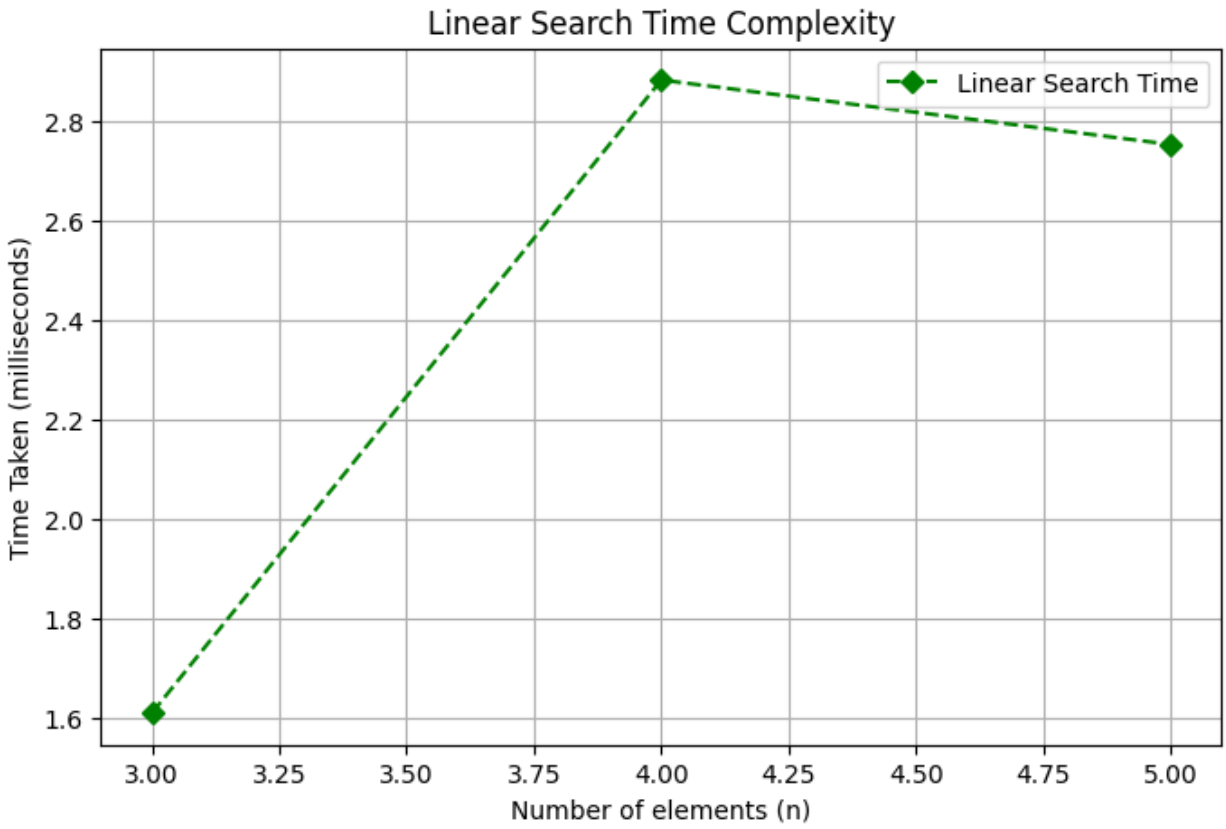
```python
import time
import matplotlib.pyplot as plt

def linear_search(arr, key):
    """Performs a linear search and returns the index of the key, or -1 if not found."""
    for i, val in enumerate(arr):
        if val == key:
            return i
    return -1

def benchmark_linear_search(r):
    """Runs the linear search multiple times and records execution time for different input sizes."""
    results = []

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

        repeat = max(10000 // n, 1000)  # Dynamically reduce repetitions
        start = time.perf_counter()  # More precise timing
        for _ in range(repeat):
            result = linear_search(arr, key)
        end = time.perf_counter()

        if result != -1:
            print(f"Key {key} found at position {result}")
        else:
            print(f"Key {key} not found")

        time_taken = (end - start) * 1000  # Convert to milliseconds
        print(f"Time taken to search a key element = {time_taken:.4f} milliseconds\n")
        results.append((n, time_taken))

    results.sort()  # Sort by number of elements for proper plotting
    return results

def plot_results(results):
    """Plots the results of the linear search timing analysis."""
    if not results:
        print("No data to plot.")
        return

    n_values, times = zip(*results)  # Unpacking tuples
```

```
    plt.figure(figsize=(8, 5))
    plt.plot(n_values, times, 'gD--', label="Linear Search Time")
    plt.xlabel("Number of elements (n)")
    plt.ylabel("Time Taken (milliseconds)")
    plt.title("Linear Search Time Complexity")
    plt.legend()
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    print("Program Name: Linear Search Algorithm Implementation")
    print("Programmer: Thejas R")

    r = int(input("Enter the number of inputs: "))
    results = benchmark_linear_search(r)
    plot_results(results)
```

```
Program Name: Linear Search Algorithm Implementation
Programmer: Venu Madhava M
Enter the number of inputs:  3
Enter the number of elements:  5

Enter the elements of an array:  10 20 30 40 85

Enter the key element to be searched:  85
Key 85 found at position 4
Time taken to search a key element = 2.7543 milliseconds

Enter the number of elements:  3

Enter the elements of an array:  2 4 1

Enter the key element to be searched:  1
Key 1 found at position 2
Time taken to search a key element = 1.6106 milliseconds

Enter the number of elements:  4

Enter the elements of an array:  20 40 30 11

Enter the key element to be searched:  40
Key 40 found at position 1
Time taken to search a key element = 2.8840 milliseconds
```

