NATIONAL INSTITUTE OF TECHNOLOGY DELHI

DESIGN & ANALYSIS OF ALGORITHMS - CSB 252

Name	:	HIMANSHU SRIVASTAVA
Roll no	:	181220031
Class	:	CSE 2nd Year

[A]. Introduction to P vs. NP problem

The P versus NP problem is to determine whether every language accepted by some non-deterministic algorithm in polynomial time is also accepted by some deterministic algorithm in polynomial time.

A.1 :: WHAT IS 'P' ?

- P (stands for *Deterministic Polynomial Time*), is the class of problems for which we have an efficient algorithm that is capable of producing a solution in polynomial time.

- When we know each and every statement of the algorithm, clearly and their working, we call it deterministic.

For instance, *multiplication of two numbers*, we can verify the result and solve it easily.

A.2 :: WHAT IS 'NP'?

- NP (stands for Non-Deterministic Polynomial Time), is the set of decision problems that are solvable in polynomial time by a Non-deterministic Turing Machine.

- NP problems do not have a know algorithm that can produce a result in polynomial time. If we are given a solution to an NP problem, verifying that it is correct is easy and can be done in polynomial time or less. They are easy to check but hard to solve, they generally have an exponential time complexity such as $O(n^n)$.

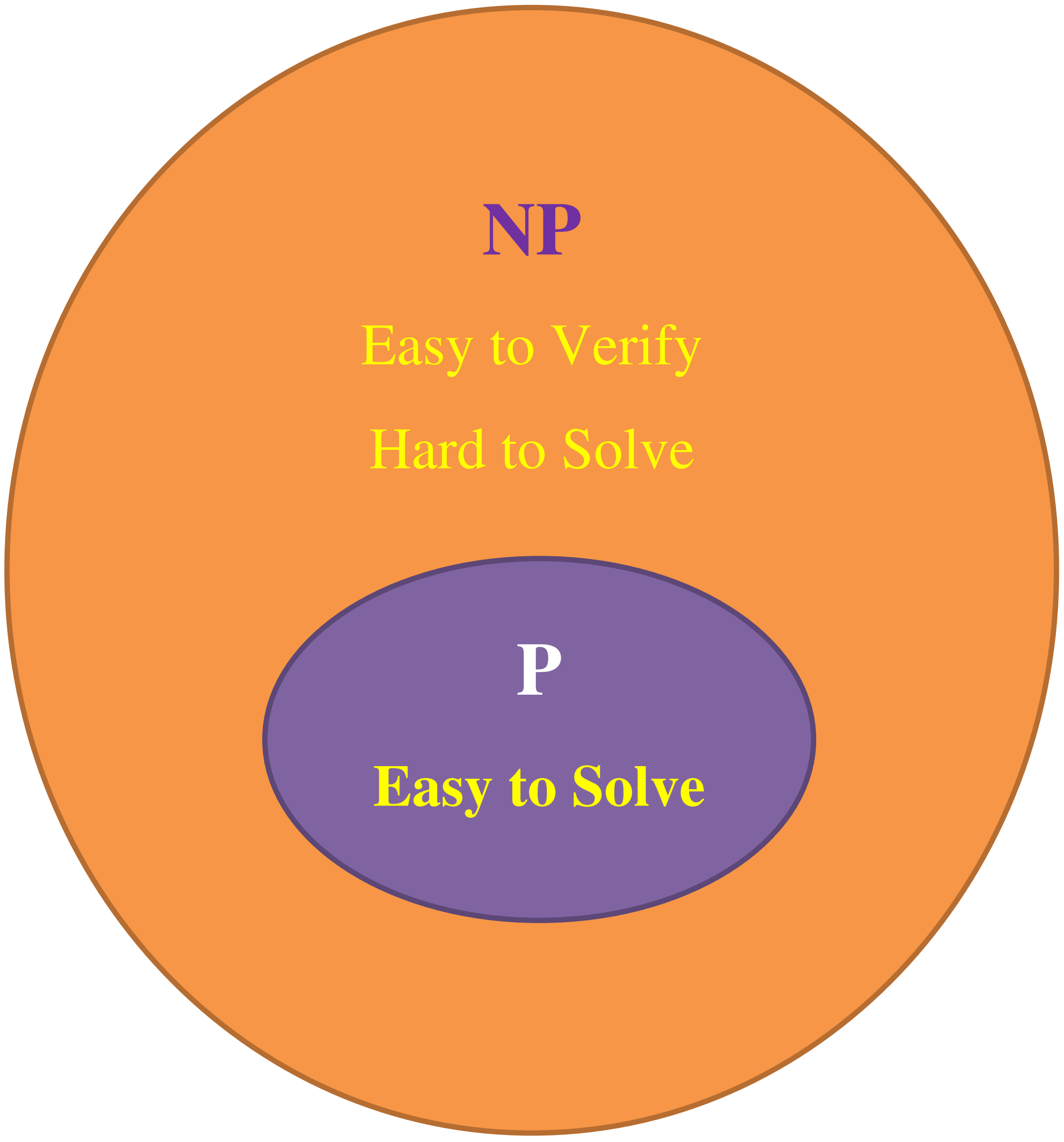
Example:

- ❖ Solving a Sudoku Puzzle
- ❖ Unlocking a Smartphone
- ❖ Sending messages over apps, etc.

