**NATIONAL INSTITUTE OF TECHNOLOGY, DELHI**

**ASSIGNMENT 5**

**DESIGN AND ANALYSIS OF ALGORITHMS**

Submitted By : Japman Singh Monga

Roll No. : - 181210024

BRANCH : CSE 2nd year

**P and NP PROBLEMS**

**Definition**

**P** (polynomial time) refers to the class of problems that can be solved by an algorithm in polynomial time. Problems in the **P** class can range from anything as simple as multiplication to finding the largest number in a list. They are the relatively ‘easier’ set of problems.

**NP** (non-deterministic polynomial time) refers to the class of problems that can be solved in polynomial time by a non-deterministic computer. This is essentially another way of saying “If I have unlimited compute power (i.e. as many computers as I need), I am capable of solving any problem in at most polynomial time”. More intuitively though, it refers to the class of problems that currently, has no way of finding a quick (polynomial time) enough answer, BUT can be quickly **verified** (in polynomial time) if one provides the solution to the problem. The term verified here means that one is able to check that the solution provided is indeed correct.

Clearly, based on the definition above, **P ⊆ NP**.

**P vs NP**

* P is a class that basically includes all the problems that can be solved by a reasonably fast program.

1. Minimum spanning tree problems using prim’s and Kruskal algorithms.

1. Fractional knapsack problem.

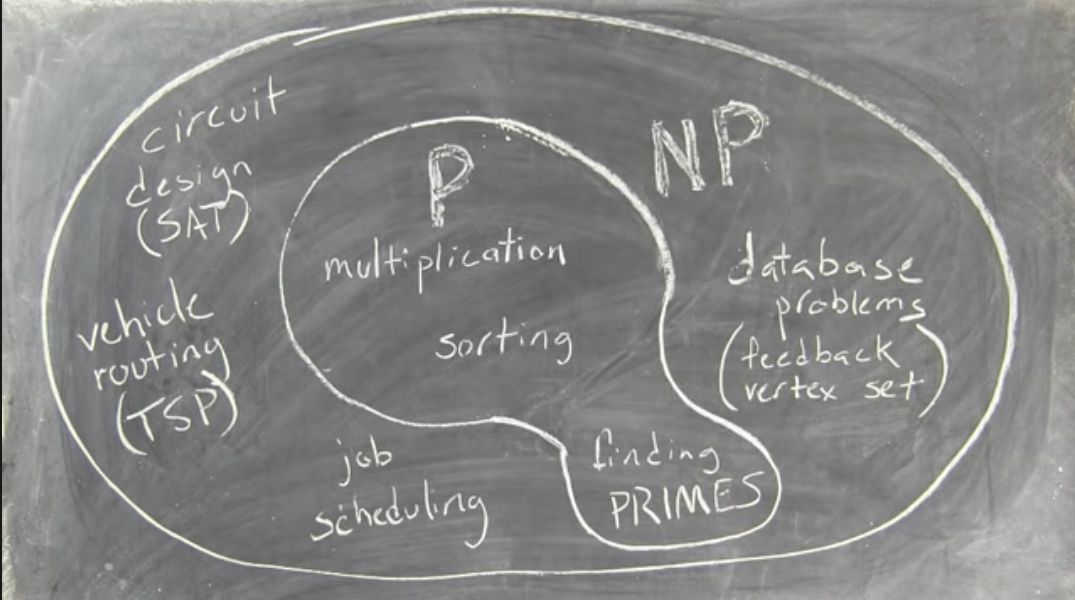
* Around and including P, there is a class called NP.
* That is all the problem where, if you’re given a correct solution you can at least check it in a reasonable amount of time.

1. If we are able to solve a problem in polynomial time, we will

surely be able to verify in polynomial time, so every P problem will also be a NP problem.

1. Travelling salesman problem :- we are not able to find polynomial time solution for this problem but we can verify this in polynomial time so this a NP problem but not a P problem.

* Often, we find that an NP problem was actually a part of P and we would have a fast program.