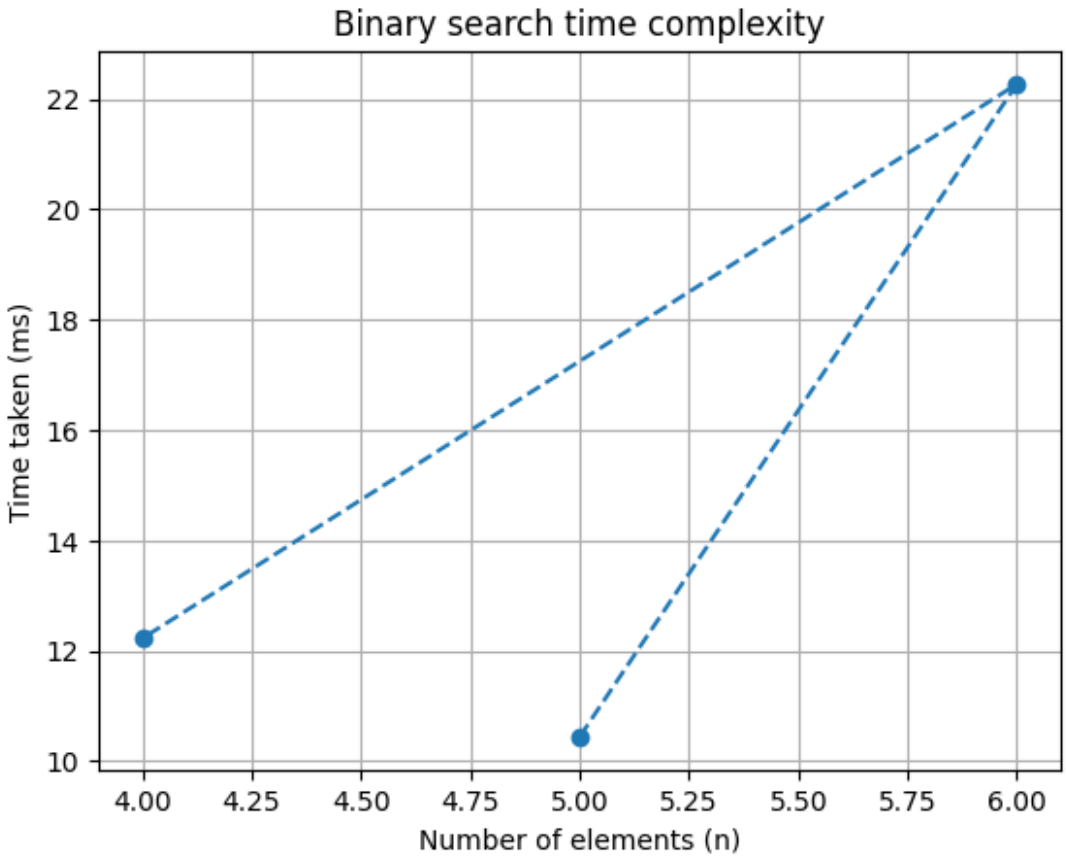



```python
import time
import matplotlib.pyplot as plt

def binary_search(arr, key):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return -1

def measure_time(arr, key):
    repeat = 10000
    start = time.time()
    for _ in range(repeat):
        index = binary_search(arr, key)
    end = time.time()

    if index != -1:
        print(f"Element found at index {index}")
```

```
    else:
        print("Element not found")
    return (end - start) * 1000

def main():
    n_values = []
    times = []
    r = int(input("Enter number of test cases: "))
    for _ in range(r):
        n = int(input("Enter the number of elements: "))
        arr = sorted(map(int, input(f"Enter {n} sorted elements:
").split()))
        key = int(input("Enter the search key/element: "))
        time_taken = measure_time(arr, key)
        print(f"Time taken: {time_taken:.4f} milliseconds")

        # Append values to the lists inside the loop
        n_values.append(n)
        times.append(time_taken)

    plt.figure()
    plt.plot(n_values, times, 'o--')
    plt.xlabel("Number of elements (n)")
    plt.ylabel("Time taken (ms)")
    plt.title("Binary search time complexity")
    plt.grid()
    plt.show()

print("Program Name: Binary search")
print("Programmer Name:Thejas R")

if __name__ == "__main__":
    main()
```

```
Program Name: Binary search
Programmer Name: Venu Madhava.M
Enter number of test cases:  3
Enter the number of elements:  5
Enter 5 sorted elements:  2 3 4 5 6
Enter the search key/element:  5
Element found at index 3
Time taken: 10.4420 milliseconds
Enter the number of elements:  6
Enter 6 sorted elements:  12 1 10 08 33 1000
Enter the search key/element:  999
Element not found
Time taken: 22.2697 milliseconds
Enter the number of elements:  4
Enter 4 sorted elements:  22 11 33 55
Enter the search key/element:  11
```

```
Element found at index 0
Time taken: 12.2256 milliseconds
```




```
def tower (n,source,temp,dest):
    global count
    if n>0:
          tower(n-1,source,dest,temp)
          print(f"Move disk {n} {source}->{dest}")
          count+=1
          tower(n-1,temp,source,dest)
source='S'
temp='T'
dest='D'
count=0
print("Program Name:Towers of Hanoi")
print("Programmer Name:Venu Madhava.M")
n=int(input("Enter the number of Disk:"))
print("Sequence is:")
tower(n,source,temp,dest)
print("The number of moves:",count)
```

```
Program Name:Towers of Hanoi
Programmer Name:Venu Madhava.M
Enter the number of Disk: 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
```

```
#lab-5
def power_brustforce(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)

print("program name:find the value of integer a,n using bruteforce\n\
t\t\t\tand divide and conquer algorithm")
print("programmer name:Venu Madhava.M")
a,n=map(int,input("Enter the value if a and n:").split())
result_burst=power_brustforce(a,n)
result_divide_conquer=power_divide_conquer(a,n)
print("Result using brust force:",result_burst)
print("Result using divide and conquer:",result_divide_conquer)
```

