ALGORITHM LAB

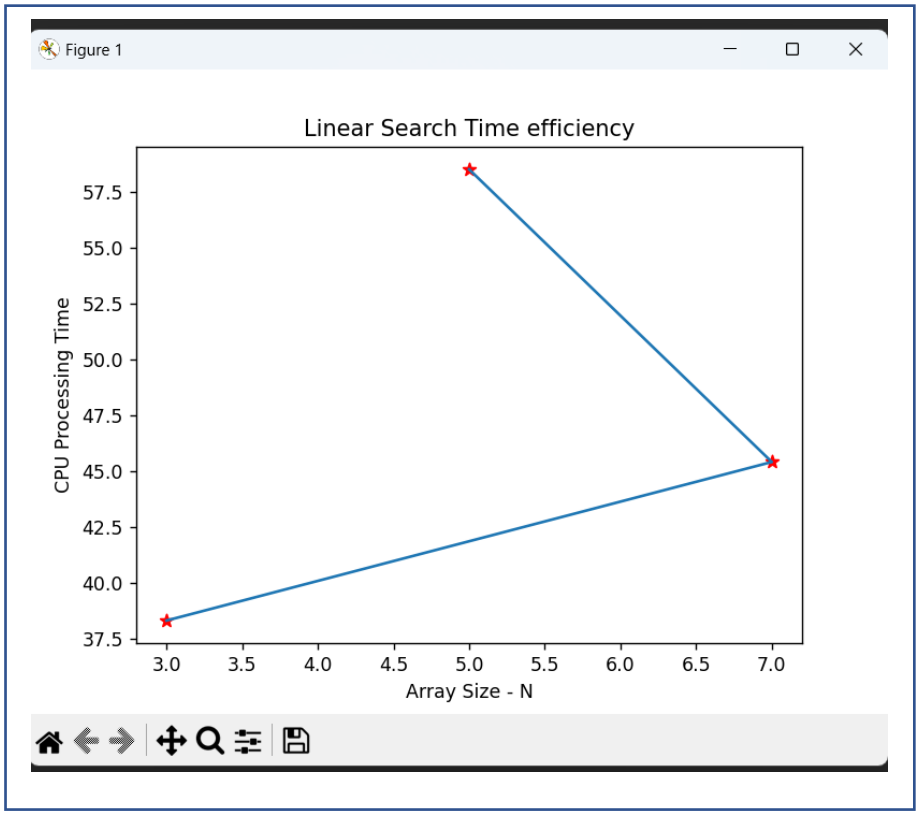
1. **Write a program to implement linear search algorithm. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of time taken versus n.**

```
import timeit
import matplotlib.pyplot as plt

# Input Array elments
def Input(Array, n):
    # iterating till the range
    for i in range(0, n):
        ele = int(input("Arr : "))
        # adding the element
        Array.append(ele)

# Linear Searching
def linear_search(Array, key):
    for x in Array:
        if x == key:
            return True
    return False

# Main Block()
N = []
CPU = []
trail = int(input("Enter no. of trails : "))
for t in range(0, trail):
    Array = []
    print("-----> TRAIL NO : ", t + 1)
    n = int(input("Enter number of elements : "))
    Input(Array, n)
    print(Array)
    key = int(input("Enter key :"))
    start = timeit.default_timer()
    s = linear_search(Array, key)
    print("Element Found = ", s)
    times = timeit.default_timer() - start
    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 Size - N')
# naming the y axis
plt.ylabel('CPU Processing Time')
# giving a title to graph
plt.title('Linear Search Time efficiency')
# function to show the plot
plt.show()
```

Output :

Enter no. of trails : 3
-----> TRAIL NO : 1
Enter number of elements : 5
Arr : 10
Arr : 20
Arr : 30
Arr : 40
Arr : 50
[10, 20, 30, 40, 50]
Enter key :30
Element Found = True
-----> TRAIL NO : 2
Enter number of elements : 7
Arr : 1
Arr : 2
Arr : 3
Arr : 4
Arr : 5
Arr : 6
Arr : 7
[1, 2, 3, 4, 5, 6, 7]
Enter key :2
Element Found = True
-----> TRAIL NO : 3
Enter number of elements : 3
Arr : 100
Arr : 200
Arr : 300
[100, 200, 300]
Enter key :300
Element Found = True
N CPU
5 58.5
7 45.4
3 38.3



