main.py

Share Run

```python
import itertools
import math

points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = min(((p1, p2), math.dist(p1, p2)) for p1, p2 in
    itertools.combinations(points, 2))
print(f"Closest pair: {closest_pair} Minimum distance: {min_distance}")
```

Output

```
Closest pair: ((1, 2), (3, 1)) Minimum distance: 2.23606797749979

=== Code Execution Successful ===
```

main.py

Share Run

```python
import itertools

def euclidean_distance(p1, p2):
    return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)**0.5

def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    for pair in itertools.combinations(points, 2):
        distance = euclidean_distance(pair[0], pair[1])
        if distance < min_distance:
            min_distance = distance
            closest_pair = pair
    return closest_pair
points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)]
result = closest_pair_brute_force(points)
print(result)
```

Output

```
((9, 3.5), (7.5, 4.5))

=== Code Execution Successful ===
```

main.py

Share Run

```python
from itertools import combinations

def orientation(p, q, r):
    val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
    if val == 0:
        return 0
    return 1 if val > 0 else -1

def convex_hull(points):
    n = len(points)
    if n < 3:
        return points

    hull = []
    for p, q in combinations(points, 2):
        side = 0
        for r in points:
            side = max(side, orientation(p, q, r))
        if side == 0:
            hull.extend([p, q])

    return sorted(list(set(hull)))
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
convex_hull_points = convex_hull(points)
print(convex_hull_points)
```

Output

```
[(0, 0), (4, 6)]

=== Code Execution Successful ===
```

main.py

Share Run

```python
import itertools

def distance(city1, city2):
    return ((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)**0.5

def tsp(cities):
    min_distance = float('inf')
    shortest_path = None

    for perm in itertools.permutations(cities[1:]):
        total_distance = 0
        path = [cities[0]] + list(perm) + [cities[0]]

        for i in range(len(path) - 1):
            total_distance += distance(path[i], path[i + 1])

        if total_distance < min_distance:
            min_distance = total_distance
            shortest_path = path

    return min_distance, shortest_path
cities = [(0, 0), (1, 2), (3, 1), (5, 3)]
min_dist, shortest_route = tsp(cities)
print(f"Minimum Distance: {min_dist}")
print(f"Shortest Route: {shortest_route}")
```

Output

```
Minimum Distance: 12.349878388032021
Shortest Route: [(0, 0), (1, 2), (5, 3), (3, 1), (0, 0)]

=== Code Execution Successful ===
```

main.py

```python
from itertools import permutations

def total_cost(assignment, cost_matrix):
    return sum(cost_matrix[worker][task] for worker, task in assignment)

def assignment_problem(cost_matrix):
    workers = range(len(cost_matrix))
    min_cost = float('inf')
    optimal_assignment = None

    for perm in permutations(workers):
        assignment = list(zip(perm, range(len(cost_matrix))))
        cost = total_cost(assignment, cost_matrix)

        if cost < min_cost:
            min_cost = cost
            optimal_assignment = assignment

    return optimal_assignment, min_cost
cost_matrix_1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
cost_matrix_2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]

optimal_assignment_1, total_cost_1 = assignment_problem(cost_matrix_1)
optimal_assignment_2, total_cost_2 = assignment_problem(cost_matrix_2)
print("Test Case 1:")
print("Optimal Assignment:", [(f"worker {worker + 1}", f"task {task + 1}")
    for worker, task in optimal_assignment_1])
print("Total Cost:", total_cost_1)
```

Output

```
Test Case 1:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'), ('worker 1',
    'task 3')]
Total Cost: 16

Test Case 2:
Optimal Assignment: [('worker 3', 'task 1'), ('worker 2', 'task 2'), ('worker 1',
    'task 3')]
Total Cost: 17

=== Code Execution Successful ===
```

main.py

```python
def total_value(items, values):
    return sum(values[i] for i in items)

def is_feasible(items, weights, capacity):
    return sum(weights[i] for i in items) <= capacity
items = 3
weights = [2, 3, 1]
values = [4, 5, 3]
capacity = 4

max_value = 0
optimal_selection = []

for i in range(1 << items):
    selected_items = [j for j in range(items) if i & (1 << j)]
    if is_feasible(selected_items, weights, capacity):
        total = total_value(selected_items, values)
        if total > max_value:
            max_value = total
            optimal_selection = selected_items

print("Optimal Selection:", optimal_selection)
print("Total Value:", max_value)
```

Share Run

Output

```
Optimal Selection: [1, 2]
Total Value: 8

=== Code Execution Successful ===
```