1. Nondeterministic Algorithms

A nondeterministic algorithm is one that may produce different outcomes even when given the same input. In theoretical computer science, this concept is modeled using a nondeterministic Turing machine, which can make choices between multiple possible next moves.

- Deterministic Algorithm: Always follows the same sequence of steps for a given input (e.g., Merge Sort, Binary Search).
- Nondeterministic Algorithm: Can conceptually explore many possible execution paths *simultaneously*, choosing one that leads to a correct answer if it exists.

In complexity theory, a nondeterministic algorithm is said to solve a problem in polynomial time if there exists *at least one computation path* that reaches a correct solution in polynomial time.

This model gives rise to the complexity class NP (Nondeterministic Polynomial time) — problems whose solutions *can be verified* quickly (in polynomial time) on a deterministic machine.

---

2. Complexity Classes P and NP

- P (Polynomial time): Problems that *can be solved* efficiently by a deterministic algorithm.

  Example: Sorting a list with Merge Sort.

- NP (Nondeterministic Polynomial time): Problems whose *solutions can be verified* efficiently once a potential solution is given.

  Example: For the Subset Sum Problem, if you are told which subset sums to the target, you can check it easily, but finding it from scratch might be hard.

The famous P vs NP problem asks whether all problems that have verifiable solutions (NP) are also efficiently solvable (P).

---

3. NP-Hard Problems

A problem is NP-Hard if *all problems in NP* can be reduced to it in polynomial time.

- They are at least as hard as the hardest NP problems.
- They do *not have to be in NP* themselves (meaning we might not be able to verify their solutions efficiently).

Examples:

- The optimization form of the Traveling Salesman Problem (TSP)
- The Halting Problem (which is undecidable)

4. NP-Complete Problems

A problem is NP-Complete if it satisfies two conditions:

1. It belongs to NP (its solution can be verified quickly).
2. It is NP-Hard (every NP problem can be reduced to it).

Thus, NP-Complete problems represent the hardest problems *within* NP.

Examples:

- Boolean Satisfiability Problem (SAT)
- 3-SAT
- Hamiltonian Cycle Problem
- Subset Sum Problem

If any NP-Complete problem can be solved in polynomial time, then P = NP.

5. Cook's Theorem (Stephen Cook, 1971)

Cook's Theorem was the first formal proof that an NP-Complete problem exists.

It states:

"The Boolean Satisfiability Problem (SAT) is NP-Complete."

In more detail:

- SAT asks whether there exists a truth assignment to Boolean variables that makes a formula true.
- Cook showed that every problem in NP can be reduced to SAT in polynomial time.
- This established SAT as the first NP-Complete problem, meaning every NP problem can be represented as a Boolean formula.

This laid the foundation for reductions: if we can reduce any NP problem to another efficiently, and that second problem is already known NP-Complete, then the second problem must also be NP-Complete.

7. Summary Table

Category	Definition	Example	Verification in Polynomial Time	Solvable in Polynomial Time
P	Problems solvable efficiently	Merge Sort	Yes	Yes
NP	Solutions verifiable efficiently	Subset Sum	Yes	Unknown
NP-Complete	Verifiable & as hard as any NP problem	SAT	Yes	Unknown
NP-Hard	At least as hard as NP problems	TSP (optimization)	Not necessarily	Unknown

---