Introduction to Complexity Classes

In computer science, problems are classified based on the resources (such as time or space) required to solve them. The most common classifications are based on **time complexity**, which measures how the running time of an algorithm grows as the size of the input increases.

P (Polynomial Time)

Definition:

- **P** is the class of decision problems (yes/no problems) that can be solved by a **deterministic** Turing machine in **polynomial time**.
- If a problem belongs to P, it means that there exists an algorithm that can solve any instance of the problem in a time that is at most a polynomial function of the input size (e.g., $O(n^k)$, where k is a constant).

Examples:

- **Sorting algorithms** like Merge Sort and Quick Sort (both run in O(nlogn).
- **Shortest Path Problem** using Dijkstra's Algorithm (runs in $O(n^2)$ for dense graphs with n vertices).
- **Maximum Flow Problem** solved using the Ford-Fulkerson algorithm.

Significance:

- **P problems** are generally considered to be "efficiently solvable," i.e., problems for which a polynomial-time solution is considered feasible in practice.

NP (Nondeterministic Polynomial Time)

Definition:

- **NP** is the class of decision problems for which a solution can be **verified** in polynomial time by a **deterministic** Turing machine.
- A problem is in NP if, given a proposed solution, we can check whether the solution is correct in polynomial time.
- It is also described as the class of problems that can be solved by a **nondeterministic** Turing machine in polynomial time.

Key Points:

- **Verification in Polynomial Time**: For NP problems, even if we don't know how to solve them efficiently, we can at least verify a given solution in polynomial time.
- **Deterministic vs. Nondeterministic Machines**: A deterministic machine follows a specific sequence of steps, while a nondeterministic machine is like having the ability to "guess" the right solution and then verify it efficiently.

Examples:

- **Boolean Satisfiability Problem (SAT)**: Given a Boolean formula, is there a truth assignment to the variables that makes the formula true?
- **Traveling Salesman Problem (TSP)** (Decision Version): Given a set of cities and distances, is there a route visiting each city exactly once with a total length less than some value D?
- **Knapsack Problem (0/1)**: Given a set of items with weights and values, is there a subset that fits in a knapsack of capacity W and has a total value of at least V?

Relationship between P and NP:

- **P ⊆ NP**: Every problem in P is also in NP because if a problem can be solved in polynomial time, we can also verify its solution in polynomial time.
- **Open Question**: Is **P = NP**? This is the most famous open problem in computer science. If P=NP, it means that every problem that can be verified in polynomial time can also be solved in polynomial time. Currently, it is widely believed that P≠NP but this has not been proven.

NP-Complete (NPC)
Definition:

- A problem is **NP-Complete** if:
    1. It is in NP.
    2. Every other problem in NP can be **reduced** to it in **polynomial time**.
- **NP-Complete problems** are the hardest problems in NP in the sense that if any NP-Complete problem can be solved in polynomial time, then every problem in NP can also be solved in polynomial time (i.e., P=NP).

Reductions:

- A polynomial-time **reduction** from one problem A to another problem B means that if we can solve B, we can also solve A using a polynomial amount of additional time.
- **Cook's Theorem** (1971) was the first to show that **SAT (Boolean Satisfiability)** is NP-Complete, thereby proving that NP-Complete problems exist.

Examples:

- **SAT (Boolean Satisfiability Problem)**: The first problem proven to be NP-Complete by Cook.
- **3-SAT**: A special case of SAT where each clause has exactly three literals.
- **Traveling Salesman Problem (TSP)** (Decision Version): Given a set of cities and distances, is there a tour of length at most k?
- **Vertex Cover**: Given a graph and an integer k, does there exist a subset of vertices of size k such that every edge has at least one endpoint in this subset?

Significance:

- **If one NP-Complete problem can be solved in polynomial time**, then all NP problems can be solved in polynomial time (i.e., P=NP).
- NP-Complete problems are widely believed to not have polynomial-time solutions, though this has not been proven.

NP-Hard (NPH)
Definition:

- A problem is **NP-Hard** if:
    - Every problem in NP can be **reduced** to it in **polynomial time**, but the problem itself may **not** belong to NP (i.e., it might not even be a decision problem, or it might not have a solution verifiable in polynomial time).
- NP-Hard problems are at least as hard as the hardest problems in NP, but they may not be decision problems, or their solutions might not be verifiable in polynomial time.

Examples:

- **Halting Problem**: The problem of determining whether a given program will halt on a particular input is NP-Hard but not in NP (because it is undecidable).
- **Traveling Salesman Problem (TSP)** (Optimization Version): Find the shortest possible route that visits each city exactly once and returns to the origin city.
- **Knapsack Problem (Optimization Version)**: Maximize the total value of the items placed into the knapsack without exceeding its weight capacity.

Relationship with NP-Complete:

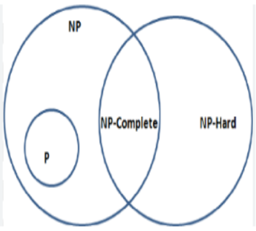
- **All NP-Complete problems are NP-Hard**, but not all NP-Hard problems are NP-Complete.
- NP-Complete problems are decision problems in NP, while NP-Hard problems may not necessarily be decision problems.

Relationship Between P, NP, NP-Complete, and NP-Hard

- **P**: Problems that can be solved in polynomial time.
- **NP**: Problems that can be verified in polynomial time.

- **NP-Complete**: The hardest problems in NP; if one of them can be solved in polynomial time, then all NP problems can be solved in polynomial time (i.e., P=NP).
- **NP-Hard**: Problems that are at least as hard as NP-Complete problems, but they may not belong to NP (they could be optimization or undecidable problems).

Diagram:
Here's how the classes relate to each other:
$P \subseteq NP \subseteq \text{NP-Hard}$
$\text{NP-Complete} \subseteq NP \cap \text{NP-Hard}$



Class	Definition	Example Problems
P	Problems solvable in polynomial time.	Sorting, Dijkstra's Shortest Path
NP	Problems whose solutions can be verified in polynomial time.	SAT, TSP (Decision), Knapsack
NP-Complete	The hardest problems in NP; all NP problems can be reduced to them.	SAT, 3-SAT, TSP (Decision), Vertex Cover
NP-Hard	Problems that are at least as hard as NP-Complete, may not be in NP.	Halting Problem, TSP (Optimization), Knapsack (Optimization)