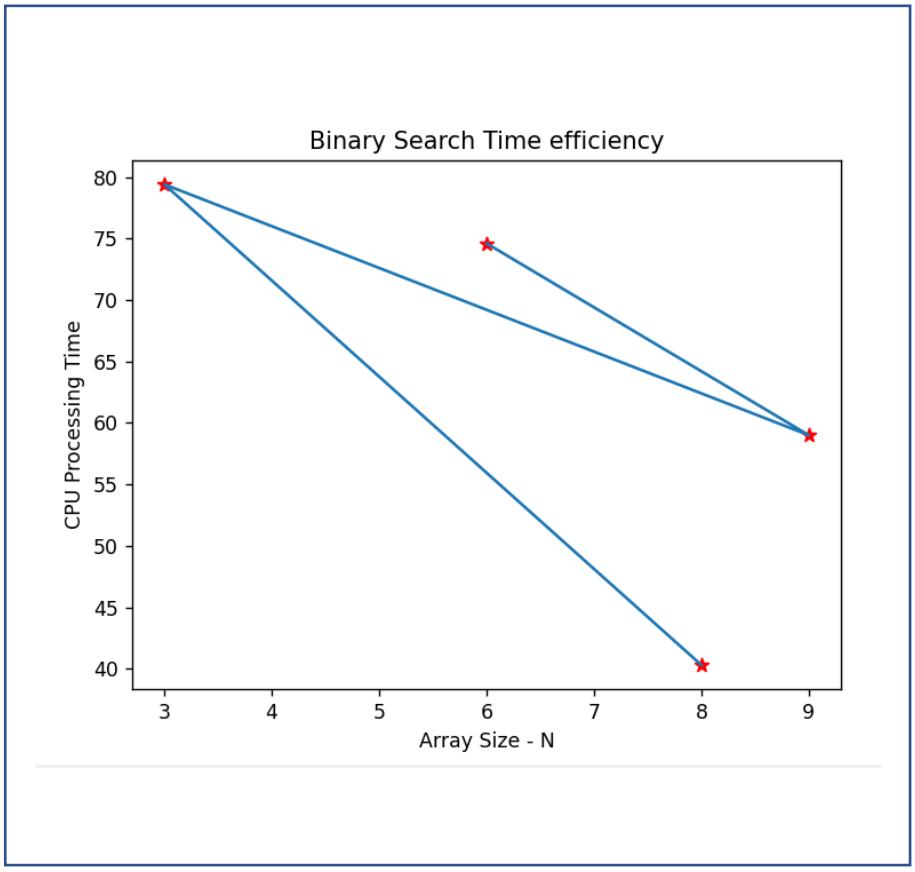
2. **Write a program to implement binary search algorithm. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of time taken versus n.**

```
import timeit
import matplotlib.pyplot as plt

# Input Array elments
def Input(Array, n):
    # iterating till the range
    for i in range(0, n):
        ele = int(input("Arr : "))
        # adding the element
        Array.append(ele)

# Binary Searching
def binary_search(Array, key):
    while len(Array) > 0:
        mid = (len(Array))//2
        if Array[mid] == key:
            return True
        elif Array[mid] < key:
            Array = Array[:mid]
        else:
            Array = Array[mid + 1:]
    return False

# Main Block()
N = []
CPU = []
trail = int(input("Enter no. of trails : "))
for t in range(0, trail):
    Array = []
    print("-----> TRAIL NO : ", t + 1)
    n = int(input("Enter number of elements : "))
    Input(Array, n)
    print(Array)
    key = int(input("Enter key :"))
    start = timeit.default_timer()
    s = binary_search(Array, key)
    print("Element Found = ", s)
    times = timeit.default_timer() - start
    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 Size - N')
# naming the y axis
plt.ylabel('CPU Processing Time')
# giving a title to graph
plt.title('Binary Search Time efficiency')
# function to show the plot
plt.show()
```

