153. You are given a list of cities represented by their coordinates. Develop a program that utilizes exhaustive search to solve the TSP. The program should:
1.Define a function distance(city1, city2) to calculate the distance between two cities (e.g., Euclidean distance).
2.Implement a function tsp(cities) that takes a list of cities as input and performs the following:
*Generate all possible permutations of the cities (excluding the starting city) using itertools.permutations.
For each permutation (representing a potential route):
Calculate the total distance traveled by iterating through the path and summing the distances between consecutive cities.
Keep track of the shortest distance encountered and the corresponding path.
Return the minimum distance and the shortest path (including the starting city at the beginning and end).
Include test cases with different city configurations to demonstrate the program's functionality. Print the shortest distance and the corresponding path for each test case.
Test Cases:
Simple Case: Four cities with basic coordinates (e.g., [(1, 2), (4, 5), (7, 1), (3, 6)])
More Complex Case: Five cities with more intricate coordinates (e.g., [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)])
Output:
Test Case 1:
Shortest Distance: 7.0710678118654755
Shortest Path: [(1, 2), (4, 5), (7, 1), (3, 6), (1, 2)]

AIM: To find the minimu shortest path

PROGRAM:

```
import itertools
import math

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

def tsp(cities):
    n = len(cities)
    if n < 2:
        return float('inf'), []
    all_permutations = itertools.permutations(cities[1:])

    min_distance = float('inf')
    shortest_path = None

    for perm in all_permutations:
        path = [cities[0]] + list(perm) + [cities[0]]


        total_distance = 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

def test_tsp(cities, case_name):
    print(f"Test Case {case_name}:")
    print(f"Cities: {cities}")
    min_dist, shortest_path = tsp(cities)
    print(f"Shortest Distance: {min_dist}")
    print(f"Shortest Path: {shortest_path}\n")

cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
test_tsp(cities1, 1)
```

OUTPUT:

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

TIME COMPLEXITY: O(n-1!*n)