# Isaiah Lugo

# CSM VI - Algorithms

# Week 12 - P vs. NP Problems

## 1. P-Problem

P problems are a class of computational problems that can be solved efficiently by a deterministic algorithm in polynomial time. This means the time it takes to solve the problem grows at most as a polynomial function of the input size.  
  
Example: The problem of sorting a list (e.g., using Merge Sort or Quick Sort). These algorithms solve the problem in O(n log n) time.  
  
Efficient algorithms for P problems have predictable performance and are scalable for large inputs. They form the basis of many common computing tasks like searching, sorting, and basic graph traversal (e.g., BFS and DFS).

## 2. NP-Problem

NP (Nondeterministic Polynomial time) problems are problems for which a proposed solution can be verified in polynomial time, even if finding the solution itself might not be feasible in polynomial time.  
  
Example: The Subset Sum problem – Given a set of integers, is there a subset whose sum is equal to a specific value?  
  
NP problems differ from P problems in that we do not necessarily have efficient algorithms to solve them, but we can quickly verify a given solution.  
  
Nondeterministic polynomial time means that a theoretical computer could 'guess' a correct answer and verify it in polynomial time, even if there’s no known way to deterministically find that answer efficiently.

## 3. NP-Complete Problems

NP-Complete problems are the hardest problems in the NP class. If a polynomial-time solution is found for any NP-Complete problem, it would imply that P = NP, meaning all NP problems could be solved efficiently.  
  
Example: The Traveling Salesman Problem (decision version) – Is there a tour that visits each city exactly once and has a total cost less than or equal to K?  
  
To prove a problem is NP-Complete, we use reduction – transforming a known NP-Complete problem into the new problem in polynomial time. This shows that solving the new problem efficiently would also solve the known NP-Complete problem efficiently.

## 4. NP-Hard Problems

NP-Hard problems are at least as hard as the hardest problems in NP. They are not necessarily in NP themselves, which means they may not even have solutions that can be verified in polynomial time.  
  
Example: The Halting Problem – Determine whether a program will halt or run forever on a given input.  
  
NP-Hard problems include all NP-Complete problems, but also problems that are undecidable or not verifiable in polynomial time. Solving an NP-Hard problem efficiently would solve all NP problems as well, but the reverse isn’t true.

## 5. Summary Table

|  |  |  |
| --- | --- | --- |
| Problem Class | Description | Example |
| P | Solvable in polynomial time | Merge Sort (O(n log n)) |
| NP | Verifiable in polynomial time | Subset Sum Problem |
| NP-Complete | Hardest in NP; verifiable and reducible | Traveling Salesman (Decision Version) |
| NP-Hard | At least as hard as NP; may not be verifiable | Halting Problem |