Output :




```
Enter no. of trails : 4
-----> TRAIL NO :  1
Enter number of elements : 6
Arr : 10
Arr : 20
Arr : 30
Arr : 40
Arr : 50
Arr : 60
[10, 20, 30, 40, 50, 60]
Enter key :30
Element Found =  False
-----> TRAIL NO :  2
Enter number of elements : 9
Arr : 1
Arr : 2
Arr : 3
Arr : 4
Arr : 5
Arr : 6
Arr : 7
Arr : 8
Arr : 9
[1, 2, 3, 4, 5, 6, 7, 8, 9]
Enter key :1
Element Found =  False
-----> TRAIL NO :  3
Enter number of elements : 3
Arr : 12
Arr : 13
Arr : 14
[12, 13, 14]
Enter key :14
Element Found =  False
```

```
-----> TRAIL NO :  4
Enter number of elements : 8
Arr : 100
Arr : 200
Arr : 300
Arr : 400
Arr : 500
Arr : 600
Arr : 700
Arr : 800
[100, 200, 300, 400, 500, 600, 700, 800]
Enter key :600
Element Found =  False
N  CPU
6 74.6
9 59.0
3 79.4
8 40.3
```

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

```
# Recursive Python function to solve the tower of hanoi

def TowerOfHanoi(n, source, destination, auxiliary):
    if n == 1:
        print("Move disk 1 from source", source, "to destination", destination)
        return
    TowerOfHanoi(n - 1, source, auxiliary, destination)
    print("Move disk", n, "from source", source, "to destination", destination)
    TowerOfHanoi(n - 1, auxiliary, destination, source)


# Main Block
n=int(input("Enter number of disk : "))
TowerOfHanoi(n, 'A', 'B', 'C')
# A, C, B are the name of rods
```

Output :

```
Enter number of disk : 4
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
```

Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B

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 random number generator.**

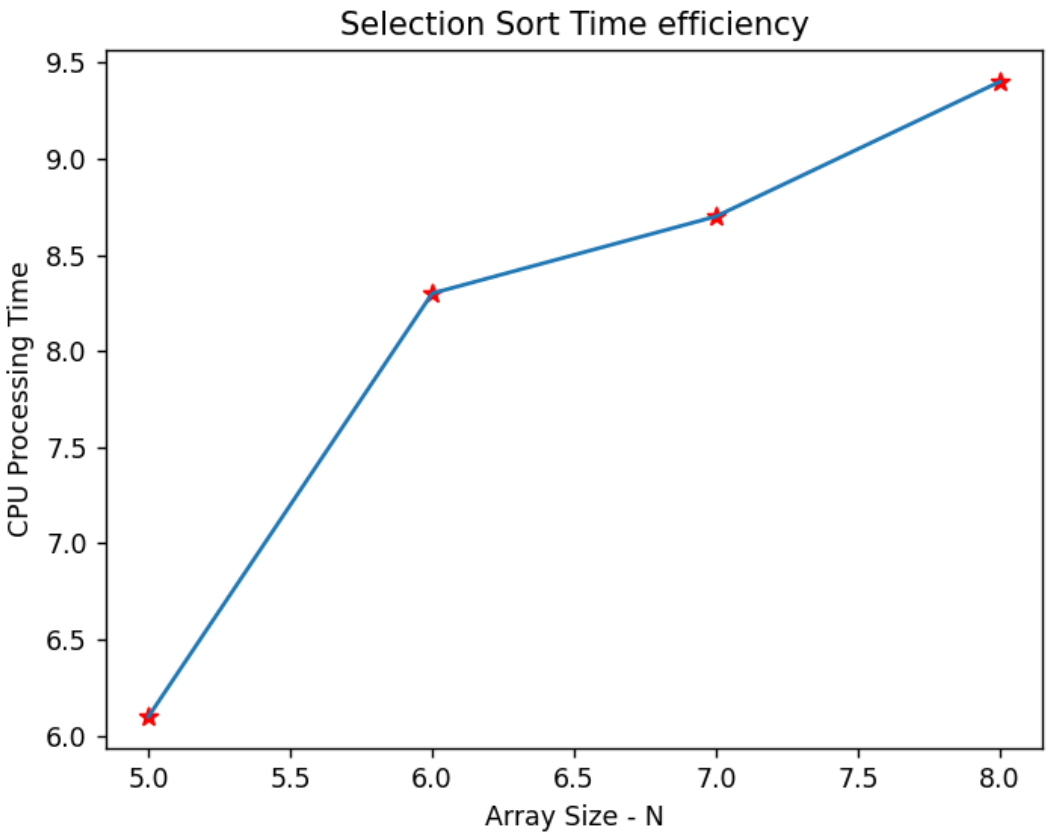
```
import timeit
import random
import matplotlib.pyplot as plt

# 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 no. 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 Size - N')
# naming the y axis
plt.ylabel('CPU Processing Time')
# giving a title to graph
plt.title('Selection Sort Time efficiency')
# function to show the plot
plt.show()
```