Problems are like :

- ❖ 0/1 Knapsack, Travelling Salesman
- ❖ Sum of subsets
- ❖ Graph Colouring. Hamiltonian Cycle
- ❖ Finding Prime Factors of a large number take exponential time to solve.

A.3 :: DIAGRAMATIC REPRESENTATION



P is a subset of NP

[B]. P vs. NP Problem

It is one of the most famous unsolved problem in Computer Science field, and also one the millennial problem.

Proving P=NP problem would have profound implication in the various fields. We have seen the definition in the introduction.

Let's take a deep dive to understand this problem better! The problem in layman's term can be stated as: -

"Can every problem whose solution can be quickly verified by a computer also be quickly solved by a computer?"

B.1 :: SATISFIABILITY

To Relate every problem Y ∈ NP, we need a problem as a base problem, that base problem is the satisfiability formula.

It is a CNF Formula (Propositional Calculus), using Boolean variables:

$$(x_1 \vee x_3 \vee \bar{x_6}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_7}) \wedge \ldots$$

B.2 :: BOOLEAN SATISFIABILITY PROBLEM

- The problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE.

- If we can find an interpretation that can make the entire formula true, then the Boolean formula is called satisfiable, if not, unsatisfiable.

- The computational time of Boolean satisfiability comes to be similar to that of other NP problems. Now, if we can show that satisfiability is solved in polynomial time then we can relate that all other NP problems can be solved in polynomial time. This Satisfiability acts as a **NP-Hard problem**.

B.2 :: WHY ARE WE DOING THIS ?

We are relating all the NP problems so that we don't have to solve each NP problem individually. Satisfiability problem is directly proportional to NP problem. So, if we show satisfiability is NP Hard or NP Complete.

Then for $Y \in$ NP, is also NP Hard or NP Complete respectively, and this implies for the relation amongst other NP problems too.

B.3 :: REDUCTION

It's the procedure to relate all other NP problems to our base problem, in other words, a process of converting the inputs of a problem A into equivalent inputs of a problem B using a polynomial time algorithm. Equivalent means both the problem A and B must output the same answer (Yes or No) for the input and the converted input. The existence of a reduction algorithm from A to B will imply the following : -

- If B ∈ P, then A ∈ P (You can just reduce from A to B in polynomial time and solve B in polynomial time. Combining this gives polynomial time algorithm for A)
- If B ∈ NP, then A ∈ NP.
- If A is NP-hard, then B is NP-hard. A can be reduced to B in polynomial time and if B is not NP-hard then B ∈ NP-NP-hard and it will mean that A ∈ NP-NP-hard which contradicts with the hypothesis (A is NP-hard).

B.4 :: NP HARD

A problem X is NP-hard if every problem Y ∈ NP ***reduces*** to X i.e. X is at least as hard to solve as every problem in NP