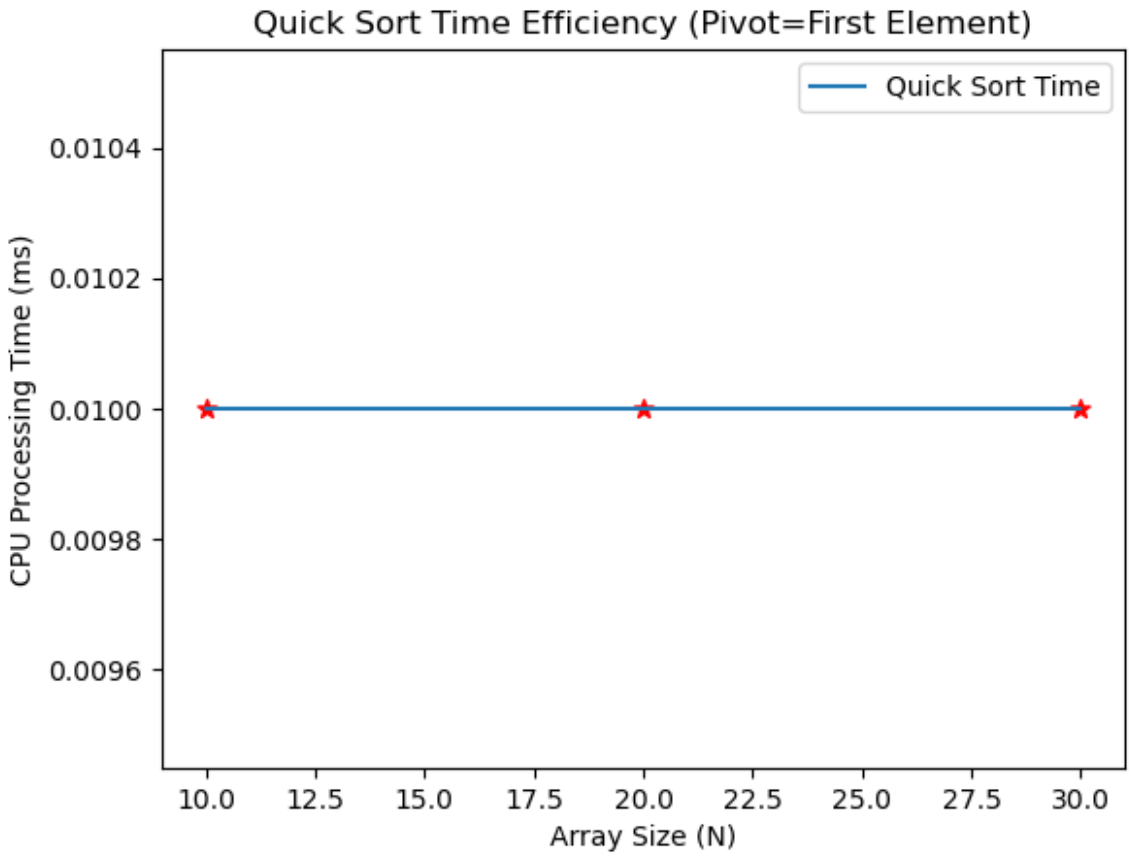
```
program name:find the value of integer a,n using bruteforce
                      and divide and conquer algorithm
programmer name:Venu Madhava.M
Enter the value if a and n: 2 5
Result using brust force: 32
Result using divide and conquer: 32
```

```
import timeit
import random
import matplotlib.pyplot as plt

def Input(array, n):
    for i in range(n):
```

```
        ele = random.randrange(1, 50)
        array.append(ele)

def partition(array, low, high):
    pivot = array[low]
    i = low + 1
    j = high

    while True:
        while i <= j and array[i] <= pivot:
            i += 1
        while i <= j and array[j] > pivot:
            j -= 1
        if i <= j:
            array[i], array[j] = array[j], array[i]
        else:
            break
    array[low], array[j] = array[j], array[low]
    return j

def quicksort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quicksort(array, low, pi - 1)
        quicksort(array, pi + 1, high)

print("program name:quick sort")
print("programmer name:Venu Madhava.M")
N = []
cpu = []
trials = int(input("Enter number of trials:"))
for t in range(trials):
    array = []
    print("\n--------->TRIAL NO:", t + 1)
    n = int(input("Enter number of elements:"))
    Input(array, n)
    print("Random array:", array)

    start = timeit.default_timer()
    quicksort(array, 0, n - 1)
    endtime = timeit.default_timer() - start
    print("Sorted array:", array)
    N.append(n)
    cpu.append(round(endtime * 1000, 2))

print("\nN CPU(microseconds)")
for t in range(trials):
    print(N[t], cpu[t])

plt.plot(N, cpu, label="Quick Sort Time")
```

```
plt.scatter(N, cpu, color="red", marker='*', s=50)
plt.xlabel("Array Size (N)")
plt.ylabel("CPU Processing Time (ms)")
plt.title('Quick Sort Time Efficiency (Pivot=First Element)')
plt.legend()
plt.show()
```

```
program name:quick sort
programmer name:Venu Madhava.M
Enter number of trials:3

--------->TRIAL NO: 1
Enter number of elements:10
Random array: [27, 8, 48, 25, 39, 4, 15, 36, 26, 38]
Sorted array: [4, 8, 15, 25, 26, 27, 36, 38, 39, 48]

--------->TRIAL NO: 2
Enter number of elements:20
Random array: [41, 1, 46, 26, 42, 22, 30, 20, 34, 48, 40, 35, 41, 22,
36, 8, 48, 35, 21, 15]
Sorted array: [1, 8, 15, 20, 21, 22, 22, 26, 30, 34, 35, 35, 36, 40,
41, 41, 42, 46, 48, 48]

