**NATIONAL INSTITUTE OF TECHNOLOGY, DELHI**



**ASSIGNMENT – P and NP problems**

NAME : MAYANK

ROLL NO. : 181210029

CSE 2ND YEAR

**P and NP introduction**

**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 connected graph 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.

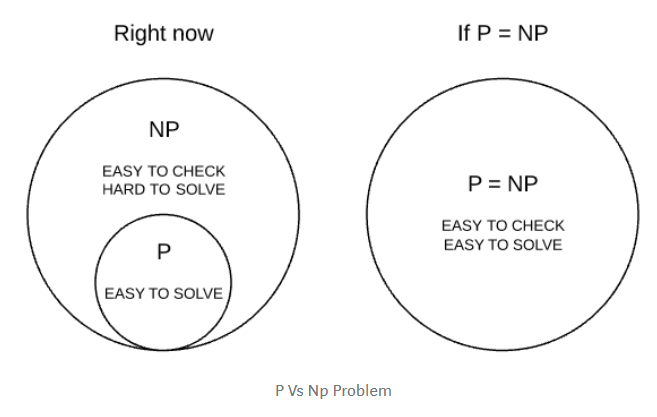
This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.

**Example**

Integer 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.

Diagram :-



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

NP hard and NP complete problems

**NP hard :-**

A problem is **NP-hard** if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem.

Or

that are **at least** as hard as NP problem hardest part is called "NP-hard".

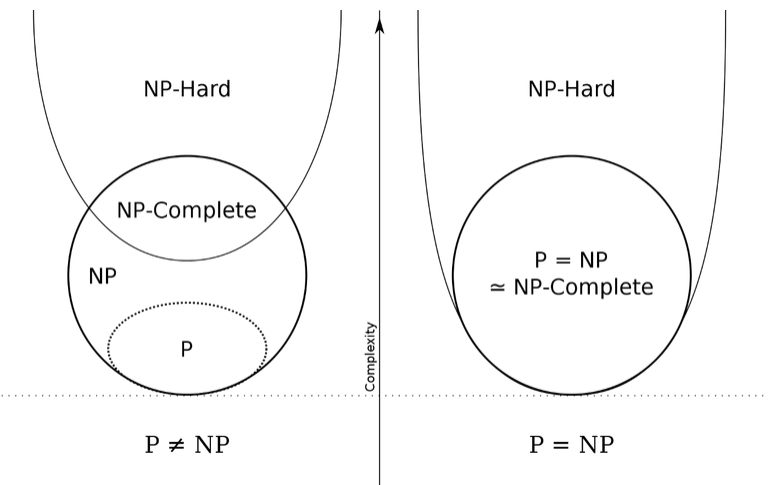
**NP complete :-**

A group of problem where if a fast solution to **any of one** the problem is found we can solve a group of problem in same set of complexity with ease

Or

NP-complete means that these problems include all the really **hard parts** of every NP problem.

Diagram :-



**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.