

Q01: What is the definition of a P problem?

- prompt: Briefly define P class problems in computational complexity. How do they relate to polynomial-time algorithms?
- model_name: ChatGPT
- model_version: 4o
- response: A **P problem** is a class of computational problems that can be solved by an algorithm in **polynomial time**. Specifically, a problem belongs to the P class if there exists an algorithm that solves the problem in time proportional to a polynomial function of the size of the input (i.e., time complexity $O(n^k)$ for some constant k). In simpler terms, P problems are those for which a solution can be found efficiently (in polynomial time) as the size of the input grows. Polynomial-time algorithms are the main characteristic of P class problems. These problems are considered "easy" or "tractable" because they can be solved in a reasonable amount of time even for large inputs.

Q02: What is the difference between NP and P?

- prompt: Briefly explain the difference between NP and P. Specifically, is every problem in NP also in P? Why or why not?
- model_name: ChatGPT
- model_version: 4o
- response: The main difference between NP and P lies in the complexity of solving problems. P consists of problems that can be solved in polynomial time, meaning they have efficient algorithms to find solutions. NP, on the other hand, includes problems for which solutions can be verified in polynomial time, but it's not necessarily known whether they can be solved in polynomial time. Not every problem in NP is known to be in P. The famous P vs NP problem asks whether every problem that can be verified in polynomial time (NP) can also be solved in polynomial time (P). Currently, no one has proven that $P = NP$, and most believe that $P \neq NP$, meaning there are problems in NP that are not in P.

Q03: What is the significance of NP-completeness?

- prompt: Briefly define NP-complete problems. Why are they significant in computational complexity theory?
- model_name: ChatGPT
- model_version: 4o
- response: NP-complete problems are the hardest problems in NP. A problem is NP-complete if it is in NP and if every problem in NP can be reduced to it in polynomial time. The significance lies in the fact that if one NP-complete problem can be solved in polynomial time, all problems in NP

can also be solved in polynomial time, thus proving that NP equals P. These problems are central to understanding the boundary between problems that are efficiently solvable and those that are not.

Q04: What is the role of polynomial reductions in classifying problems as NP-hard?

- prompt: Briefly explain the concept of polynomial reductions and their role in classifying problems as NP-hard.
- model_name: ChatGPT
- model_version: 4o
- response: Polynomial reductions are a key concept in classifying problems as NP-hard. They refer to the process of transforming one problem into another in polynomial time, such that a solution to the second problem can be used to solve the first. Specifically, if a problem AAA can be reduced to problem BBB in polynomial time, and if BBB is known to be NP-complete (i.e., NP-hard and in NP), then AAA is considered NP-hard.

This reduction shows that solving problem AAA is at least as difficult as solving problem BBB. If we can solve problem BBB efficiently, we could also solve problem AAA efficiently, implying that AAA is NP-hard. The role of polynomial reductions is crucial because they establish the computational difficulty of a problem by showing that it is as hard as, or harder than, other known NP-hard problems.

Q05: How does the concept of NP-hard relate to real-world applications in fields like cryptography?

- prompt: Briefly discuss the relevance of NP-hard problems in practical applications like cryptography. How does their complexity influence the design of cryptographic systems?
- model_name: ChatGPT
- model_version: 4o
- response: NP-hard problems play a significant role in cryptography, particularly in the design of secure systems. These problems are computationally challenging to solve because there is no known efficient algorithm (polynomial time) to find an exact solution, which makes them ideal for use in cryptographic schemes.

In cryptography, the difficulty of NP-hard problems underpins the security of many algorithms. For example, encryption methods such as RSA rely on the difficulty of factoring large prime numbers, which is an NP-hard problem. Similarly, the security of lattice-based cryptographic schemes often depends on solving problems like the Shortest Vector Problem (SVP), which is NP-hard. The hardness of these problems ensures that even with access to large computational resources, breaking the cryptographic system remains practically infeasible.

The complexity of NP-hard problems influences cryptographic design by ensuring that encryption

and decryption operations are one-way functions—easy to compute in one direction (e.g., encryption) but computationally expensive in the reverse direction (e.g., decryption or breaking the encryption). This asymmetry is crucial for creating secure systems.

As cryptographic systems evolve, researchers continue to explore NP-hard problems as the foundation for next-generation algorithms, particularly in post-quantum cryptography, which aims to resist attacks from quantum computers that could potentially solve these problems more efficiently than classical computers.