--------->TRIAL NO: 3
Enter number of elements:30
Random array: [5, 1, 38, 1, 2, 41, 11, 37, 28, 39, 25, 32, 24, 43, 13,
23, 36, 37, 49, 35, 30, 12, 6, 15, 33, 22, 9, 3, 15, 12]
Sorted array: [1, 1, 2, 3, 5, 6, 9, 11, 12, 12, 13, 15, 15, 22, 23,
24, 25, 28, 30, 32, 33, 35, 36, 37, 37, 38, 39, 41, 43, 49]

N CPU(microseconds)
10 0.01
20 0.01
30 0.01
```




```
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("Programmer Name:Venu Madhava.M")
    print("Program Name:BFS TRAVERSAL")
    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)
```

```
Programmer Name:Venu Madhava.M
Program Name:BFS TRAVERSAL
Enter the Number of Vertices in the graph:
4
Enter the cost Matrix of the Graph:
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
Enter the starting Vertex:
1
BFS TRAVERSAL of the Graph is: 1 2 4 3
```

```
def factorial(n):
    fact=1
    for i in range(2,n+1):
        fact*=i
    return fact
def binomialCoeff_bruteforce(n,k):
    return factorial(n) // (factorial(k)*factorial(n-k))
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):
            if j==0 or j==i:
                C[i][j]=1
            else:
                C[i][j]=C[i-1][j-1]+C[i-1][j]
    return C[n][k]
print("Programmer Name:Venu Madhava.M")
print("Program Name:Binomial Coefficient")
n=int(input("Enter value of n:"))
k=int(input("Enter 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}")
```

```
Programmer Name:Venu Madhava.M
Program Name:Binomial Coefficient
Enter value of n:4
Enter value of k:2
Binomial Coefficient(Brute Force):6
Binomial Coefficient(Dynamic Programming):6
```

```
#lab-9
MAX_CHARS=256
def max_shift(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("Pattern occurs at position=",s)
            s+=(m-badchar[ord(text[s+m])] if (s+m)<n else 1)
        else:
            s+=max_shift(1,j-badchar[ord(text[s+j])])

print("PROGRAM NAME:String Matching using Boyer-moore approach")

print("PROGRAMMER NAME:Venu Madhava.M")
text=input("Enter the text:").rstrip('\n')
pat=input("Enter the pattern:").rstrip('\n')
patternsearch(text,pat)
```

```
PROGRAM NAME:String Matching using Boyer-moore approach
PROGRAMMER NAME:Venu Madhava.M
Enter the text:aabcabbacab
Enter the pattern:abba
Pattern occurs at position= 4
```

```
def computeLPSArray(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,text):