Output :

```
Enter no. of trails : 4
-----> TRAIL NO :  1
Enter number of elements : 5
Sorted Array :
[14, 17, 23, 26, 37]
-----> TRAIL NO :  2
Enter number of elements : 6
Sorted Array :
[18, 19, 20, 28, 38, 48]
-----> TRAIL NO :  3
Enter number of elements : 7
Sorted Array :
[13, 24, 29, 35, 35, 47, 49]
```

-----> TRAIL NO : 4
Enter number of elements : 8
Sorted Array :
[5, 6, 7, 9, 10, 21, 25, 33]
N CPU
5 6.1
6 8.3
7 8.7
8 9.4



5. Write a program to find a^n using (a) Brute-force based algorithm
(b) Divide and conquer based algorithm

 # Brute force method

#A simple solution to calculate pow(a, n) would multiply a exactly n times. We can do that by using a simple for loop

```
def bpower(a, n):
    pow = 1
    for i in range(n):
        pow = pow * a
    return pow
```

 # Divide and Conquer method

#The problem can be recursively defined by:
*# dpower(x, n) = dpower(x, n / 2) * dpower(x, n / 2); // if n is even*
*# dpower(x, n) = x * dpower(x, n / 2) * dpower(x, n / 2); // if n is odd*

```
def dpower(x, y):
    if (y == 0):
        return 1
    elif (int(y % 2) == 0):
        return (dpower(x, int(y / 2)) *
                dpower(x, int(y / 2)))
    else:
        return (x * dpower(x, int(y / 2)) *
                dpower(x, int(y / 2)))
```

 # Main block

```
a=int(input("Enter a :"))
n=int(input("Enter n :"))
print("Brute Force method a^n : ",bpower(a,n))
print("Divide and Conquer a^n : ",dpower(a,n))
```

Output

```
Enter a :2
Enter n :3
Brute Force method a^n :  8
Divide and Conquer a^n :  8
```

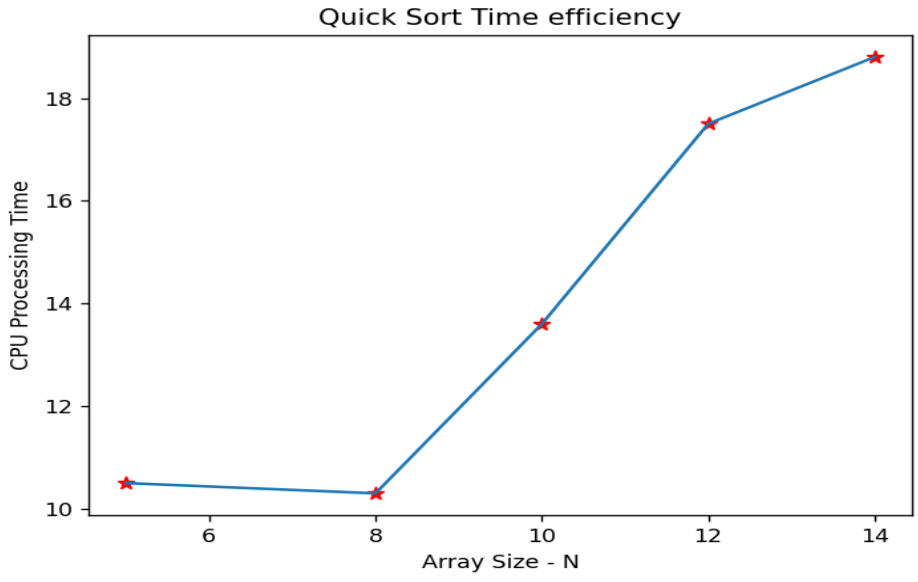
6. **Write a program to sort a given set of numbers using quick sort algorithm. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.**

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

# Input Array elments
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:
      # increment
      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 no. 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 Size - N')
# naming the y axis
plt.ylabel('CPU Processing Time')
# giving a title to graph
plt.title('Quick Sort Time efficiency')
# function to show the plot
plt.show()
```



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

 # Brute force method

#The value of C(n, k) can be recursively calculated using the following standard formula for Binomial Coefficients.

 # C(n, k) = C(n-1, k-1) + C(n-1, k)

 # C(n, 0) = C(n, n) = 1

```
def binomialCoeff_BF(n, k):
    if k > n:
        return 0
    if k == 0 or k == n:
        return 1

