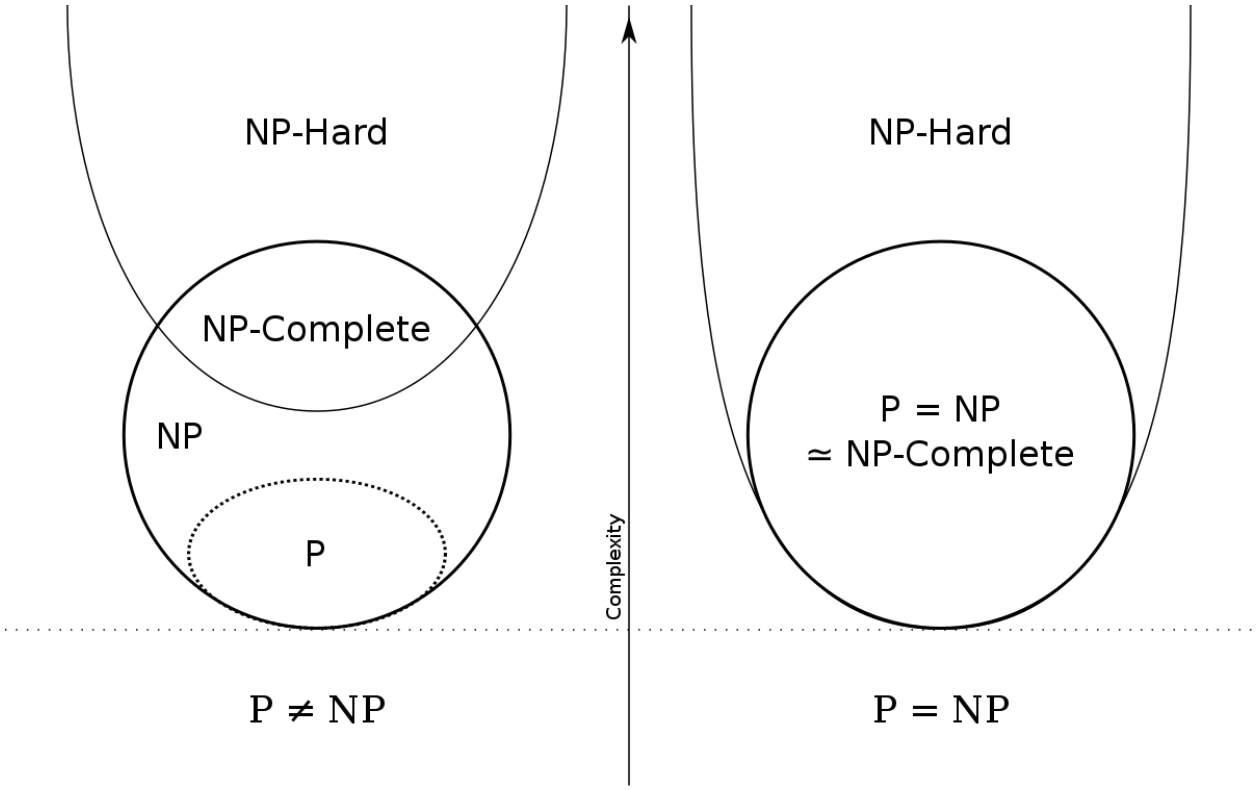
Intractability II (P, NP, NP Hard and NP-Complete)

In computer science, there exist some problems whose solutions are not yet found, the problems are divided into classes known as Complexity Classes. In complexity theory, a Complexity Class is a set of problems with related complexity. These classes help scientists to group problems based on how much time and space they require to solve problems and verify the solutions. It is the branch of the theory of computation that deals with the resources required to solve a problem.

The common resources are time and space, meaning how much time the algorithm takes to solve a problem and the corresponding memory usage.
The time complexity of an algorithm is used to describe the number of steps required to solve a problem, but it can also be used to describe how long it takes to verify the answer.
The space complexity of an algorithm describes how much memory is required for the algorithm to operate.