**Polynomial time reduction algorithm :-**

To prove P = NP we have to prove that every problem which lies in NP can be solved in polynomial time

There are millions of NP problems we can not solve each problem to prove this, here comes reduction.

A problem ‘A’ is said to be polynomial time reducible to a problem ‘B’ if :-

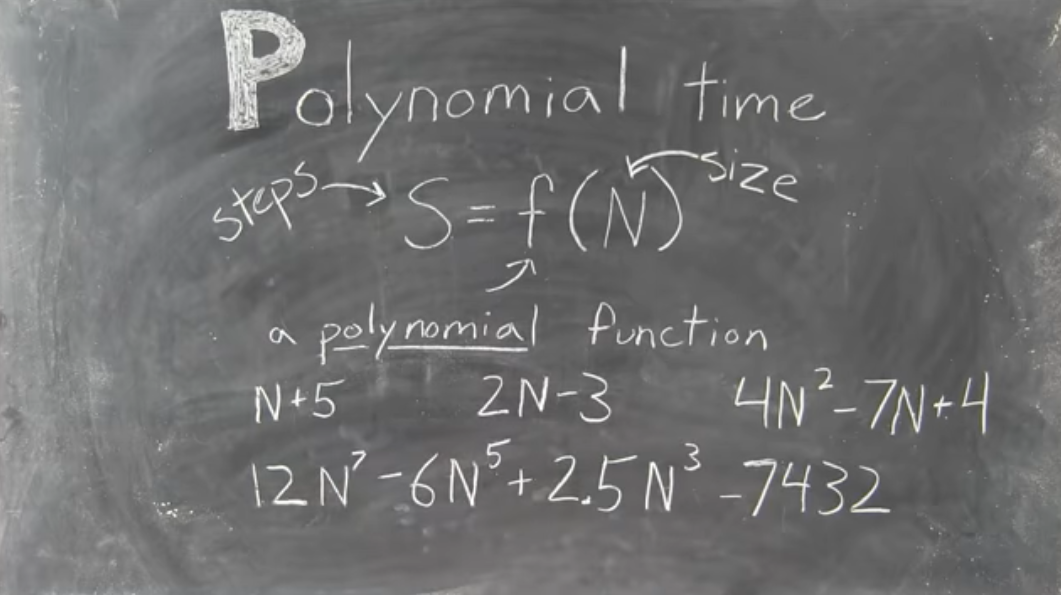
1. Every instance ‘a’ of ‘A’ can be transformed to some instance ‘b’ of ‘B’ in polynomial time.
2. Answer of ‘a’ is ‘YES’ if and only if answer of ‘b’ is ‘YES’.

So

If A is reduced to B in polynomial time then :-

* If B is easy then A is also easy.
* If B is in P then A is also in P.
* If this is proven that A cannot be solved in polynomial time then B is also cannot be solved in polynomial time.
* If A is not in P then B is also not in P

**P Type:**



**NP Type:-**

**NP Complete Problems:**

* After a long time researchers realized that dozens of NP problems that they were struggling with were essentially all the same problem with some easy polynomial time complications thrown here and there.

Eg. Sudoku, protein folding, puzzles etc.

* It consists of really hard parts of NP Problem.

**NP Hard:**

* The problems that are at least as hard as NP complete is also known as NP Hard type of problems.

**Co-NP:**

* The class of problems where instead of being easy to check the right answers, it is easy to exclude wrong answers, which may or may not be the same of NP.

**Real Life Example-Sudoku**

One of the most common yet effective examples is Sudoku. Given an unsolved Sudoku grid (9 x 9 for example), it would take an algorithm a fair amount of time to solve one. However, if the 9 x 9 grid increases to a 100 x 100 or 10,000 x 10,000 grid, the time it would take to solve it would increase **exponentially** because the problem itself becomes significantly harder. However, given a solved Sudoku grid (of 9 x 9), it is fairly straightforward to verify that the particular solution is indeed correct even if the size scales to 10,000 by 10,000. It would be slower, but the time to check a solution increases at a slower rate (**polynomially**). So, Sudoku is in NP (quickly checkable) but does not seem to be in P (quickly solvable).