    # Recursive Call
    return binomialCoeff_BF(n - 1, k - 1) + binomialCoeff_BF(n - 1, k)
```

 # Divide and Conquer method

#re-computations of the same subproblems can be avoided by constructing a temporary 2D-array C[][] in a bottom-up manner.

 # uses Overlapping Subproblems concept

```
def binomialCoef_DC(n, k):
    C = [[0 for x in range(k + 1)] for x in range(n + 1)]

    # Calculate value of Binomial
    # Coefficient in bottom up manner
    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

            # Calculate value using
            # previously stored values
            else:
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

    return C[n][k]
```

 # Main block

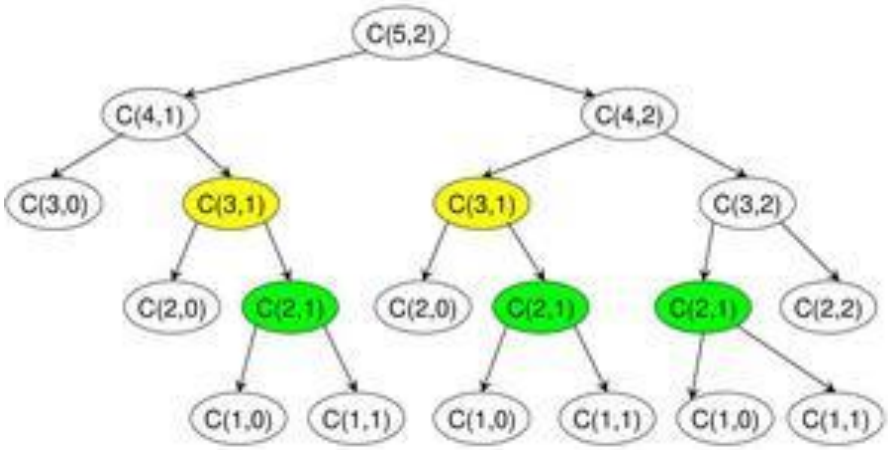
```
n=int(input("Enter n :"))
k=int(input("Enter k :"))
print("Brute Force method n^k : ",binomialCoeff_BF(n, k))
print("Divide and Conquer n^k : ",binomialCoef_DC(n, k))
```

Output

```
Enter n :5
Enter k :2
Brute Force method n^k :  10
Divide and Conquer n^k :  10
```

Overlapping Subproblems

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree for n = 5 and k = 2. The function C(3, 1) is called two times. For large values of n, there will be many common subproblems.



Since the same subproblems are called again, this problem has the Overlapping Subproblems property. So the re-computations of the same subproblems can be avoided by constructing a temporary 2D-array C[][] in a bottom-up manner. Above is Dynamic Programming-based implementation.

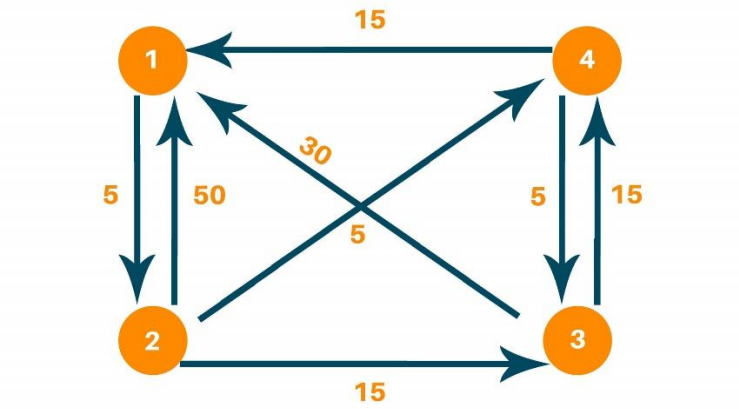
---

8. **Write a program to implement Flyod's algorithm and find the lengths of the shortest paths from every pairs of vertices in a weighted graph.**

```
# Number of vertices
nV = 4
INF = 999

# Algorithm
def floyd(G):
    dist = list(map(lambda p: list(map(lambda q: q, p)), G))

    # Adding vertices individually
    for r in range(nV):
        for p in range(nV):
            for q in range(nV):
                dist[p][q] = min(dist[p][q], dist[p][r] + dist[r][q])
    sol(dist)