    M=len(pat)
    N=len(text)
    lps=[0]*M
    computeLPSArray(pat,M,lps)
    i=j=0
    while i<N:
        if pat[j]==text[i]:
            i+=1
            j+=1
        if j==M:
            print(f"Found pattern at index {i-j} ")
            j=lps[j-i]
        elif i<N and pat[j]!=text[i]:
            if j!=0:
                j=lps[j-1]
            else:
                i+=1
print("Programmer Name:Venu Madhava.M")
print("Program Name:String Matching using KMP")
text=input("Enter the text:").strip()
pat=input("Enter the pattern:").strip()
KMPSearch(pat,text)
```

```
Programmer Name:Venu Madhava.M
Program Name:String Matching using KMP
Enter the text:ababab
Enter the pattern:abab
Found pattern at index 0
```

```
# lab-11

INF = 99999

def printsolution(v, d):
```

```
    print("\nThe 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()

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)

print("Programmer Name: Venu Madhava.M")
print("Program Name: Floyd's algorithm for weighted graph")
v = int(input("ENTER the number of vertices:"))
c = [[0]*v for _ in range(v)]
print("Enter the cost matrix ROW by ROW (SPACE-SEPARATED):")
print("[Enter 99999 for infinity]")
print("[Enter 0 for cost(i,i)]")

for i in range(v):
    c[i] = list(map(int, input().split()))

floyd(v, c)
```

```
Programmer Name: Venu Madhava.M
Program Name: Floyd's algorithm for weighted graph
ENTER the number of vertices:4
Enter the cost matrix ROW by ROW (SPACE-SEPARATED):
[Enter 99999 for infinity]
[Enter 0 for cost(i,i)]
0 10 99999 20
99999 0 9999 40
80 99999 0 99999
99999 99999 60 0

The following matrix shows the shortest distances between every pair
of vertices:
      0      10      80      20
    180       0     100      40
```

```
     80      90       0     100
    140     150      60       0
```

```
#lab-12  B
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():
    print("Programmer Name: Venu Madhava.M")
    print("Program Name: Topological ordering of vertices [B]
transitive closure")
    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)

if __name__=="__main__"    :
    main()
```

```
Programmer Name: Venu Madhava.M
Program Name: Topological ordering of vertices [B] transitive closure
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:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