P (Polynomial Time):

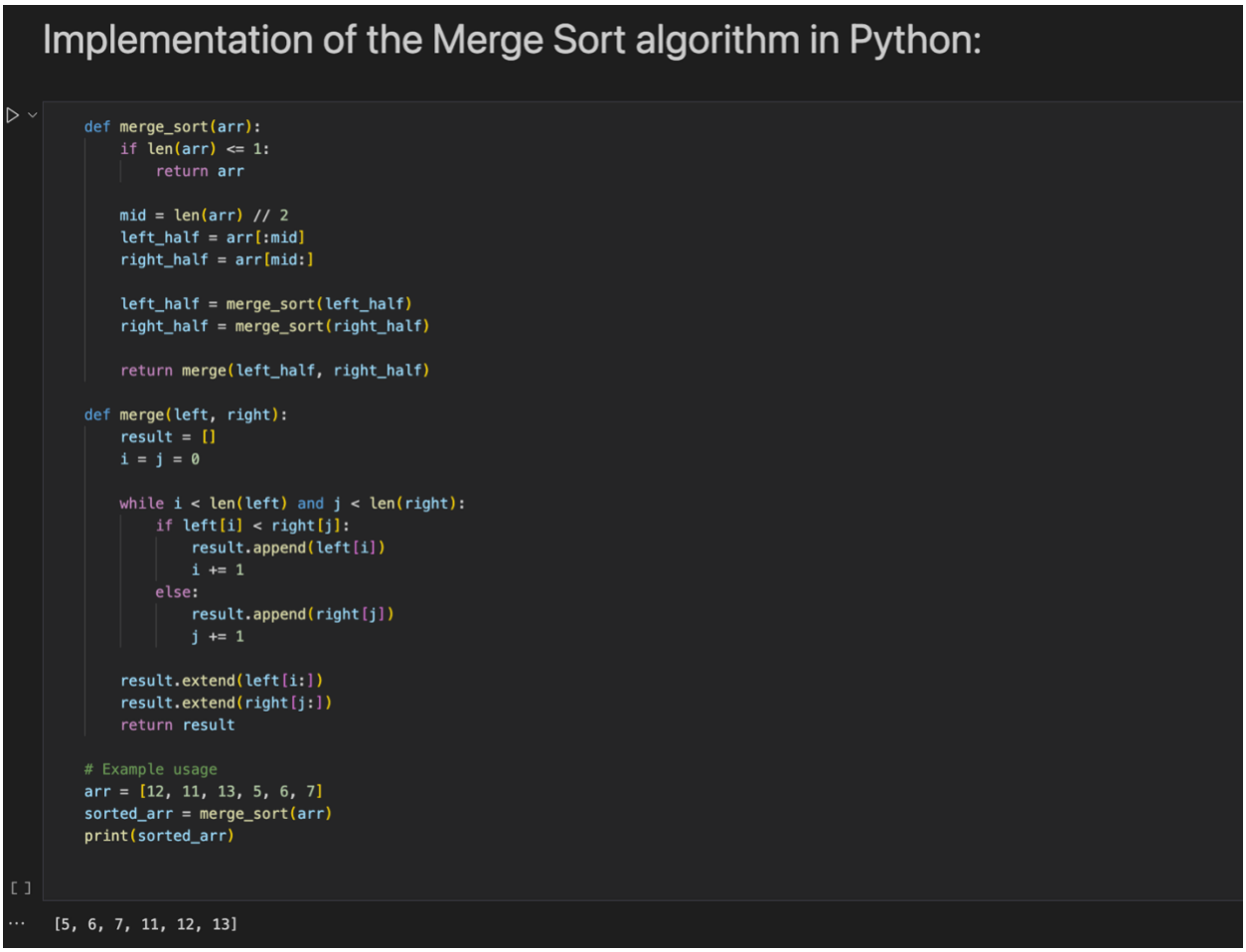
The class P contains decision problems that can be solved by a deterministic Turing machine in polynomial time. In simpler terms, these are problems for which there exists an efficient algorithm that can provide a solution in a reasonable amount of time as the input size grows.