# Printing the output
def sol(dist):
    for p in range(nV):
        for q in range(nV):
            if(dist[p][q] == INF):
                print("INF", end=" ")
            else:
                print(dist[p][q], end="  ")
        print(" ")
#Input
G = [[0, 5, INF, INF],
        [50, 0, 15, 5],
        [30, INF, 0, 15],
        [15, INF, 5, 0]]
floyd(G)
```

Output :

```
0  5  15  10
20  0  10  5
30  35  0  15
15  20  5  0
```



Creating matrix D_0 contains the distance between each node with '0' as an intermediate node.

$$D_0 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

Updating matrix D_1 which contains the distance between each node with '1' as an intermediate node. Update distance if minimum distance value smaller than existing distance value found.

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_1[3, 2] = \min(D_{1-1}[3, 2], D_{1-1} [3, 1] + D_{1-1} [1 , 2])$
$D_1[4, 2] = \min(\text{Infinity}, 35)$

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_1[4, 2] = \min(D_{1-1}[4, 2], D_{1-1} [4, 1] + D_{1-1} [1 , 2])$
$D_1[4, 2] = \min(\text{Infinity}, 20)$
Here distance (3, 2) is updated from infinity to 35, and distance (4, 2) is updated from infinity to 20 as shown below.

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Updating matrix D_2 contains the distance between two nodes with '2' as an intermediate node.

$D_2 [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_2[1, 3] = \min(D_{2-1}[1, 3], D_{2-1} [1, 2] + D_{2-1} [2 , 3])$
$D_2[1, 3] = \min(\text{Infinity}, 20)$

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_2[1, 4] = \min(D_{2-1}[1, 4], D_{2-1} [1, 2] + D_{2-1} [2 , 4])$
$D_2[1, 4] = \min(\text{Infinity}, 10)$
Update distance if minimum distance value smaller than existing distance value found. Here distance (1, 3) is updated from infinity to 20, and distance (1, 4) is updated from infinity to 10.

$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Updating matrix D_3 contains the distance between two nodes with '3' as an intermediate node.

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_3[2, 1] = \min(D_{3-1}[2, 1], D_{3-1} [2, 3] + D_{3-1} [3 , 1])$
$D_3[2, 1] = \min(50, 45)$
Update distance if minimum distance value smaller than existing distance value found. Here distance (2, 1) is updated from 50 to 45

$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Updating matrix D_4 contains the distance between two nodes with '4' as an intermediate node. Update distance if minimum distance value smaller than existing distance value found.

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_4[1, 3] = \min(D_{4-1}[1, 3], D_{4-1} [1, 4] + D_{4-1} [4 , 3])$
$D_4[1, 3] = \min(20, 15)$

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_4[2, 1] = \min(D_{4-1}[2, 1], D_{4-1} [2, 4] + D_{4-1} [4 , 1])$
$D_4[1, 3] = \min(45, 20)$

$D_k [i , j] = \min (D_{k-1} [i , j], D_{k-1} [i, k] + D_{k-1} [k , j])$
$D_4[2, 3] = \min(D_{4-1}[2, 3], D_{4-1} [2, 4] + D_{4-1} [4 , 3])$
$D_4[2, 3] = \min(15, 10)$
Here distance (1, 3) is updated from 20 to 15; distance (2, 1) is updated from 45 to 20, and distance (2, 3) is updated from 15 to 10

$$D_4 = \begin{pmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

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

```
import timeit

def polynomial_BF(poly,x,n):
    # Declaring the result
    result = 0
    # Running a for loop to traverse through the list
    for i in range(n):

        # Declaring the variable Sum
        Sum = poly[i]

        # Running a for loop to multiply x (n-i-1)
        # times to the current coefficient
        for j in range(n - i - 1):
            Sum = Sum * x

        # Adding the sum to the result
        result = result + Sum

    # Printing the result
    print("Value of polynomial 2x3 - 6x2 + 2x - 1 for x = 3 using [BRUTE FORCE method ]
:",result)


def horner(poly, x, n):
    # Initialize result
    res = poly[0]

    # Evaluate value of polynomial
    # using Horner's method
    for i in range(1, n):
        res = res * x + poly[i]
    print("Value of polynomial 2x3 - 6x2 + 2x - 1 for x = 3 using [HORNER method ] :",
res)
```

