**1. Write a program that finds the closest pair of points**

import math

# Input: list of points

points = [(1, 2), (4, 5), (7, 8), (3, 1)]

# Initialize minimum distance and closest pair

min\_distance = float('inf')

closest\_pair = None

# Brute force approach to find the closest pair of points

n = len(points)

for i in range(n):

for j in range(i + 1, n):

# Calculate the Euclidean distance between points[i] and points[j]

dist = math.sqrt((points[i][0] - points[j][0])\*\*2 + (points[i][1] - points[j][1])\*\*2)

# Update minimum distance and closest pair if a smaller distance is found

if dist < min\_distance:

min\_distance = dist

closest\_pair = (points[i], points[j])

# Output results

print(f"Closest pair: {closest\_pair[0]} - {closest\_pair[1]}")

print(f"Minimum distance: {min\_distance}")

O/P: Closest pair: (1, 2) - (3, 1)

Minimum distance: 1.4142135623730951

**2.Convex Hull Problem (Brute Force)**

import math

def calculate\_distance(point1, point2):

"""Calculate the Euclidean distance between two points."""

return math.sqrt((point1[0] - point2[0])\*\*2 + (point1[1] - point2[1])\*\*2)

def find\_closest\_pair(points):

"""Find the closest pair of points using brute force."""

min\_distance = float('inf')

closest\_pair = None

n = len(points)

for i in range(n):

for j in range(i + 1, n):

dist = calculate\_distance(points[i], points[j])

if dist < min\_distance:

min\_distance = dist

closest\_pair = (points[i], points[j])

return closest\_pair, min\_distance

points = [(1, 2), (4, 5), (7, 8), (3, 1)]

closest\_pair, min\_distance = find\_closest\_pair(points)

print(f"Closest pair: {closest\_pair[0]} - {closest\_pair[1]}")

print(f"Minimum distance: {min\_distance}")

**3 .program for to find the shortest path pair of points in the given set using brute force approch**

import math

def euclidean\_distance(point1, point2):

"""Calculate the Euclidean distance between two points."""

return math.sqrt((point1[0] - point2[0]) \*\* 2 + (point1[1] - point2[1]) \*\* 2)

def closest\_pair(points):

"""Find the closest pair of points using the brute force method."""

min\_distance = float('inf')

closest\_points = (None, None)

for i in range(len(points)):

for j in range(i + 1, len(points)):

distance = euclidean\_distance(points[i], points[j])

if distance < min\_distance:

min\_distance = distance

closest\_points = (points[i], points[j])

return closest\_points, min\_distance

# Sample set of points

points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)]

closest\_points, distance = closest\_pair(points)

print(f"The closest pair of points is: {closest\_points} with a distance of {distance:.2f}")

**4. to slove travelling sales man person by the exhaustive search**

import itertools

import math

def distance(city1, city2):

return math.sqrt((city1[0] - city2[0]) \*\* 2 + (city1[1] - city2[1]) \*\* 2)

def tsp(cities):

start\_city = cities[0]

shortest\_distance = float('inf')

shortest\_path = []

for perm in itertools.permutations(cities[1:]):

current\_path = [start\_city] + list(perm) + [start\_city]

current\_distance = sum(distance(current\_path[i], current\_path[i + 1]) for i in range(len(current\_path) - 1))

if current\_distance < shortest\_distance:

shortest\_distance = current\_distance

shortest\_path = current\_path

return shortest\_distance, shortest\_path

# Test Cases

cities\_case1 = [(1, 2), (4, 5), (7, 1), (3, 6)]

cities\_case2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]

# Running Test Cases

result1 = tsp(cities\_case1)

result2 = tsp(cities\_case2)

print("Test Case 1:")

print("Shortest Distance:", result1[0])

print("Shortest Path:", result1[1])

print("\nTest Case 2:")

print("Shortest Distance:", result2[0])

print("Shortest Path:", result2[1])

**5. Exhaustive Search with Permutations**

import itertools

def total\_cost(assignment, cost\_matrix):

total = 0

for worker, task in enumerate(assignment):

total += cost\_matrix[worker][task]

return total

def assignment\_problem(cost\_matrix):

num\_workers = len(cost\_matrix)

workers = list(range(num\_workers))

optimal\_assignment = None

min\_cost = float('inf')

for perm in itertools.permutations(workers):

current\_cost = total\_cost(perm, cost\_matrix)

if current\_cost < min\_cost:

min\_cost = current\_cost

optimal\_assignment = perm

return optimal\_assignment, min\_cost

# Test Case 1

cost\_matrix\_1 = [[3, 10, 7],

[8, 5, 12],

[4, 6, 9]]

optimal\_assignment\_1, min\_cost\_1 = assignment\_problem(cost\_matrix\_1)

print(f"Optimal Assignment: {optimal\_assignment\_1}, Minimum Cost: {min\_cost\_1}")

# Test Case 2

cost\_matrix\_2 = [[15, 9, 4],

[8, 7, 18],

[6, 12, 11]]

optimal\_assignment\_2, min\_cost\_2 = assignment\_problem(cost\_matrix\_2)

print(f"Optimal Assignment: {optimal\_assignment\_2}, Minimum Cost: {min\_cost\_2}")

**6. # 6.0-1 Knapsack Problem Solutions**

def total\_value(items, values):

return sum(values[i] for i in items)

def is\_feasible(items, weights, capacity):

total\_weight = sum(weights[i] for i in items)

return total\_weight <= capacity

def knapsack(weights, values, capacity):

n = len(weights)

best\_value = 0

best\_combination = []

for i in range(1 << n): # Iterate through all possible combinations

selected\_items = [j for j in range(n) if (i & (1 << j)) > 0]

if is\_feasible(selected\_items, weights, capacity):

current\_value = total\_value(selected\_items, values)

if current\_value > best\_value:

best\_value = current\_value

best\_combination = selected\_items

return best\_combination, best\_value

# Test Case 1

weights1 = [2, 3, 1]

values1 = [4, 5, 3]

capacity1 = 4

result1 = knapsack(weights1, values1, capacity1)

print(f"Optimal Selection: {result1[0]}, Total Value: {result1[1]}")

# Test Case 2

weights2 = [1, 2, 3, 4]

values2 = [2, 4, 6, 3]

capacity2 = 6

result2 = knapsack(weights2, values2, capacity2)

print(f"Optimal Selection: {result2[0]}, Total Value: {result2[1]}")