Introduction

Computer science problems are classified into:

- **Tractable Problems:** Solved by a polynomial-time algorithm, with running time $O(n^k)$ for constants k and C. Most known algorithms (e.g., sorting, shortest path) fall into this category.
- **Intractable Problems:** Unlikely to have a polynomial-time solution. Best-known algorithms take exponential time (e.g., $O(2^n)$). Examples include the Traveling Salesman Problem and Clique.

Problem Types

- **Decision Problem:** Has only two possible answers: *yes* or *no*.
- **Optimization Problem:** Involves maximizing or minimizing a quantity.
- Many optimization problems have decision versions. NP-completeness theory focuses on decision problems for simplicity.

Algorithm Types

- **Deterministic Algorithm:** Given the same input, always produces the same output and follows the same sequence of steps.
- **Non-deterministic Algorithm:** May produce different outputs for the same input on different runs. Contains steps (e.g., `choice()`) that are not predefined.

NP Problems□ Non-deterministic polynomial.

A special class of intractable problems with properties:

1. Only exponential-time algorithms are known.
2. There is nondeterministic algorithm on nondeterministic machine that can solve it in polynomial-time.
3. Can not find solution in polynomial-time but if given solution it can be verified in polynomial-time.
4. We can verify the correctness of solution in polynomial-time.

Satisfiability (SAT)

- A Boolean formula is *satisfiable* if there exists a truth assignment to its variables that makes the formula true.
- Input: A Boolean formula in Conjunctive Normal Form (CNF).
- Output: Determine if a satisfying assignment exists.
- Solved in $O(2n\cdot)O(2n)$ time by checking all $2n$ possible assignments.

$(x \wedge y) \wedge (z \wedge y) \wedge (z \wedge x)$ 3 2!

Clique Problem

- Input: An undirected graph $G(V,E)$ and an integer k.
- Task: Determine if G contains a *clique* of size ***k*** (a complete subgraph).

- Solved in $O\left(\binom{n}{k}k^2\right)$ time by checking all subsets of k vertices.

Complexity Classes P and NP

- P: Class of decision problems solvable in **deterministic polynomial time** (e.g., sorting, MST).
- NP: Class of decision problems **verifiable** in polynomial time if a solution is provided (e.g., SAT, Clique). Stands for "nondeterministic polynomial time."
- It is known that $P \subseteq NP$. The question of whether P=NP remains one of the biggest open problems in computer science.

NP-Hard and NP-Complete

- **NP-Hard:** A problem is NP-hard if every problem in NP can be *polynomially reduced* to it. It is at least as hard as the hardest problems in NP.
- **NP-Complete:** A problem that is both in NP and NP-hard. These are the hardest problems in NP.
- **Strategy:** When a polynomial-time deterministic algorithm is not found for an NP problem:
  1. Write a non-deterministic polynomial-time algorithm (preserving the work for future research) it is verifiable in p.
  2. Show polynomial-time reductions to other NP-hard problems (e.g., SAT) to prove NP-hardness reduction take p.

Polynomial Reduction

- Problem A is polynomially reducible to problem B ($A \propto B$) if there exists a polynomial-time transformation converting any instance of A into an instance of B, such that a "yes" answer for B implies a "yes" for A, and vice versa.
- Reduction is transitive: if SAT $\propto$ Problem $L1$ and $L1 \propto L2$,
- then $L2$ is also NP-hard and SAT$\propto L2$.