```python
import math

def closest_pair(points):
    min_distance = float('inf')
    closest_points = None

    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance = math.dist(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_points = (points[i], points[j])

    return closest_points, min_distance

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

# Finding the closest pair
closest_points, min_distance = closest_pair(points)

# Output
print(f"Closest pair: {closest_points[0]} - {closest_points[1]}")
print(f"Minimum distance: {min_distance}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "c:\Users\hp\OneDrive\Desktop\tempCodeRunnerFile.python"
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979

[Done] exited with code=0 in 0.183 seconds
```

```python
import math

# Function to calculate the Euclidean distance
def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Function to find the closest pair of points using brute force
def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_points = None
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            dist = euclidean_distance(points[i], points[j])
            if dist < min_distance:
                min_distance = dist
                closest_points = (points[i], points[j])
    return closest_points, min_distance

# Test the function
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
result, distance = closest_pair_brute_force(points)
print(f"Closest pair: {result} with distance: {distance}")

# Function to determine the orientation of the triplet (p, q, r)
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  # collinear
    elif val > 0:
        return 1  # clockwise
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "c:\Users\hp\OneDrive\Desktop\tempCodeRunnerFile.python"
Closest pair: ((1, 2), (3, 1)) with distance: 2.23606797749979
Convex hull points: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]

[Done] exited with code=0 in 0.181 seconds
```

```python
import itertools
import math
# Function to calculate the Euclidean distance between two cities
def distance(city1, city2):
    return math.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)
def tsp(cities):
    min_distance = float('inf')
    shortest_path = []
    start_city = cities[0]
    for perm in itertools.permutations(cities[1:]):
        path = [start_city] + list(perm) + [start_city]
        distance_travelled = sum(distance(path[i], path[i + 1]) for i in range(len(path) - 1))
        if distance_travelled < min_distance:
            min_distance = distance_travelled
            shortest_path = path
    return min_distance, shortest_path
cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
cities2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]
print(f"Test Case 1: Shortest Distance: {tsp(cities1)[0]}, Path: {tsp(cities1)[1]}")
print(f"Test Case 2: Shortest Distance: {tsp(cities2)[0]}, Path: {tsp(cities2)[1]}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "c:\Users\hp\OneDrive\Desktop\tempCodeRunnerFile.python
Test Case 1: Shortest Distance: 16.969112047670894, Path: [(1, 2), (7, 1), (4, 5), (3, 6), (1, 2)]
Test Case 2: Shortest Distance: 23.12995011084934, Path: [(2, 4), (1, 7), (5, 9), (8, 1), (6, 3), (2, 4)]

[Done] exited with code=0 in 0.145 seconds
```

```python
import itertools
def total_cost(assignment, cost_matrix):
    return sum(cost_matrix[i][assignment[i]] for i in range(len(assignment)))
def assignment_problem(cost_matrix):
    num_tasks = len(cost_matrix)
    min_cost = float('inf')
    best_assignment = []
    for perm in itertools.permutations(range(num_tasks)):  # Generate all task assignments
        cost = total_cost(perm, cost_matrix)
        if cost < min_cost:
            min_cost = cost
            best_assignment = perm
    return min_cost, best_assignment
cost_matrix1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
cost_matrix2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
print(f"Test Case 1: Minimum Cost: {assignment_problem(cost_matrix1)[0]}, Assignment: {assignment_problem(cost_matrix1)[1]}")
print(f"Test Case 2: Minimum Cost: {assignment_problem(cost_matrix2)[0]}, Assignment: {assignment_problem(cost_matrix2)[1]}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Done] exited with code=0 in 0.145 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\tempCodeRunnerFile.python"
Test Case 1: Minimum Cost: 16, Assignment: (2, 1, 0)
Test Case 2: Minimum Cost: 17, Assignment: (2, 1, 0)
```

```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(weights, values, capacity):
    n = len(weights)
    max_value = 0
    best_items = []
    for i in range(1, n + 1):
        for subset in itertools.combinations(range(n), i):
            if is_feasible(subset, weights, capacity):
                value = total_value(subset, values)
                if value > max_value:
                    max_value = value
                    best_items = subset
    return max_value, best_items
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4
weights2 = [1, 2, 3, 4]
values2 = [2, 4, 6, 3]
capacity2 = 6
print(f"Test Case 1: Maximum Value: {knapsack(weights1, values1, capacity1)[0]}, Items: {knapsack(weights1, values1, capacity1)[1]}")
print(f"Test Case 2: Maximum Value: {knapsack(weights2, values2, capacity2)[0]}, Items: {knapsack(weights2, values2, capacity2)[1]}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "c:\Users\hp\OneDrive\Desktop\tempCodeRunnerFile.python"
Test Case 1: Maximum Value: 8, Items: (1, 2)
Test Case 2: Maximum Value: 12, Items: (0, 1, 2)

[Done] exited with code=0 in 0.133 seconds
```

```python
import itertools
import math
# Function to calculate the Euclidean distance between two cities
def distance(city1, city2):
    return math.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)
def tsp(cities):
    min_distance = float('inf')
    shortest_path = []
    start_city = cities[0]
    for perm in itertools.permutations(cities[1:]):
        path = [start_city] + list(perm) + [start_city]
        distance_travelled = sum(distance(path[i], path[i + 1]) for i in range(len(path) - 1))
        if distance_travelled < min_distance:
            min_distance = distance_travelled
            shortest_path = path
    return min_distance, shortest_path
cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
cities2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]
print(f"Test Case 1: Shortest Distance: {tsp(cities1)[0]}, Path: {tsp(cities1)[1]}")
print(f"Test Case 2: Shortest Distance: {tsp(cities2)[0]}, Path: {tsp(cities2)[1]}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "c:\Users\hp\OneDrive\Desktop\tempCodeRunnerFile.python
Test Case 1: Maximum Value: 8, Items: (1, 2)
Test Case 2: Maximum Value: 12, Items: (0, 1, 2)

[Done] exited with code=0 in 0.133 seconds
```