1.

```python
import math

def euclidean_distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)

def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    n = len(points)

    for i in range(n):
        for j in range(i + 1, n):
            distance = euclidean_distance(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])

    return closest_pair, min_distance

points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = closest_pair_brute_force(points)
print(f"Closest pair: {closest_pair[0]} - {closest_pair[1]}")
print(f"Minimum distance: {min_distance}")
```

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

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

2.

```python
def is_counter_clockwise(p, q, r):
    return (q[1] - p[1]) * (r[0] - q[0]) > (q[0] - p[0]) * (r[1] - q[1])
def convex_hull_brute_force(points):
    hull = []
    n = len(points)
    for i in range(n):
        for j in range(i + 1, n):
            left = right = False
            for k in range(n):
                if k != i and k != j:
                    if is_counter_clockwise(points[i], points[j], points[k]):
                        left = True
                    else:
                        right = True
                if left and right:
                    break
            if not (left and right):
                if points[i] not in hull:
                    hull.append(points[i])
                if points[j] not in hull:
                    hull.append(points[j])

    return sorted(hull)
points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)]
hull = convex_hull_brute_force(points)
print("Convex Hull:", hull)
```

```
Convex Hull: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]

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

3.

```python
def is_counter_clockwise(p, q, r):
    return (q[1] - p[1]) * (r[0] - q[0]) > (q[0] - p[0]) * (r[1] - q[1])
def convex_hull_brute_force(points):
    hull = []
    n = len(points)

    for i in range(n):
        for j in range(i + 1, n):
            left = right = False
            for k in range(n):
                if k != i and k != j:
                    if is_counter_clockwise(points[i], points[j], points[k]):
                        left = True
                    else:
                        right = True
                if left and right:
                    break
            if not (left and right):
                if points[i] not in hull:
                    hull.append(points[i])
                if points[j] not in hull:
                    hull.append(points[j])

    return sorted(hull)
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
hull = convex_hull_brute_force(points)
print("Convex Hull:", hull)
```

```
Convex Hull: [(0, 0), (4, 6), (8, 1)]

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

4.

```python
import itertools
import math
def euclidean_distance(city1, city2):
    return math.sqrt((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2)
def tsp(cities):
    min_distance = float('inf')
    best_path = None
    start_city = cities[0]
    for permutation in itertools.permutations(cities[1:]):
        current_path = [start_city] + list(permutation) + [start_city]
        current_distance = sum(euclidean_distance(current_path[i], current_path[i + 1]) for
            i in range(len(current_path) - 1))
        if current_distance < min_distance:
            min_distance = current_distance
            best_path = current_path
    return min_distance, best_path
cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
print("Test Case 1:")
min_distance, best_path = tsp(cities1)
print(f"Shortest Distance: {min_distance}")
print(f"Shortest Path: {best_path}")
```

```
Test Case 1:
Shortest Distance: 16.969112047670894
Shortest Path: [(1, 2), (7, 1), (4, 5), (3, 6), (1, 2)]

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

5.

```python
import itertools
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
def knapsack_problem(weights, values, capacity):
    n = len(weights)
    max_value = 0
    best_combination = []
    for r in range(n + 1):
        for combination in itertools.combinations(range(n), r):
            if is_feasible(combination, weights, capacity):
                current_value = total_value(combination, values)
                if current_value > max_value:
                    max_value = current_value
                    best_combination = combination
    return best_combination, max_value
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4
print("Test Case 1:")
best_combination, max_value = knapsack_problem(weights1, values1, capacity1)
print(f"Optimal Selection: {best_combination}")
print(f"Total Value: {max_value}")
```

```
Test Case 1:
Optimal Selection: (1, 2)
Total Value: 8

=== Code Execution Success
```