**NATIONAL INSTITUTE OF TECHNOLOGY DELHI**



**DESIGN AND ANALYSIS OF ALGORITHMS**

**ASSIGNMENT 5**

**TOPIC : “P VS NP”**

**Submitted by :**

**Name : AKI SRI CHARAN**

**Roll number : 181210007**

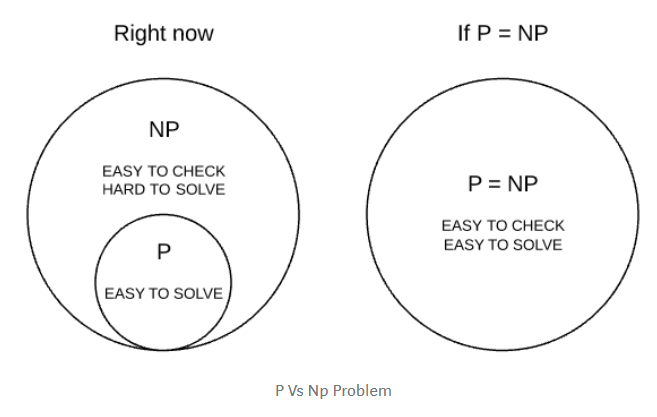
**Cse 2nd year**

**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**.

**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. Examples for this class of problems are Sudoku problem, Knapsack problem, Travelling Salesperson Problem, etc.

****

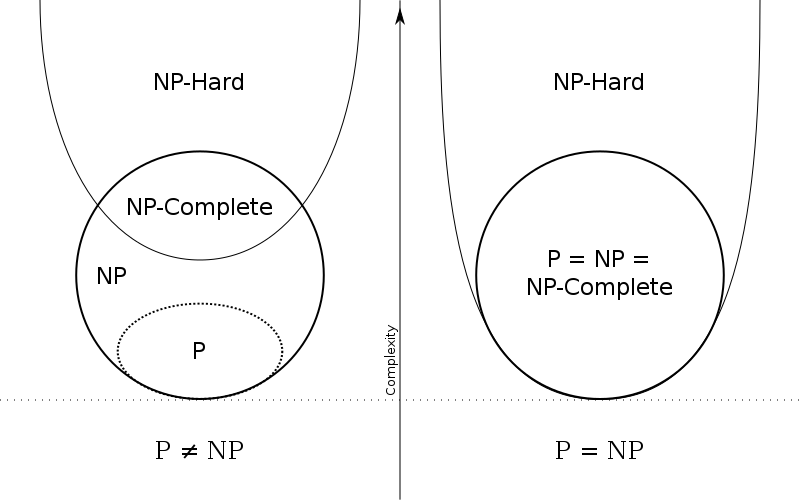
**NP HARD:**

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:**

**NP-complete** is a special category of **NP problems** that have time complexities greater than polynomial time, are verifiable in polynomial time, and belong to a set of problems known as **NP-hard**. **NP-hard** problems are essentially those that are at least as hard as the hardest **NP problem**, but don’t need to be verifiable in polynomial time.

**P VS NP:**

**P VS NP 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. Researchers have shown that many of the problems in **NP** have the extra property that a fast solution to any one of them could be used to build a quick solution to any other problem in **NP**, a property called **NP** completeness. Decades of searching have not yielded a fast solution to any of these problems, so most scientists suspect that none of these problems can be solved quickly. This, however, has never been proven.