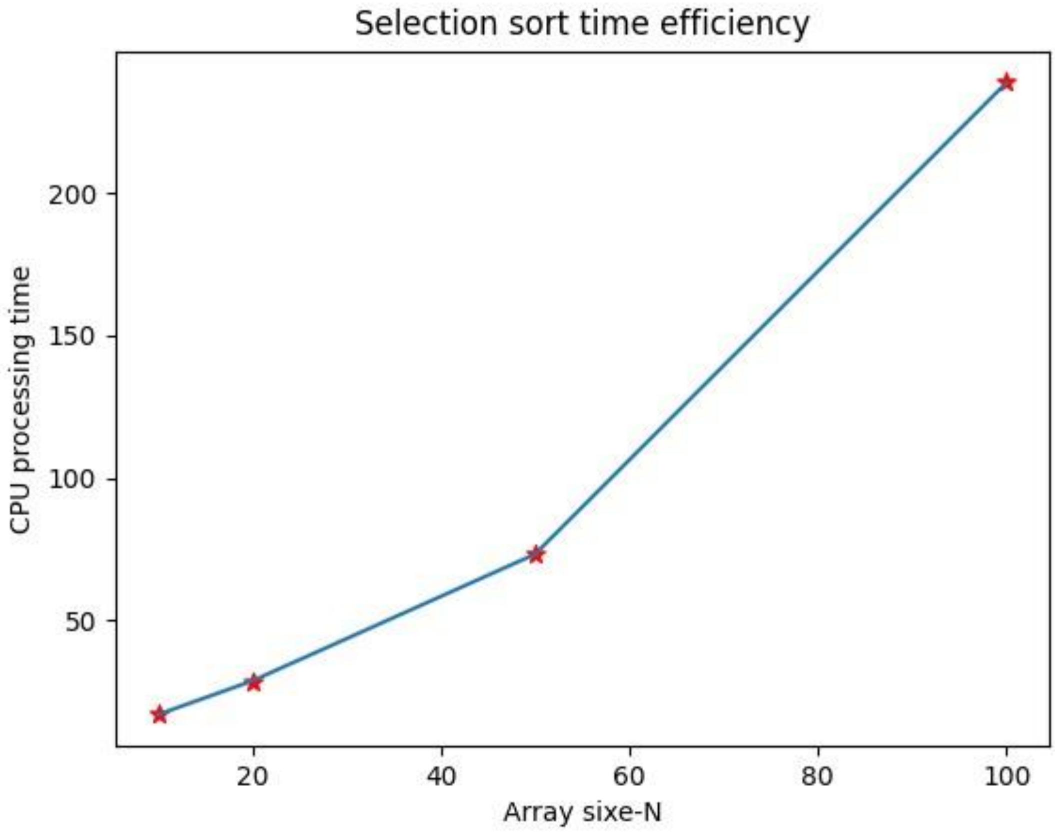
1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
import timeit
import random
import matplotlib.pyplot as plt
#Input array elements
def Input(array,n):
    #iterating till the range
    for i in range(0,n):
        ele=random.randrange(1,50)
        #adding the element
        array.append(ele)
#selection sort
def selectionsort(array,size):
    for ind in range(size):
        min_index=ind
        for j in range(ind+1,size):
            #select the minimum element in every iteration
            if array[j]<array[min_index]:
                min_index=j
        #swapping the elements to sort the array
        (array[ind],array[min_index])=(array[min_index],array[ind])
#Main block
N=[]
cpu=[]
trail=int(input('Enter number of trails'))
for t in range(0,trail):
    array=[]
    print("----->trail no:",t+1)
    n=int(input("Enter number of elements:"))
    Input(array,n)
    start=timeit.default_timer()
    selectionsort(array,n)
    times=timeit.default_timer()-start
    print("sorted array")
    print(array)
    N.append(n)
    cpu.append(round(float(times)*1000000,2))

print("n cpu")
for t in range(0,trail):
    print(N[t],cpu[t])

#plotting graph
plt.plot(N,cpu)
plt.scatter(N,cpu,color='red',marker='*',s=50)

#naming the x axis
plt.xlabel('Array sixe-N')
plt.ylabel('CPU processing time')
plt.title('Selection sort time efficiency')
plt.show()
```

Output:

```
Enter number of trails4
----->trail no: 1
Enter number of elements:10
sorted array
[16, 17, 24, 26, 34, 35, 38, 42, 46, 48]
----->trail no: 2
Enter number of elements:20
sorted array
[1, 3, 4, 5, 7, 10, 12, 13, 16, 18, 23, 24, 24, 25, 28, 29, 31, 32, 34, 34]
----->trail no: 3
Enter number of elements:50
sorted array
[1, 1, 5, 7, 7, 8, 9, 9, 10, 12, 12, 15, 15, 15, 16, 16, 17, 17, 20, 22, 23, 23, 24, 26, 26, 27, 27,
28, 30, 30, 31, 31, 33, 35, 35, 35, 36, 39, 39, 40, 40, 41, 43, 47, 48, 48, 49, 49, 49, 49]
----->trail no: 4
Enter number of elements:100
sorted array
[1, 1, 3, 4, 5, 5, 6, 7, 7, 8, 9, 9, 9, 9, 10, 11, 11, 11, 12, 13, 14, 14, 15, 15, 16, 16, 16, 16, 16,
17, 19, 19, 20, 20, 21, 21, 21, 22, 22, 23, 24, 24, 25, 25, 26, 26, 27, 27, 28, 28, 29, 29, 29, 29,
30, 30, 30, 30, 31, 33, 33, 33, 33, 34, 34, 34, 35, 35, 35, 35, 35, 36, 37, 37, 37, 37, 38, 39, 39,
40, 41, 41, 42, 42, 42, 42, 43, 43, 43, 44, 46, 47, 47, 48, 49, 49, 49, 49, 49, 49]
n cpu
10 16.9
20 28.7
50 73.2
100 238.7
```



PROGRAM NO -5

OBJECTIVE:

To find the value of an (where a and n are integers) using both brute-force based algorithm and divide and conquer based algorithm.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
def power_bruteforce(a,n):
    result=1
    for i in range(n):
        result*=a
    return result
def power_divide_conquer(a,n):
    if n==0:
        return 1
    elif n%2==0:
        return power_divide_conquer(a*a,n//2)
    else:
        return a*power_divide_conquer(a*a,n//2)

#main code
a,n=map(int,input("Enter the value of a and n:").split())

result_brute=power_bruteforce(a,n)
result_divide_conquer=power_divide_conquer(a,n)

print("Result using brute force:",result_brute)
print("Result using divide and conquer:",result_divide_conquer)
```

Output:

```
Enter the value of a and n:2 8
Result using brute force: 256
Result using divide and conquer: 256
```

PROGRAM NO -6

OBJECTIVE:

To Sort a given set of elements using quick sort algorithm. 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.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

