**NATIONAL INSTITUTE OF TECHNOLOGY, DELHI**

**ASSIGNMENT**

**DESIGN AND ANALYSIS OF ALGORITHMS**

**P and NP PROBLEMS**



Name : RAGHAV SHUKLA

Roll. No. : 181210038

BRANCH : CSE 2nd year(group-II)

P Versus NP Problem

*Problems that we known an efficient algorithm for that is capable of producing a solution in polynomial time are classified as****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****.*

*Later, we found another subset of problems that****P****itself was a subset of,****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.*

## P-Class

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*.

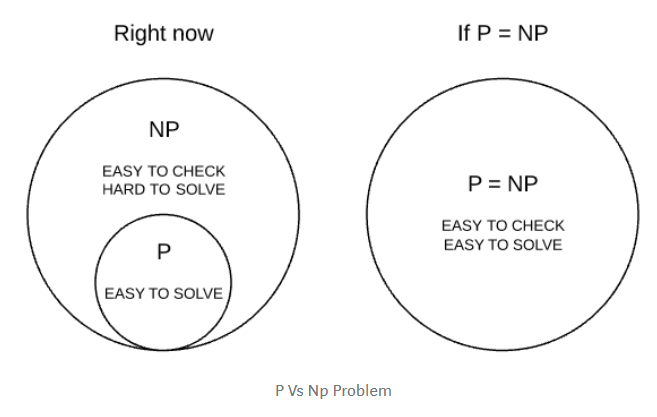
The advantages in considering the class of polynomial-time algorithms is that all reasonable deterministic single processor model of computation can be simulated on each other with at most a polynomial slow-d.

## NP-Class

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.

* P is a subset of NP.
* Both classes of problems can be at least verified/checked in polynomial time.
* For P, computing the correct solution to the given problem can be done in polynomial time. Whereas for NP, there is no algorithm that can produce the solution in polynomial time.
* Given a problem which belongs to NP and a possible solution to the said problem, it can be easily verified if that is a correct solution or not within reasonable polynomial time.



By the term “deterministic”, we mean that we know the working of every step and statement of the algorithm used clearly.

## NP-Complete and NP-Hard

Amongst these NP problems, there exists a King of all problems which researchers call NP-Complete problems. Formally, they are a set of problems to each of which any other NP problem can be **reduced** (addressed below) in polynomial time and whose solution may still be verified in polynomial time. This means that any NP problem can be transformed into a NP-Complete problem.

Informally, they are the “hardest” of the NP problems. Thus if any one NP-Complete problem can be solved in polynomial time, then every NP-Complete problem can be solved in polynomial time, and every problem in NP can be solved in polynomial time (i.e. P=NP). The most famous example would be the Traveling Salesmen problem.

There also exists a set of problems called NP-Hard problems. These problems are at least as hard as NP problems, but **without**the condition that requires it to be solved in polynomial time. This suggests that NP-Hard problems may not necessarily be part of the NP class. An example would be solving a chess board — given a state of a chess board, it is almost impossible to tell if a given move at the given state, is in fact the optimal move. Formally, there exists no polynomial time algorithm to verify a solution to a NP-Hard problem.

If we put the two together, a NP-Complete problem implies it being NP-Hard, but a NP-Hard problem does NOT imply it being NP-Complete.

## Defining NP-Completeness

A problem L, is NP-Complete if:

1. L is NP-Hard
2. L belongs to NP

The diagram below (focus on the left-hand side) should make things clearer.