```
#Main block
# 2x3 - 6x2 + 2x - 1 for x = 3
poly = [2, -6, 2, -1]
x = 3
n = len(poly)
start1 = timeit.default_timer()
polynomial_BF(poly,x,n)
t1 = timeit.default_timer() - start1

start2 = timeit.default_timer()
horner(poly,x,n)
t2 = timeit.default_timer() - start2

print("Time complexity of Brute force method O(n2) : ",t1)
print("Time complexity of Horner method O(n) : ",t2)
```

Output :

```
Value of polynomial 2x3 - 6x2 + 2x - 1 for x = 3 using [BRUTE FORCE method ] : 5
Value of polynomial 2x3 - 6x2 + 2x - 1 for x = 3 using [HORNER method ] : 5
Time complexity of Brute force method O(n2) :  2.5799999999992496e-05
Time complexity of Horner method O(n) :  8.299999999988872e-06
```

Brute Force method : In this approach, the following methodology will be followed. This is the most naive approach to do such questions.

- First coefficient $\mathbf{c_n}$ will be multiplied with $\mathbf{x^n}$
- Then coefficient $\mathbf{c_{n-1}}$ will be multiplied with $\mathbf{x^{n-1}}$
- The results produced in the above two steps will be added
- This will go on till all the coefficient are covered.

Horner's method can be used to evaluate polynomial in O(n) time. To understand the method, let us consider the example of $2x^3 – 6x^2 + 2x – 1$. The polynomial can be evaluated as $((2x – 6)x + 2)x – 1$. The idea is to initialize result as the coefficient of x_n which is 2 in this case, repeatedly multiply the result with x and add the next coefficient to result. Finally, return the result.

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

Note :

A string searching algorithm based upon Boyer-Moore string searching, which is considered one of the most efficient string searching algorithms. Boyer-Moore-Horspool only uses the bad-suffix window for matching and is therefore simpler to implement and faster than normal BM. This algorithm is more efficient than KMP and has low overhead to implement. It is one of the few string searching algorithms that balances memory consumption and speed very well. There have been many comparisons and studies on this and other string searching algorithms in the field and charts can be found which prove the usefulness of this algorithm. According to Moore himself, this algorithm gets faster the larger the pattern.

```
def BoyerMooreHorspool(pattern, text):
    m = len(pattern)
    n = len(text)
    if m > n: return -1
    skip = []
    for k in range(256): skip.append(m)
    for k in range(m - 1): skip[ord(pattern[k])] = m - k - 1
    skip = tuple(skip)
    k = m - 1
    while k < n:
        j = m - 1; i = k
        while j >= 0 and text[i] == pattern[j]:
            j -= 1; i -= 1
        if j == -1: return i + 1
        k += skip[ord(text[k])]
    return -1

#Main block
if __name__ == '__main__':
    text = "this is the string to search in"
    pattern = "the"
    s = BoyerMooreHorspool(pattern, text)
    print ('Text:',text)
    print ('Pattern:',pattern)
    if s > -1:
        print ('Pattern \"' + pattern + '\" found at position',s)
```

Output :

Text: this is the string to search in

Pattern: the

Pattern "the" found at position 8

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

```
def KMP_String(pattern, text):
    a = len(text)
    b = len(pattern)

    prefix_arr = get_prefix_arr(pattern, b)

    initial_point = []

    m = 0
    n = 0

    while m != a:

        if text[m] == pattern[n]:
            m += 1
            n += 1

        else:
            n = prefix_arr[n - 1]

        if n == b:
            initial_point.append(m - n)
            n = prefix_arr[n - 1]
        elif n == 0:
            m += 1

    return initial_point


def get_prefix_arr(pattern, b):
    prefix_arr = [0] * b
    n = 0
    m = 1

    while m != b:
        if pattern[m] == pattern[n]:
            n += 1
            prefix_arr[m] = n
            m += 1
        elif n != 0:
            n = prefix_arr[n - 1]
```

```
        else:
            prefix_arr[m] = 0
            m += 1

    return prefix_arr

# Main module
string = "ABABDABACDABABCABABCABAB"
pat = "ABABCABAB"

initial_index = KMP_String(pat, string)
print("String :",string)
print("Pattern :",pat)
for i in initial_index:
    print('pattern is found in the string at index number', i)
```

Output :

