Computer Science & Engineering Department
National Institute of Technology, Delhi

Design and Analysis of Algorithms
P and NP Problems

Submitted by: Namrata Prasad
Roll Number: 181210032
Branch: CSE 2nd Year, Group 1

P and NP Problems

In the field of computer science, especially algorithms, humans are constantly thriving to get accurate results of problems using as minimum time as possible and continuously trying to improve the already available algorithms to improve the time complexity.

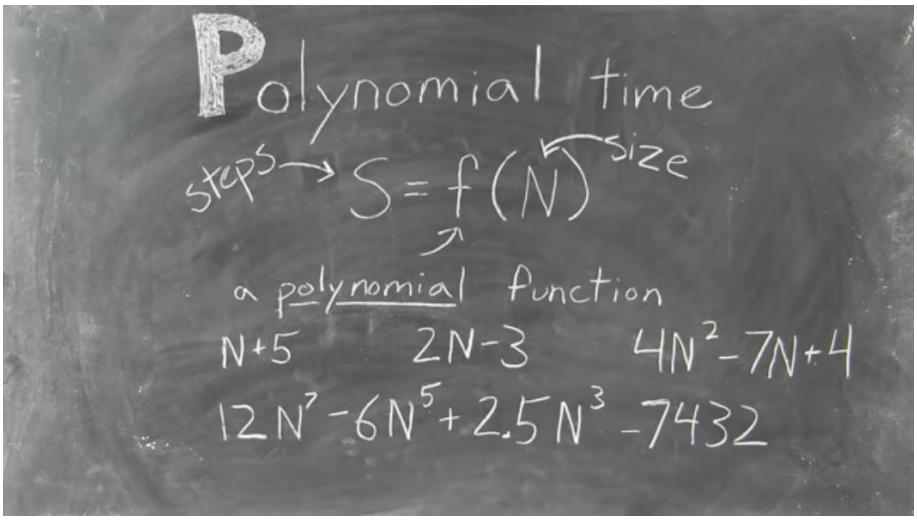
The computational complexity of a problem is the minimum of the complexities of all possible algorithms for this problem.

This introduces us to the **P and NP** class of problems.

Most of the problems in computational space can be reduced to a **decision problem**. That means problems where the answer is either YES or NO.

Both P and NP can be considered as a set of problems which are grouped based on how difficult it is to solve and evaluate the solution.

P (polynomial time) contains all decision problems that can be solved by a deterministic Turing machine(DTM) using a polynomial amount of computation time, or polynomial time.



For example: searching algorithms, sorting algorithms, matrix multiplication, optimal merge pattern etc

Then there is another set of problems which can be verified in polynomial time but, in order to solve this problem it will take a lot more time.

For example, let's take Sudoku. Given that we have a solution for any game, we can verify it easily. This means that we can do the verification part in polynomial time. But in order to solve the puzzle, we need more than polynomial time. Also as the number of grids increase, the complexity of finding a solution increases exponentially.

NP (non-deterministic polynomial time) is a complexity class which contains the set of decision problems for which the problem instances have proofs verifiable in polynomial time but are not actually solvable in polynomial time.
Equivalently, it could also be said that NP is the set of decision problems that are solvable in non deterministic polynomial time i.e can be solved in polynomial time by a Non-deterministic Turing Machine

Some more examples of NP problems are:

1. Suppose that you are organizing housing accommodations 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 you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe!

2. A traveling salesman wants to visit 100 different cities by driving, starting and ending his trip at home. He has a limited supply of gasoline, so he can only drive a total of 10,000 kilometers. He wants to know if he can visit all of the cities without running out of gasoline.

3. A farmer wants to take 100 watermelons of different masses to the market. She needs to pack the watermelons into boxes. Each box can only hold 20 kilograms without breaking. The farmer needs to know if 10 boxes will be enough for her to carry all 100 watermelons to market.

4. Graph coloring which is used by compilers can also be declared within NP class.

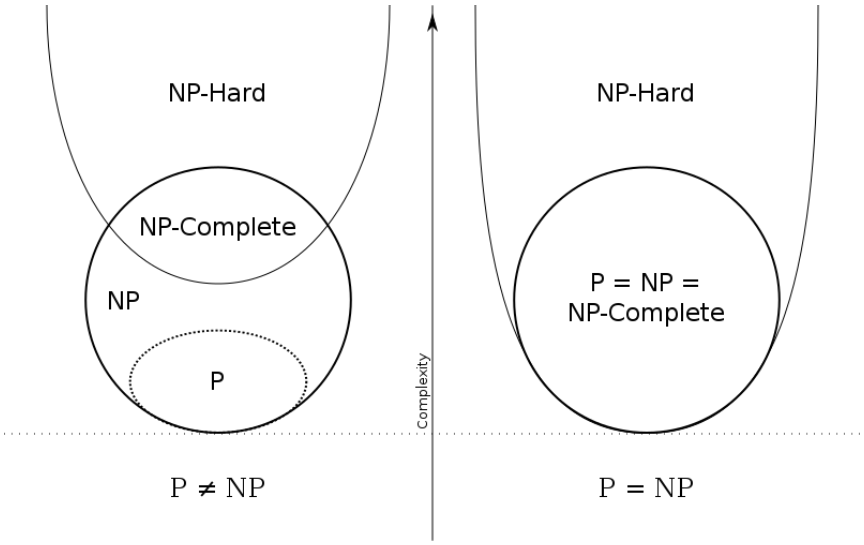
NP-HARD

A problem is **NP-hard** if an algorithm for solving it can be translated into one for solving any NP-problem (non-deterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.
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)

Reduction — 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.