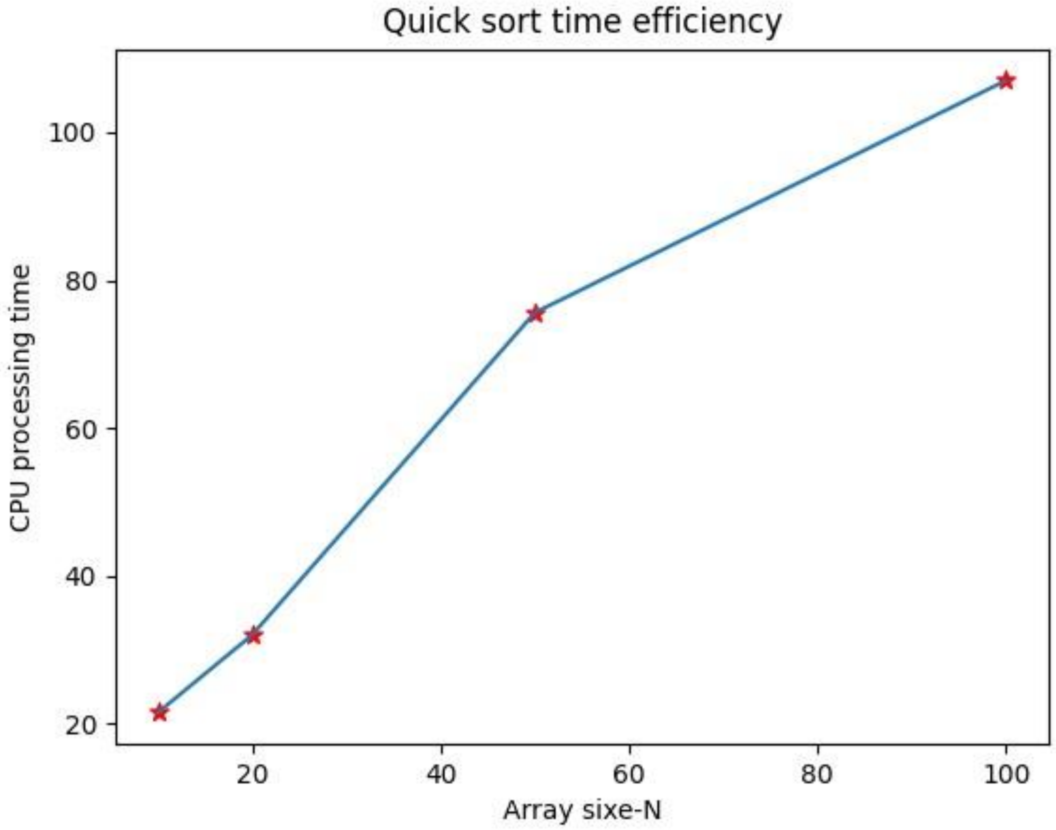
SOURCE CODE:

```
import timeit
import random
import matplotlib.pyplot as plt
#Input array elements
def Input(array,n):
    #iterating till the range
    for i in range(0,n):
        ele=random.randrange(1,50)
        #adding the element
        array.append(ele)
#divide function
def partition(array,low, high):
    i=(low-1)
    pivot=array[high]  #pivot element
    for j in range(low,high):
        #if current element is smaller
        if array[j]<=pivot:
            i=i+1
            array[i],array[j]=array[j],array[i]
    array[i+1],array[high]=array[high],array[i+1]
    return(i+1)
#quick sort
def quicksort(array,low,high):
    if low<high:
        #index
        pi=partition(array,low,high)
        #sort the partitions
        quicksort(array,low,pi-1)
        quicksort(array,pi+1,high)

#main block
N=[]
cpu=[]
trail=int(input("Enter number of trails"))
for t in range(0,trail):
    array=[]
    print("----->trail no:",t+1)
    n=int(input("Enter number of elements:"))
    Input(array,n)
    start=timeit.default_timer()
    quicksort(array,0,n-1)
    times=timeit.default_timer()-start
    print("sorted array")
    print(array)
    N.append(n)
    cpu.append(round(float(times)*1000000,2))
print("n cpu")
for t in range(0,trail):
    print(N[t],cpu[t])
#plotting graph
plt.plot(N,cpu)
plt.scatter(N,cpu,color='red',marker='*',s=50)
#naming the x axis
plt.xlabel('Array sixe-N')
plt.ylabel('CPU processing time')
plt.title('Quick sort time efficiency')
plt.show()
```

Output:

```
Enter number of trails4
----->trail no: 1
Enter number of elements:10
sorted array
[2, 7, 13, 17, 21, 23, 26, 27, 41, 46]
----->trail no: 2
Enter number of elements:20
sorted array
```

```
[3, 6, 9, 10, 19, 20, 20, 23, 24, 26, 27, 28, 30, 33, 34, 42, 42, 44, 45, 46]
----->trail no: 3
Enter number of elements:50
sorted array
[1, 3, 3, 6, 8, 9, 10, 12, 12, 12, 14, 15, 16, 18, 18, 18, 20, 24, 24, 25, 25, 27, 30, 30, 30, 31, 33,
35, 35, 35, 36, 36, 37, 37, 38, 38, 42, 43, 43, 44, 45, 46, 46, 47, 47, 47, 48, 49, 49, 49]
----->trail no: 4
Enter number of elements:100
sorted array
[1, 2, 5, 6, 6, 6, 7, 7, 7, 8, 8, 9, 9, 9, 9, 10, 10, 11, 12, 13, 14, 14, 15, 16, 16, 16, 17, 17, 18, 18,
19, 19, 19, 20, 20, 20, 21, 21, 22, 22, 22, 22, 23, 23, 24, 24, 24, 24, 25, 25, 25, 26, 26, 26, 26,
27, 27, 28, 29, 29, 30, 30, 30, 30, 30, 30, 31, 31, 33, 33, 33, 33, 34, 34, 34, 34, 35, 35, 37, 38,
38, 39, 42, 42, 42, 43, 43, 45, 46, 46, 47, 47, 47, 47, 48, 48, 48, 48, 49, 49]
n cpu
10 21.5
20 32.0
50 75.6
100 107.0
```



PROGRAM NO -7

OBJECTIVE:

To find the binomial co-efficient C(n, k), [where n and k are integers and n > k] using brute force based algorithm and also dynamic programming based algorithm.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
#Function to calculate factorial for brute force method
def factorial(n):
    fact=1
    for i in range(2,n+1):
        fact *=i
    return fact

#Brute force method to find binomical coefficient
def binomialcoeff_bruteForce(n,k):
    return factorial(n)//(factorial(k)*factorial(n-k))

#Dynamic programming method to find binomical coefficient
def binomialcoeff_DP(n,k):
    c=[[0 for j in range(k+1)] for i in range(n+1)]
    for i in range(n+1):
        for j in range(min(i,k)+1):
            #Base cases
            if j==0 or j==i:
                c[i][j]=1
            #calculating value using previously stored values
            else:
                c[i][j]=c[i-1][j-1]+c[i-1][j]
    return c[n][k]

#main code
n=int(input("Enter the value of n:"))
k=int(input("Enter the value of k:"))
result_bruteForce=binomialcoeff_bruteForce(n,k)
result_DP=binomialcoeff_DP(n,k)
print(f"Binomial coefficient(Brute Force): {result_bruteForce}")
print(f"Binomial coefficient(Dynamic Programming): {result_DP}")
```

Output:

```
Enter the value of n:5
Enter the value of k:2
Binomial coefficient(Brute Force): 10
Binomial coefficient(Dynamic Programming): 10
```

PROGRAM NO -8

OBJECTIVE:

To implement Floyd’s algorithm and find the lengths of the shortest paths fromevery pairs of vertices in a given weighted graph.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
INF=99999

#print the solution matrix
def printSolution(V,D):
    print("The following matrix shows the shortest distances between every pair of vertices")
    for i in range(V):
        for j in range(V):
            if D[i][j]==INF:
                print("%7s" % "INF",end="")
            else:
                print("%7d" % D[i][j],end="")
        print()

    #implementing floyd warshall algorithm
def floyd(V,C):
        D=[[0]*V for _ in range(V)]
        for i in range(V):
            for j in range(V):
                D[i][j]=C[i][j]

        for k in range(V):
            for i in range(V):
                for j in range(V):
                    if D[i][j]>(D[i][k]+D[k][j]):
                        D[i][j]=D[i][k]+D[k][j]
        printSolution(V,D)

#Main code
V=int(input("Enter the number of vertices:"))

#allocate memory for the cost matrix
```

```
C=[[0]*V for _ in range(V)]

print("Enter the cost matrix row by row (space - seperated):")
print("[Enter 99999 for indinity]")
print("[Enter 0 for cost(i,i)]")
for i in range(V):
    C[i]=list(map(int,input().split()))
floyd(V,C)
```

Output:

```
Enter the number of vertices:4
Enter the cost matrix row by row (space - seperated):
[Enter 99999 for indinity]
[Enter 0 for cost(i,i)]
0 99999 2 99999
3 0 99999 99999
99999 5 0 1
6 99999 99999 0
The following matrix shows the shortest distances between every pair of vertices
      0      7      2      3
      3      0      5      6
      7      5      0      1
      6     13      8      0
```

PROGRAM NO -9

OBJECTIVE:

To evaluate a polynomial using brute-force based algorithm and using Horner’s rule andcompare their performances.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
import time
import math
def bruteForce(coef, n, x):
    sum=0.0
    for i in range(n+1):
        sum+=coef[i]*math.pow(x,i)
    return sum


def hornerRule(coef,n,x):
    result=coef[n]
    for i in range(n-1,-1,-1):
        result=result*x+coef[i]
    return result

#Main code
n=int(input("Enter the degree of the polynomial:"))

coef=[0]*(n+1)
print("Enter the coefficients from highest degree to lowest")
for i in range(n, -1, -1):
    coef[i]=int(input())

x=float(input("Enter the value of x:"))
start=time.time()
brute_force_result=bruteForce(coef, n, x)
end=time.time()
time_used=end-start
print(f"Brute force result:{brute_force_result:.2f}, time used:{time_used:.6f} seconds")
start=time.time()
horners_rule_result=hornerRule(coef, n, x)
end=time.time()
time_used=end-start
print(f"Horner's rule result:{horners_rule_result:.2f}, time used:{time_used:.6f}
seconds")
```

Output:

```
Enter the degree of the polynomial:3
Enter the coefficients from highest degree to lowest
2
-6
2
-1
Enter the value of x:3
Brute force result:5.00, time used:0.000000 seconds
Horner's rule result:5.00, time used:0.000000 seconds
```

PROGRAM NO -10

OBJECTIVE:

To solve the string-matching problem using Boyer-Moore approach.

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
Max_chars=256
def max(a,b):
   return a if a >b else b
def badCharHeuristic(pat, size, badchar):
   for i in range(Max_chars):
      badchar[i]=-1
   for i in range(size):
      badchar[ord(pat[i])]=i
def patternsearch(text, pat):
   m=len(pat)
   n=len(text)
   badchar= [-1]*Max_chars
   badCharHeuristic(pat,m,badchar)
   s=0
   while s<=(n-m):
      j=m-1
      while j>=0 and pat[j]==text[s+j]:
         j-=1
      if j<0:
         print("\n patterrn occors at position=",s)
         s+=m- badchar[ord(text[s+m])]if (s+m)<n else 1
      else:
         s+=max(1,j- badchar[ord(text[s+j])])


#main code
text=input("Enter the text:").rstrip('\n')
pat=input("Enter the Pattern:").rstrip('\n')
patternsearch(text,pat)
```

Output:

Run-1:

```
Enter the text:Acharya College
Enter the Pattern:Coll
patterrn occors at position= 8
```

Run-2:

```
Enter the text:Acharya College
Enter the Pattern:a
patterrn occors at position= 3
patterrn occors at position= 6
```

PROGRAM NO -11

OBJECTIVE:

To solve the string-matching problem using KMP algorithm

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
def computeArray(pat,m,lps):
    length=0
    lps[0]=0
    i=1
    while i<m:
        if pat[i]==pat[length]:
            length+=1
            lps[i]=length
            i+=1
        else:
            if length !=0:
                length=lps[length-1]
            else:
                lps[i]=0
                i+=1
def KMPSearch(pat, txt):
    m=len(pat)
    n=len(txt)
    lps=[0]*m
    computeArray(pat,m,lps)
    i=j=0
    while i<n:
        if pat[j]==txt[i]:
            i+=1
            j+=1
        if j==m:
            print(f"Found Pattern at index{i-j}")
            j=lps[j-1]
        elif i<n and pat[j]!=txt[i]:
            if j!=0:
                j=lps[j-1]
            else:
                i+=1
#main code
txt=input("Enter the text:")
pat=input("Enter the Pattern:")
KMPSearch(pat,txt)
```

Output:

Run-1:

```
Enter the text:Acharya College
Enter the Pattern:Coll
Found Pattern at index8
```

Run-2:

```
Enter the text:Acharya College
Enter the Pattern:a
Found Pattern at index3
Found Pattern at index6
```

PROGRAM NO -12

OBJECTIVE:

To implement BFS traversal algorithm

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
Max=100

c=[[0]*Max for _ in range(Max)]
visited= [0]*Max
queue=[0]*Max

def BFS(v):
    front=0
    rear=-1

    visited[v]=1
    queue[rear+1]=v
    rear+=1

    while front<=rear:
        v=queue[front]
        front+=1
        print(f"{v}",end="")

        for i in range(1,n+1):
            if c[v][i]==1 and visited[i]==0:
                queue[rear+1]=i
                rear+=1
                visited[i]=1

if __name__=="__main__":
    print("Enter the number of vertices in the graph")
    n=int(input())
    print("Enter the cost matrix of the graph:")
    for i in range(1,n+1):
        c[i]=[0]+list(map(int,input().split()))

    for i in range(1,n+1):
        visited[i]=0

    print("Enter the starting vertex:")
    v=int(input())

    print("BFS traversal of the graph is:",end="")
    BFS(v)
```

Output:

```
Enter the number of vertices in the graph
6
Enter the cost matrix of the graph:
0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 0 1
0 1 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
Enter the starting vertex:
1
BFS traversal of the graph is:123456
```

PROGRAM NO -13

OBJECTIVE:

To find the minimum spanning tree of a given graph using Prim‟s algorithm

PROCEDURE:

1. Create: Open Idle, write a program after that save the program with .py extension.
2. Execute: F5

SOURCE CODE:

```
import sys
def minkey(key, mstset,n):
    min_value=sys.maxsize
    for v in range(n):
        if mstset[v]==False and key[v] <min_value:
            min_value=key[v]
            min_index=v
    return min_index


def printMST(parent,c, n):
    totalweight=0
    print("Edge Weight")
    for i in range(1,n):
        print(str(parent[i]+1)+"_"+ str(i+1)+" "+ str(c[i][parent[i]]))
        totalweight+=c[i][parent[i]]
    return totalweight


def primMST(c,n):
    parent=[None]*n
    key=[sys.maxsize]*n
    mstset=[False]*n
    key[0]=0
    parent[0]=-1
    for count in range(n):
        u=minkey(key,mstset,n)
        mstset[u]=True
        for v in range(n):
            if c[u][v]>0 and mstset[v]==False and c[u][v]<key[v]:
                parent[v]=u
                key[v]=c[u][v]
    totalweight=printMST(parent,c, n)
    print("total cost of the minimum spanning tree:"+str(totalweight))

#main code
n=int(input("Enter the number of vertices:"))
c=[]
print("Enter the cost adjacency matrix:")
for i in range(n):
    c.append(list(map(int,input().split())))
primMST(c,n)
```

Output:

```
Enter the number of vertices:5
Enter the cost adjacency matrix:
0 11 9 7 8
11 0 15 14 13
9 15 0 12 14
7 14 12 0 6
8 13 14 6 0
Edge Weight
1_2 11
1_3 9
1_4 7
4_5 6
total cost of the minimum spanning tree:33
```

PROGRAM NO -14(a)

OBJECTIVE:

Write a Program to obtain the topological ordering of vertices in a given digraph.

SOURCE CODE:

```
def main():

    n=int(input("Enter the number of vertices:"))

    count=0

    c=[[0 for _ in range(n)] for _ in range(n)]

    indeg=[0]*n

    flag=[0]*n

    i,j,k=0,0,0


    print("Enter the cost matrix (row by row):")

    for i in range(n):

        row=input().split()

        for j in range(n):

            c[i][j]=int(row[j])


    for i in range(n):

        for j in range(n):

            indeg[i]+=c[j][i]

    print("The topological order is:")

    while count<n:

        for k in range(n):

            if indeg[k]==0 and flag[k]==0:

                print(f"{k+1:3}",end="")

                flag[k]=1
                count +=1

                for i in range(n):

                    if c[k][i]==1:

                        indeg[i]-=1

    return 0

if __name__=="__main__":

    main()
```

Output:

```
Enter the number of vertices:5

Enter the cost matrix (row by row):

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 order is:

  1    2  3  4  5
```

PROGRAM NO -14(b)

OBJECTIVE:

Write a program to compute transitive closure of a given directed graph using warshal’s

SOURCE CODE:

```
def warshalls(c,n):



    for k in range(n):

        for i in range(n):

            for j in range(n):

                if c[i][j] or (c[i][k] and c[k][j]):

                    c[i][j]=1
    print("The transitive closure of the graph is :")
    for i in range(n):
        for j in range(n):
            print(c[i][j],end="")
        print()

def main():
    n=int(input("Enter the number of vertices:"))
    c=[]
    print("Enter the adjacency cost matrix:")
    for i in range(n):
        row=list(map(int, input().split()))
        c.append(row)
    warshalls(c,n)
main()
```

Output:

```
Enter the number of vertices:4
Enter the adjacency cost matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
The transitive closure of the graph is :
1111
1111
0000
1111
```

PROGRAM NO -15

OBJECTIVE:

```
def sum_of_subsets(s, k, r):
    global count, x, w,d,i
    x[k] = 1
    if s + w[k] ==d:
        print("\nSubset %d =" % (count+1) , end=" ")
        for i in range(k + 1):
            if x[i]:
                print("%d" % w[i], end=" ")
    elif s + w[k]+w[k+1]<=d:
        sum_of_subsets(s + w[k], k + 1, r - w[k])
    if s+r - w[k] >= d and s + w[k + 1] <=d:
        x[k] = 0
        sum_of_subsets(s, k+1, r - w[k])
if __name__ == "__main__":
    w = [0] * 10
    x = [0] * 10
    count = 0
    i = 0
    n = int(input("Enter the number of elements: "))
    print("Enter the elements in ascending order: ")
    for i in range(n):
        w[i] =int(input())
    d = int(input("Enter the sum: "))
    sum = 0
    for i in range(n):
        x[i] = 0
        sum += w[i]
    if sum<d  or w[0] > d:
        print("\n No subset possible\n")
    else:
        sum_of_subsets(0,0,sum)
```

Output:

```
Enter the number of elements: 4
Enter the elements in ascending order:
7
11
13
24
Enter the sum: 31

Subset 1 = 7 11 13
Subset 1 = 7 24
```