```
String : ABABDABACDABABCABABCABAB
Pattern : ABABCABAB
pattern is found in the string at index number 10
pattern is found in the string at index number 15
```

String = “ABABDABACDABABCABABCABAB”

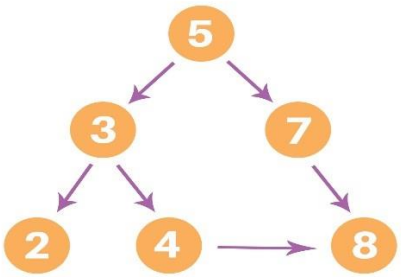
Pattern = “ABABCABAB”

Here the pattern first checks with the string. At index 5 there will be a mismatch. Now the pattern shifts one position. That means, now the pattern starts checking from index 1. Here KMP String Matching algorithms optimizes over Normal String Matching. According to Normal String Matching algorithm, the pattern starts checking from string ‘A’, that means index 0 in pattern to the end of the pattern. Even though similar strings are present in both the pattern and in the given string from index 0 to index 4, Normal String Matching algorithm checks from the starting of the pattern.

But, KMP String Matching algorithm starts checking from index 5 of letter ‘C’ because we know first four characters will anyway match, we skipped matching first four characters. This is how optimization is done in this algorithm.

12.**Write a program to implement BFS traversal algorithm.**

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
```



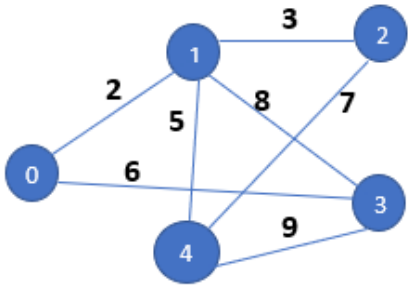

```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Main module
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

Output :

```
Following is the Breadth-First Search
        5   3 7 2 4 8
```

13. **Write a program to find minimum spanning tree of a given graph using Prim's Algorithm.**

```
# Prim's Algorithm in Python

INF = 9999999
V = 5
G =[[0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 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]):
                    # not in selected and there is an edge
                    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 :

Edge : Weight

0-1 : 2
1-2 : 3
1-4 : 5
0-3 : 6

14(a). Write a program to obtain the topological ordering of vertices in a given digraph.

```
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)
# Main block
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

```
Topological Sort:
[1, 3, 2, 5, 4, 6]
```

14(b). Write a program to compute transitive closure of a given directed graph using Warshall's algorithm.

```
# Python program for transitive closure using Floyd Warshall Algorithm
# Complexity : O(V^3)

from collections import defaultdict
# Class to represent a graph
class Graph:

    def __init__(self, vertices):
        self.V = vertices

    # A utility function to print the solution
    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):
                if (i == j):
                    print("%7d\t" % (1), end=" ")
                else:
                    print("%7d\t" % (reach[i][j]), end=" ")
            print()

    # Prints transitive closure of graph[][] using Floyd Warshall algorithm
    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)

#Main module
g = Graph(4)
graph = [[1, 1, 0, 1],
         [0, 1, 1, 0],
         [0, 0, 1, 1],
         [0, 0, 0, 1]]

# Print the solution
g.transitiveClosure(graph)
```

Output :

Following matrix transitive closure of the given graph

1	**1**	**1**	**1**
0	**1**	**1**	**1**
0	**0**	**1**	**1**
0	**0**	**0**	**1**

15.Write a program to find subset of a given set S={s1,s2,….sn} of n positive integers whose sum is equal to given positive integer d. For example if S={1,2,5,6,8} and d=9 then two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if given problem doesn't have solution.

```
 from itertools import combinations
def Input(S, n):
    # iterating till the range
    for i in range(0, n):
        ele = int(input("Arr : "))
        # adding the element
        S.append(ele)

def sub_set_sum(size, S, sub_set_sum):
  count=0
  for i in range(size+1):
    for my_sub_set in combinations(S, i):

      if sum(my_sub_set) == d:
        print(list(my_sub_set))
        count=count+1
  if(count==0) :
    print("Subset Not found for the given d=",d)


#Main module

S = []
n = int(input("Enter size :"))
Input(S, n)
print(S)
d = int(input("Enter sum d :"))
print("The result is :")
sub_set_sum(n, S, d)
```

Output:

```
Enter size :5
Arr : 1
Arr : 2
Arr : 5
Arr : 6
Arr : 8
[1, 2, 5, 6, 8]
Enter sum d :9
The result is :
[1, 8]
[1, 2, 6]

Enter size :3
Arr : 1
Arr : 2
Arr : 3
[1, 2, 3]
Enter sum d :10
The result is :
Subset Not found for the given d= 10
```