Key characteristics of problems in P:

1. Efficient Algorithms: There are algorithms that can solve these problems with a time complexity of O(n^k) for some constant k, where n is the input size.
2. Polynomial-Time Verification: The solutions can be verified in polynomial time.

Examples of problems in P:

1. Sorting a list of numbers using Merge Sort or Quick Sort.
2. Finding the shortest path in a weighted graph using Dijkstra's algorithm.



NP (Nondeterministic Polynomial Time):

The class NP contains decision problems for which solutions can be verified quickly by a deterministic Turing machine in polynomial time. While finding a solution might be hard, verifying a proposed solution is relatively easy.

Key characteristics of problems in NP:

1. Polynomial-Time Verification: Solutions can be verified in polynomial time.
2. Non-deterministic Computation: A hypothetical non-deterministic Turing machine could potentially solve these problems in polynomial time, even if we don't know how to do so efficiently using a deterministic machine.

Examples of problems in NP:

1. The Hamiltonian Cycle problem: Given a graph and a cycle, verifying that it visits each vertex exactly once can be done in polynomial time.
2. The Subset Sum problem: Given a set of integers and a target sum, verifying that a subset sums to the target can be done in polynomial time.

```python
def is_hamiltonian_cycle(graph, cycle):
    if len(cycle) != len(graph) or cycle[0] != cycle[-1]:
        return False

    visited = [False] * len(graph)

    for i in range(len(cycle) - 1):
        if cycle[i + 1] not in graph[cycle[i]]:
            return False
        if visited[cycle[i]]:
            return False
        visited[cycle[i]] = True

    for vertex in visited:
        if not vertex:
            return False

    return True

# Example graph represented as an adjacency list
graph = {
    0: [1, 3],
    1: [0, 2, 3],
    2: [1],
    3: [0, 1]
}

# Example cycle to test
cycle = [0, 1, 2, 3, 0]

if is_hamiltonian_cycle(graph, cycle):
    print("The given cycle is a Hamiltonian Cycle.")
else:
    print("The given cycle is not a Hamiltonian Cycle.")
```

✓ 0.2s

```
The given cycle is not a Hamiltonian Cycle.
```

NP-hard and NP-complete:

The terms NP-hard and NP-complete describe problems that are at least as hard as the hardest problems in NP. An NP-hard problem may or may not be in NP itself, while an NP-complete problem is both in NP and NP-hard.

Key characteristics of NP-hard/NP-complete problems:

1. High Complexity: These problems are challenging and often involve searching through a large space of possibilities.
2. Reducibility: Many NP-hard problems can be reduced to each other, meaning that if one problem can be solved efficiently, then all problems it can be reduced to can also be solved efficiently.

Examples of NP-hard/NP-complete problems:

1. The Traveling Salesman Problem (TSP): Finding the shortest route that visits a set of cities and returns to the starting city.
2. The Knapsack Problem: Choosing a subset of items with maximum value while staying within a weight limit.
3. The Boolean Satisfiability Problem (SAT): Determining whether a given Boolean formula can be satisfied by assigning truth values to its variables.

```python
import itertools

def distance(city1, city2):
    # Replace this with your distance calculation logic (e.g., Euclidean distance)
    x1, y1 = city1
    x2, y2 = city2
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

def total_distance(route, cities):
    total = 0
    for i in range(len(route)):
        total += distance(cities[route[i]], cities[route[(i + 1) % len(route)]])
    return total

def solve_tsp(cities)
    num_cities = len(cities)
    shortest_distance = float('inf')
    shortest_route = []

    for perm in itertools.permutations(range(num_cities)):
        current_distance = total_distance(perm, cities)
        if current_distance < shortest_distance:
            shortest_distance = current_distance
            shortest_route = perm

    return shortest_route, shortest_distance

# Example usage with different cities
cities = [(0, 0), (3, 1), (1, 4), (5, 6), (8, 3)]
shortest_route, shortest_distance = solve_tsp(cities)

print("Cities:", cities)
print("Shortest route:", shortest_route)
print("Shortest distance:", shortest_distance)
```