NP-COMPLETE

A problem is **NP-complete** when it can be solved by a restricted class of brute force search algorithms and it can be used to simulate any other problem with a similar algorithm.
A problem X is NP-Complete if X ∈ NP and X is NP-hard.



The halting problem is an NP-hard problem. This is the problem that given a program P and input I, will it halt? This is a decision problem but it is not in NP. It is clear that any NP-complete problem can be reduced to this one. As another example, any NP-complete problem is NP-hard.
The Minesweeper problem is an NP-complete problem.

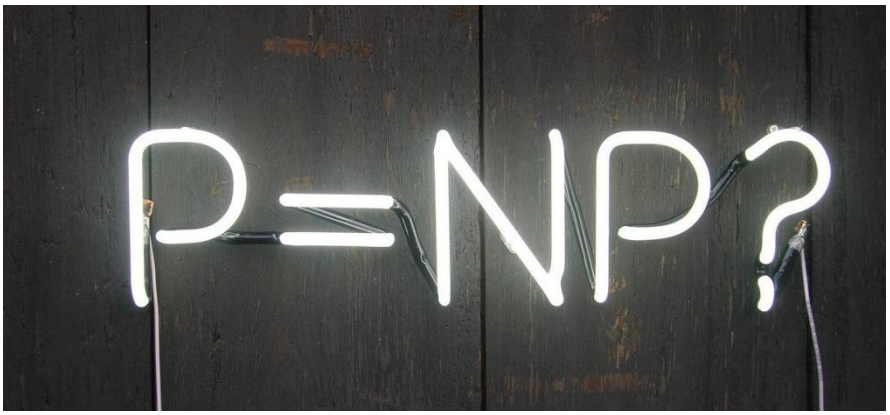
THE P vs NP PROBLEM

It is important to realize that all P problems also belong to NP because if a problem is solvable by DTM in polynomial time, verifying a potential solution will be easier than solving. So a DTM would also be able to verify in polynomial time. So trivially, **P ⊆ NP i.e P is a subset of NP.**

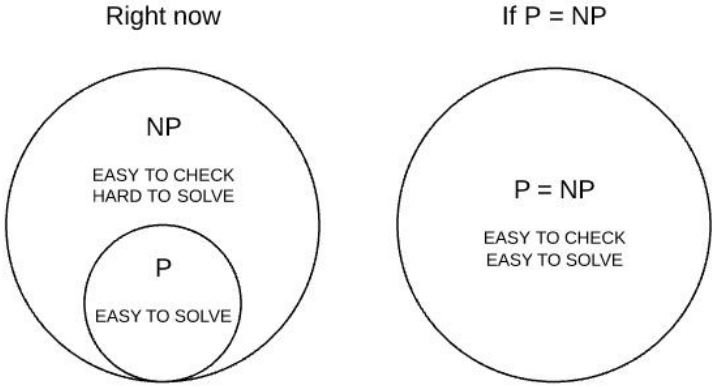
Now the question arises that - **Does equality holds?**

This is the famous P vs NP problem which states :

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



P.S.-- It is one of the 7 Millennium Prize Problems selected by the Clay Mathematics Institute to carry a 1 million dollar prize for the first correct solution and is still open.



If P=NP were true, it would have profound effects on society.

> If P = NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found.
>
> — *Scott Aaronson, UT Austin*

It would mean online security systems would all be vulnerable to attacks because our current inability to efficiently factor huge numbers and NP problem forms the basis of modern cryptography which everything - from national security to banking depends on.

It would also mean that we could make everything more efficient.

- ✓ Transportation schedules
- ✓ Production cycles for manufactures
- ✓ **Protein folding simulations which would effectively cure cancer!!!**

Along with that, there would be some other benefits as well such as:

Compilers: Since compilers use graph coloring for register allocation, we would be able to allocate for large numbers of registers exactly. Existing compilers using an approximate solution (like chordal graphs) would get better output, and those using an exact solution would get faster.

Buying plane tickets: Airline tickets are weird since they don't follow triangle equality. Sometimes it's cheaper to fly from A -> B -> C than directly from A -> C, something that doesn't come up when modelling distances. It would be easy to make a website that finds the absolute cheapest sequence of flights that visit some number of cities and starts and ends in your hometown.

Scheduling: Mad that your school put two of your exams on the same time? If P=NP your school could either know how many time slots they need so no student has a conflict, or given a number of time slots, minimize the number of conflicts.

On the other side, if there was someone who proved that P $\neq$ NP, it would prove that there are some problems which no amount of data,expertise or intuition could solve!

BEYOND NP PROBLEMS-
THE COMPUTATIONAL COMPLEXITY ZOO

There are problems outside NP which are hard to make a solution at the same time hard to check the solution as well, for example, consider chess. Given any position on a board it is hard to find the best next move also it is hard to verify the next move to be accurate or not This problem is in EXP (Exponential time complexity)and maybe outside of NP.