**P – Complexity Class**

P is a complexity class that represents the set of all decision problems that can be

solved in polynomial time. That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

**Example**

Given a connected graph g, can its vertices be colored using two colors so that no edge is monochromatic?

**Algorithm**: Start with an arbitrary vertex, color it red and all of its neighbors blue and continue. stop when you run out of vertices or you areforced to make an edge have both of its endpoints be the same color.

**NP – COMPLEXITY CLASS**

NP is a complexity class that represents the set of all decision problems for which

The instances where the answer is "Yes" have proofs that can be verified in

Polynomial time. This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.

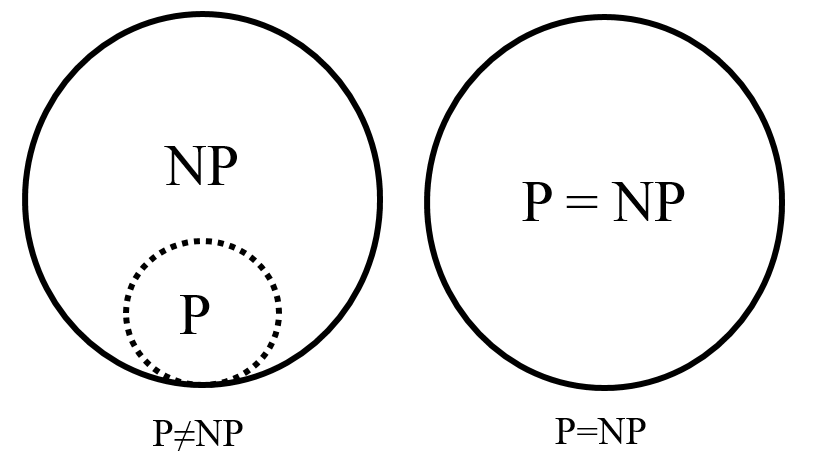
**Example**

Integer factorization is in NP. This is the problem that given integers n and m, is there an integer f with 1 < f < m, such that f divides n (f is a small factor of n)?

This is a decision problem because the answers are yes or no. If someone hands us an instance of the problem (so they hand us integers n and m) and an integer f with 1 < F < m, and claim that f is a factor of n (the certificate), we can check the answer in polynomial time by performing the division n / f.

**Classifying Problems: P Vs NP**

Problems that we know an efficient algorithm for that is capable of producing a solution in polynomial time are classified as P problems—P means polynomial time, in this instance. This was obviously the first subset of problems we were able to classify: of all these problems out there, at least we managed to solves these over here. Things like sorting lists, balancing trees, encrypting data are all problems that we have efficient algorithms for and so belong to the subset P. Later, we found another subset of problems that P itself was a subset of, NP problems. The NP stands for nondeterministic polynomial time, but for our purposes, you don’t need to know too much about what that means except that its part of the foundational, Turing-era computer science that underpins every single modern computer. What you do need to know is that NP problems do not have a known algorithm that can produce a result in polynomial time. However, if you are given a solution to an **NP problem**, verifying that it is correct is easy and can be done in **polynomial time** or less. We use this fact every time we unlock our iPhones or send messages over WhatsApp. As it turns out, **NP problems** are perfect for encryption; there is only one way to solve the problem that unlocks the encryption quickly, you need to have the answer ahead of time.



**Real Life Example**

Consider Sudoku, a game where the player is given a partially filled-in grid of numbers and attempts to complete the grid following certain rules. Given an incomplete Sudoku grid, of any size, is there at least one legal solution? Any proposed solution is easily verified, and the time to check a solution grows slowly (polynomially) as the grid gets bigger. However, all known algorithms for finding solutions take, for difficult examples, time that grows exponentially as the grid gets bigger. So, Sudoku is in NP (quickly checkable) but does not seem to be in P (quickly solvable). Thousands of other problems seem similar, in that they are fast to check but slow to solve.