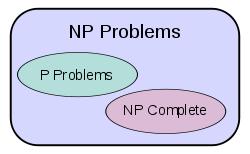
P vs NP Problem



The P versus NP problem is a major unsolved problem in computer science. It asks whether every problem whose solution can be quickly verified can also be solved quickly.

The informal term quickly, used above, means the existence of an algorithm solving the task that runs in polynomial time, such that the time to complete the task varies as a polynomial function on the size of the input to the algorithm (as opposed to, say, exponential time). The general class of questions for which some algorithm can provide an answer in polynomial time is called "class P" or just "P". For some questions, there is no known way to find an answer quickly, but if one is provided with information showing what the answer is, it is possible to verify the answer quickly. The class of questions for which an answer can be verified in polynomial time is called NP, which stands for "nondeterministic polynomial time".

**Deterministic Polynomial Time (P):**

The class P consists of those problems that are solvable in polynomial time, i.e. these problems can be solved in time ***O(nk)*** in worst-case, where **k** is constant.

These problems are called **tractable**, while others are called **intractable or super-polynomial**.

Formally, an algorithm is polynomial time algorithm, if there exists a polynomial ***p(n)*** such that the algorithm can solve any instance of size **n** in a time ***O(p(n))***.

Problem requiring ***Ω(n50)*** time to solve are essentially intractable for large ***n***. Most known polynomial time algorithm run in time ***O(nk)*** for fairly low value of ***k***.

The advantages in considering the class of polynomial-time algorithms is that all reasonable **deterministic single processor model of computation** can be simulated on each other.

**Non-Deterministic Polynomial Time (NP):**

The class NP consists of those problems that are verifiable in polynomial time. NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information. Hence, we aren’t asking for a way to find a solution, but only to verify that an alleged solution really is correct.

Every problem in this class can be solved in exponential time using exhaustive search.

**NP – Hard:**

This is the class of problems which are at least as hard as the hardest problems in NP. Problems belonging to this class may or may not be part of NP, that is, the hardest problems of NP belong to the intersection of NP and NP-Hard. Problems in NP-Hard may not even be decision problems.

Example of a problem which is NP-Hard but not NP is the problem of identifying a chess move in any given board state that is the best possible move to make.

**NP – Complete:**

A language **B** is ***NP-complete*** if it satisfies two conditions:

* **B** is in NP
* Every **A** in NP is polynomial time reducible to **B**.

If a language satisfies the second property, but not necessarily the first one, the language **B** is known as **NP-Hard**. Informally, a search problem **B** is **NP-Hard** if there exists some **NP-Complete** Problem **A** that Turing reduces to **B**.

**Real-life Examples**

P is the set of decision problems solvable in time polynomial in the size of the input, where time is typically measured in terms of the number of basic mathematical operations performed. An example would be basic multiplication of two numbers. Even just using the typical multiplication algorithm you learn in school to multiply two n digit numbers will only require n² single-digit multiplications, which is a polynomial in n.

NP is the set of decision problems whose solutions can be checked in time polynomial in the size of the input and solution size. An example would again be basic multiplication. You can check that a particular number is the product of two other numbers by simply multiplying those numbers (in polynomial time, as indicated above) and checking that their product is indeed the product you were checking. An example in which checking the answer is faster than finding the answer would be finding the solution to a multivariable system of equations. In this case, one need only substitute the solution values in for the variables in the equations and check that all of the equations are indeed satisfied. Typically, solving such a system is very difficult, if it is even exactly solvable at all.

NP-hard describes the set of problems which are as hard as (or harder than) any problem in NP. In other words, a method which could be used to solve such a problem could be readily and efficiently adapted to solve any other NP problem. An eminently practical example is the Traveling Salesman Problem: Given a bunch of cities on a map and all the roads that connect them, find the shortest route that visits all of the cities at least once. Note that this problem, as stated, is not in NP. There is no efficient way to prove that a proposed solution is indeed the shortest. However, there is a way (which I will not provide) to take any problem in NP and create a map for which the shortest route through the cities is equivalent to a solution to the original problem.