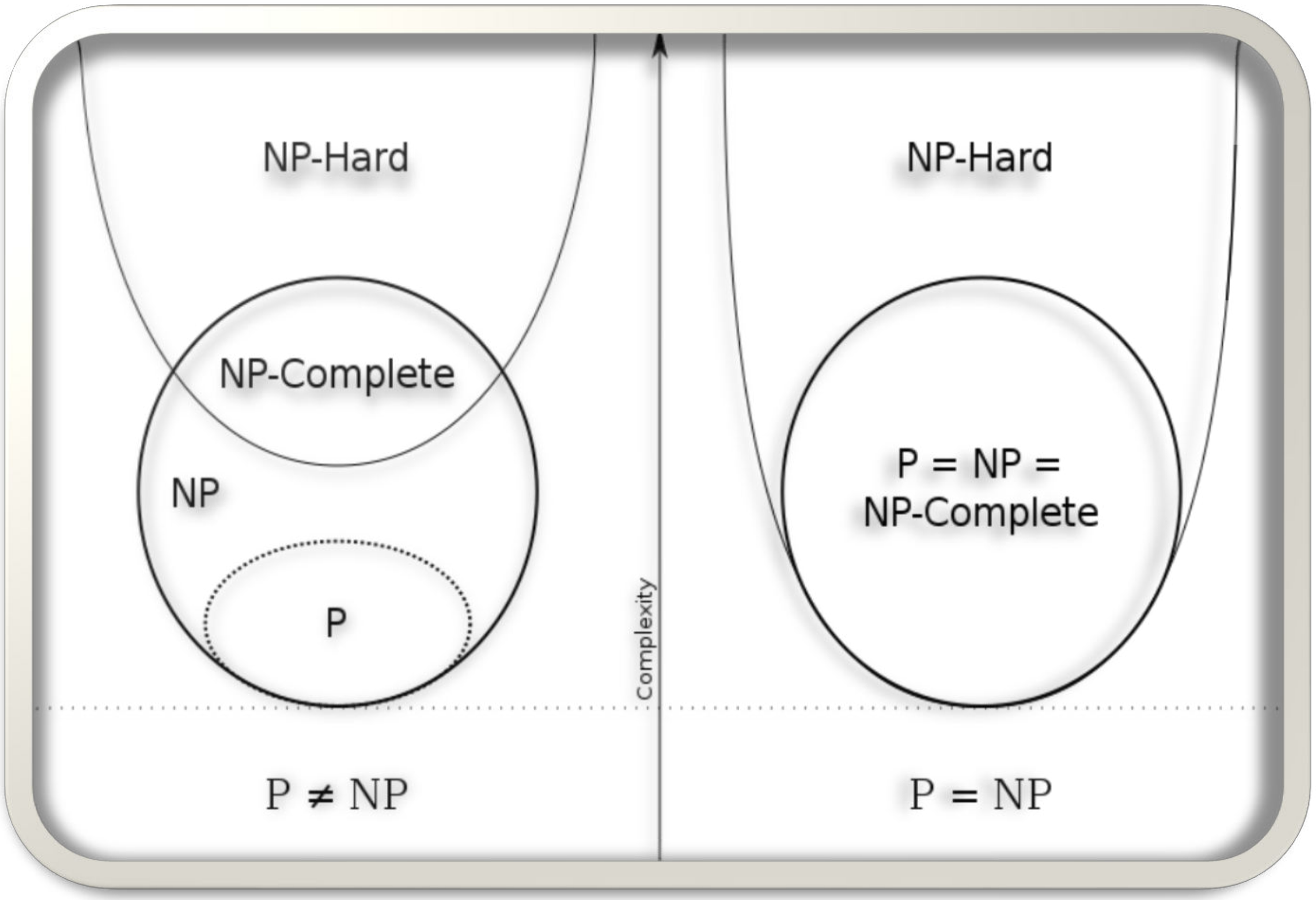
(If P! = NP, then X does not belong to P)

B.5 :: NP-COMPLETE

A problem L is NP-complete if :-

- L is in P.
- an algorithm which solves L can be used to solve any problem L' in NP; that is, given an instance of L' we can create an instance of L that has a solution iff the instance of L' has a solution.

Formally speaking, every problem L' in NP is reducible to L.



[C]. DIFFERENCE BETWEEN P & NP

C.1 :: ADDITION OF TWO NUMBERS

When the digits are less it is easy for us to compute, but the larger the numbers get the harder the computation becomes for us humans. But the computers can add larger numbers in a fairly simple amount of polynomial time quickly and efficiently.

Such type of problems which can be solved by a computer are known as **P problems.**

C.2 :: PRIME FACTORIZATION

We know that, every composite number can be expressed as a product of two or more prime factors. Our normal computers can handle this problem within seconds for numbers up to a billion. But as the numbers grow this problem becomes lot harder even for the fastest of computers.

So, this is not solvable in Polynomial time.

- But what if I provide the factors?
- Can we verify the solution?

Like if I say, 50 can be factorized as **2*5*5**. We can easily verify it by multiplying 2,5 and 5 real quick and find that it indeed produces 50.

In fact, multiplication is a P problem. Computers can multiply fairly large numbers in no time. So, factorization is not solvable in polynomial time but the solution is verifiable in polynomial time.

These type of problems are known as **NP problems.**

S No.	P PROBLEMS	NP PROBLEMS
1.	These can be solved in polynomial time by deterministic algorithms.	These can be solved in non-deterministic polynomial time.
2.	Such problems can be solved and verified in polynomial time.	NP problems solution cannot be obtained in polynomial time but if solution is given it can be verified in polynomial time.
3.	P problems are subset of NP problems.	NP problems are a superset of P problems.
4.	e.g. Searching, Sorting, Addition, Multiplication, etc.	e.g. Sudoku Puzzle ,Travelling Salesman Problem, etc.

[D]. REAL LIFE EXAMPLES

D.1 :: HOSTEL ACCOMMODATION

Suppose that a university is organizing accommodations in hostel for a group of four hundred university students.

Space is limited and only one hundred of the students will receive places in the dormitory.

To complicate matters, the Dean has provided a list of pairs of incompatible students, and requested that no pair from this list appear in the final choice.

D.2 :: BOXING WATERMELONS

A farmer wants to take 200 watermelons of different masses to the market. He needs to pack the watermelons into boxes.

Each box can only hold 30 kilograms without breaking. The farmer needs to know if 10 boxes will be enough for her to carry all 200 watermelons to market.

D.3 :: SUPER MARIO GAME

The vintage Super Mario game by Nintendo Inc. is itself a NP-hard.