```
    #lab-12 A
    def main():
        print("Programmer Name: Venu Madhava.M")
```

```
        print("Program Name: Topological ordering of vertices [A]
digraph.")
        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

    if __name__=="__main__"            :
        main()
```

```
Programmer Name: Venu Madhava.M
Program Name: Topological ordering of vertices [A] digraph.
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
```

```
#lab-13
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))

print("Programmer Name: Venu Madhava.M")
print("Program Name: Prim's Algorithm to find minimum spanning tree.")

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)
```

```
Programmer Name: Venu Madhava.M
Program Name: Prim's Algorithm to find minimum spanning tree.
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
```

```
#lab-14
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


print("Programmer Name: Venu Madhava.M")
print("Program Name: Polynomial using Brute force algorithm and
Horners rule & compare.")
n=int(input("ENTER THE DEGREE OF THE POLYNOMIAL:"))
coef=[0]*(n+1)
print("ENTET 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()
bruteforceresult=bruteforce(coef,n,x)
end=time.time()
timeused1=end-start
print(f"Brute Force Result:{bruteforceresult:.2f},time used:
{timeused1:.6f} seconds")

start=time.time()
hornersruleresult=hornerrule(coef,n,x)
end=time.time()
timeused2=end-start

print(f"Horners Rule Result:{hornersruleresult:.2f},time used:
{timeused2:.6f} seconds")
```

```
Programmer Name: Venu Madhava.M
Program Name: Polynomial using Brute force algorithm and Horners rule
& compare.
```

```
ENTER THE DEGREE OF THE POLYNOMIAL:3
ENTET 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
Horners Rule Result:5.00,time used:0.000000 seconds
```

```python
#lab-15
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=" ")
        count+=1

    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("\nNo subset possible\n")
    else:
        sum_of_subsets (0, 0, sum)
```

```
Enter the number of elements: 5
Enter the elements in ascending order:
1
2
5
6
8
Enter the sum: 9

Subset 1 =  1 2 6
Subset 2 =  1 8
```