I assume that you are looking for intuitive definitions, since the technical definitions require quite some time to understand. First of all, let's remember a preliminary needed concept to understand those definitions.

* **Decision problem**: A problem with a **yes** or **no** answer.

Now, let us define those *complexity classes*.

**P**

*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 graph connected G, can its vertices be coloured using two colours so that no edge is monochromatic?

Algorithm: start with an arbitrary vertex, color it red and all of its neighbours blue and continue. Stop when you run out of vertices or you are forced to make an edge have both of its endpoints be the same color.

**NP**

*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 factorisation* is in NP. This is the problem that given integers n and m, is there an integer fwith 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.

**NP-Complete**

*NP is a complexity class which represents the set of all problems X for which it is possible to reduce any other NP problem Y to X in polynomial time.*

Intuitively this means that we can solve Y quickly if we know how to solve X quickly. Precisely, Yis reducible to X, if there is a polynomial time algorithm f to transform instances y of Y to instances x = f(y) of X in polynomial time, with the property that the answer to y is yes, if and only if the answer to f(y) is yes.

**Example**

3-SAT. This is the problem wherein we are given a conjunction (ANDs) of 3-clause disjunctions (ORs), statements of the form

(x\_v11 OR x\_v21 OR x\_v31) AND

(x\_v12 OR x\_v22 OR x\_v32) AND

... AND

(x\_v1n OR x\_v2n OR x\_v3n)

where each x\_vij is a boolean variable or the negation of a variable from a finite predefined list (x\_1, x\_2, ... x\_n).

It can be shown that *every NP problem can be reduced to 3-SAT*. The proof of this is technical and requires use of the technical definition of NP (*based on non-deterministic Turing machines*). This is known as *Cook's theorem*.

What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, every NP problem is solvable in polynomial time (one problem to rule them all).

**NP-hard**

Intuitively, these are the problems that are *even harder than the NP-complete problems*. Note that NP-hard problems *do not have to be in NP*, and *they do not have to be decision problems*.

The precise definition here is that *a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time*.

But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

**Example**

The *halting problem* is the classic NP-hard problem. This is the problem that given a program P and input I, will it halt? This is a decision problem but it is not in NP. It is clear that any NP-complete problem can be reduced to this one.

My favorite NP-complete problem is the [Minesweeper problem](http://web.mat.bham.ac.uk/R.W.Kaye/minesw/ordmsw.htm).

**P = NP**

This one of most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences. In fact, the [Clay Institute](http://www.claymath.org/millennium/P_vs_NP/) is offering one million dollars for a solution to the problem (Stephen Cook's [writeup](http://www.claymath.org/millennium/P_vs_NP/pvsnp.pdf) on the Clay website is quite good).

It's clear that P is a subset of NP. The open question is whether or not NP problems have deterministic polynomial time solutions. It is largely believed that they do not. Here is an outstanding recent article on the latest (and the importance) of the P = NP problem: [The Status of the P versus NP problem](http://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext).

The best book on the subject is [Computers and Intractability](http://rads.stackoverflow.com/amzn/click/0716710455) by Garey and Johnson.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Problem Type | Verifiable in P time | Solvable in P time | Increasing Difficulty

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

| P | Yes | Yes | |

| NP | Yes | Yes or No \* | |

| NP-Complete | Yes | Unknown | |

| NP-Hard | Yes or No \*\* | Unknown \*\*\* | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ V

Notes on Yes or No entries:

* \* An NP problem that is also P is solvable in P time.
* \*\* An NP-Hard problem that is also NP-Complete is verifiable in P time.
* \*\*\* NP-Complete problems (all of which form a subset of NP-hard) might be. The rest of NP hard is not.

***A decision problem is a collection of questions with yes or no answers that vary only in one parameter.***

There are many problems for which no polynomial-time algorithms ins known. Some of these problems are traveling salesperson, optimal graph coloring, the knapsack problem, Hamiltonian cycles, integer programming, finding the longest simple path in a graph, and satisfying a Boolean formula.

These problems belongs to an interesting class of problems, called the NP-Complete" problems, whose status is unknown.

The NP-Complete problems are tractable i.e., require a superpolynomial time. The reason is that a polynomial time algorithm to solve any one of the NP-Complete problems would automatically provide us with polynomial-time algorithms for all of them.

The basic idea is that there may be problems that are hard to solve, but the validity of any purported solution can be verified easily.

Consider the problem "Hamiltonian Cycle". Hence an undirected graph, the problem is to find a path that starts from some node, visits each node once and only once, and returns to the starting node. If such cycle exists, we say that the graph is Hamiltonian. This problem is hard. However, it is easy to verify whether a "given" sequence of nodes defines a Hamiltonian cycle.

**Polynomial-Time Algorithm**

1. Algorithms with worst case running time of O(nk), where k is a constant, are called tractable others are called intractable or super-polynomial.
2. 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)).
3. Problem requiring Ω(n35) 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 with at most a polynomial slow-down.

### Decision Problem

For these problems, the answer is either yes or no or equivalently either true or false.  
For example, the problem "Find a Hamiltonian  cycle in graph G" is not a decision problem, but "Is  graph G Hamiltonian?" is a decision problem.

The theory of NP-Completeness is concerned with the notion of polynomially varifiable properties. Intuitively, a decision problem X is polynomial-time verifiable if someone could show that *x* in X whenever this is so. Given this example (i.e., *x* in X), one should be able to verify in polynomial-time (efficiently or easily) that indeed *x* in X. However, if in fact *x* not in X, then one should not be falsely show that *x* in X.