(parameter) cities: Any

✓ 0.2s

```
Cities: [(0, 0), (3, 1), (1, 4), (5, 6), (8, 3)]
Shortest route: (0, 1, 4, 3, 2)
Shortest distance: 21.385324735039408
```

How to prove that a given problem is NP-complete?

From the definition of NP-complete, it appears impossible to prove that a problem L is NP-Complete. It requires us to that show every problem in NP is polynomial time reducible to L. Fortunately, there is an alternate way to prove it. The idea is to take a known NP-Complete problem and reduce it to L. If a polynomial-time reduction is possible, we can prove that L is NP-Complete by transitivity of reduction (If an NP-Complete problem is reducible to L in polynomial time, then all problems are reducible to L in polynomial time).

In summary, these classifications help us understand the complexity of problems and their relationships in terms of solvability and computational efficiency. The exploration of these classes is fundamental to various areas of computer science, including algorithm design and complexity theory.

Quiz Questions

Question 1:
Which complexity class represents decision problems that can be solved by a non-deterministic Turing machine in polynomial time?
a) P
b) NP
c) NP-hard
d) NP-complete

Answer: b) NP

Question 2:
A problem is in NP-complete if:
a) It can be solved using a non-deterministic Turing machine in polynomial time.
b) It can be solved using a deterministic Turing machine in polynomial time.
c) It is at least as hard as the hardest problems in NP.
d) It is in the class P.

Answer: c) It is at least as hard as the hardest problems in NP.

Question 3:
The problem of finding the shortest path in a weighted graph is:
a) In P
b) In NP
c) NP-hard
d) NP-complete

Answer: a) In P

Question 4:
Which problem is NP-hard but not known to be in NP?
a) Subset Sum
b) Traveling Salesman Problem
c) Sorting
d) Binary Search

Answer: b) Traveling Salesman Problem

Question 5:
The P vs. NP problem asks whether:
a) All problems in P are also in NP.
b) All problems in NP are also in P.

c) P and NP are the same complexity class.
d) P is a proper subset of NP.

Answer: a) All problems in P are also in NP.

Question 6:
If a problem A can be reduced to problem B in polynomial time, and problem B is NP-complete, then problem A:
a) Is also NP-complete
b) Is in P
c) Is in NP-hard
d) Cannot be solved

Answer: a) Is also NP-complete

Question 7:
Which sorting algorithm has the worst-case time complexity of O(n log n) but is not stable?
a) Merge Sort
b) Quick Sort
c) Insertion Sort
d) Bubble Sort

Answer: b) Quick Sort

Question 8:
The problem of determining whether a given Boolean formula is satisfiable is known as:
a) Traveling Salesman Problem
b) Subset Sum
c) Hamiltonian Cycle
d) Boolean Satisfiability (SAT)

Answer: d) Boolean Satisfiability (SAT)

Question 9:
A problem is NP-complete if:
a) It can be solved by a non-deterministic Turing machine in polynomial time.
b) It is in NP and at least as hard as the hardest problems in NP.
c) It can be solved by a deterministic Turing machine in polynomial time.
d) It is in P and NP.

Answer: b) It is in NP and at least as hard as the hardest problems in NP.

Question 10:
Which of the following statements is true about NP-hard problems?

a) NP-hard problems can be solved in polynomial time.
b) NP-hard problems are easier than problems in P.
c) NP-hard problems are at least as hard as the hardest problems in NP.
d) NP-hard problems are the same as problems in P.

Answer: c) NP-hard problems are at least as hard as the hardest problems in NP.