N, NP and NP-COMPLETE

N (Nondeterministic Polynomial Time):

- N represents the class of decision problems that can be solved in polynomial time on a nondeterministic Turing machine.
- In simpler terms, problems in class N can be verified in polynomial time, but we don't necessarily know how to find the solution efficiently.
- Eg. We're checking if a number is prime in the code below. Verifying that a number is prime can be done in polynomial time, so it's in class N.

```
def isPrime(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

NP (Nondeterministic Polynomial Time):

- NP represents the class of decision problems for which a proposed solution can be verified in polynomial time on a deterministic Turing machine.
- While NP problems can't necessarily be solved in polynomial time, if you give a proposed solution, you can efficiently check if it's correct.
- If we were to check if a subset of numbers sums up to a target value. Verifying a solution (checking if a subset sums to the target) can be done in polynomial time, so it's in class NP.

NP-COMPLETE:

- NP-COMPLETE represents a special class of problems within NP that are as hard as the hardest problems in NP.
- If you can find an efficient algorithm to solve any NP-COMPLETE problem, you can solve all problems in NP efficiently.
- Eg. Traveling Salesman Problem (TSP). It's a combinatorial optimization problem where you want to find the shortest possible route that visits a set of cities exactly once and returns to the starting city. There's no known polynomial-time algorithm to solve it optimally.

```
# Example of a brute-force solution for the TSP
def traveling_salesman(graph, start):
    # Generate all possible permutations of cities
    permutations = itertools.permutations(graph.keys())
    min_distance = float('inf')
    best_path = None

    for perm in permutations:
        path = list(perm) + [start]
        distance = calculate_total_distance(path, graph)
        if distance < min_distance:
            min_distance = distance
            best_path = path

    return best_path, min_distance
```

- This code demonstrates a brute-force approach to solving the TSP, but it's not efficient for larger instances, which is a characteristic of NP-COMPLETE problems.

Summary

- N represents problems that can be verified in polynomial time.
- NP represents problems for which proposed solutions can be verified in polynomial time.
- NP-COMPLETE are believed to be not solvable in polynomial time.