P, NP, NP-Hard, NP-Complete:

P: The class of problems that have polynomial-time deterministic algorithms. Eg. MST, shortest path, Sorting, Fractional Knapsack etc. These are also called tractable problems

NP: The class of decision problems that are solvable in polynomial time on a nondeterministic machine (or with a nondeterministic algorithm). A nondeterministic computer can be considered as a parallel machine that can freely spawn an infinite number of processes. The solution of the problems of NP class can be verified in polynomial time. NP stands for "Nondeterministic Polynomial-time".

NP-hard: The class of problems which are at least as hard as the hardest problems in NP. Problems that

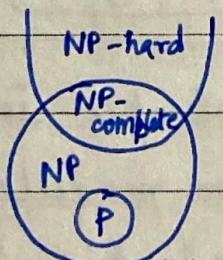
are NP-hard do not have to be elements of NP. It may or may not have polynomial-time verifiable solutions. Any problems in NP can be reduced to an NP-hard problem in polynomial time. Therefore, if a problem of NP-hard can be solved by DTM in polynomial time ever, all NP problems also will be solved. (P=NP ?)

NP-complete: A problem P is NP-complete, if and only if P belongs to NP and every problem in NP is reducible to P in polynomial time. P can be shown to be in NP by demonstrating that a solution of P can be verified in polynomial time. If we had a polynomial time algorithm for P, we could solve all problems in NP in polynomial time. NP-complete problems represent the hardest problems in NP with respect to polynomial time reductions. A problem is considered NP-complete if it satisfies two conditions:

① It is in the class NP, which means a proposed solution can be verified in polynomial time and it has a solution in non-deterministic polynomial time.

② It is NP-hard, which means it is at least as hard as the hardest problems in NP. This implies that any problem in NP can be reduced to a NP-complete problem in polynomial time. And if one problem can be solved in polynomial time, then all NP problems can be

solved in polynomial time. However, no polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has anyone yet been able to prove a super polynomial-time lower bound for any of them. This so-called $P \neq NP$ question is most perplexing open research problems in theoretical computer science. Most scientists believe that the NP-complete problems are intractable. The reason is that if any single NP-complete problem can be solved in polynomial time, then every NP-complete problem has a polynomial-time algorithm. Given the wide range of NP-complete problems that have been studied to date, without any progress toward a polynomial-time solution, it would be truly astounding if all of them could be solved in polynomial-time.



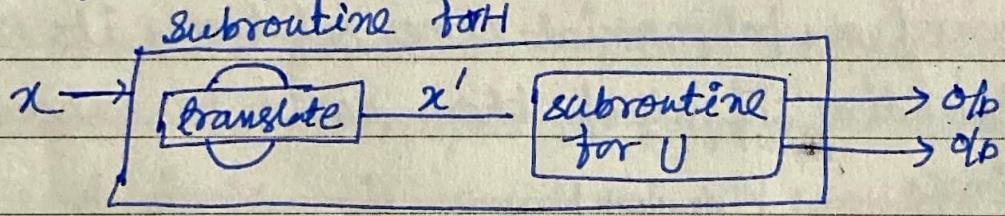
Conjunctive Normal Form & Satisfiability Problem

3-CNF and 3-SAT:

3-CNF: A finite set of binary variables $X = \{x_1, x_2, \cdots, x_n\}$. and a collection of clauses $C = \{c_1, c_2, \cdots, c_m\}$ on X such that $|c_j| = 3$ of $1 \leq j \leq m$.

3-SAT: Given a 3-CNF formula, 3-SAT finds the values of the Boolean variables that make the formula true or return a message that no such values exist.

Reduction: Suppose that there are two problems, H and U. We know that H is hard, that is it can't be solved in polynomial time. On the other hand, the complexity of U is unknown. We want to prove that U is also hard. To show that U is not solvable in polynomial time, we assume that a polynomial time algorithm for U exists (contradiction), and then we will use this algorithm to solve H in polynomial time, if H can be mapped in U. But H is hard so our assumption is wrong and U is also hard. Suppose, that we have a subroutine that can solve any instance of problem U in polynomial time. Given an input x for the problem H, we could translate it into an equivalent input x' for U. The following diagram explains it.



It is easy to see that if U is solvable in polynomial time, then so is H. The translation module runs in polynomial time. Therefore, a polynomial reduction of problem H is problem U is done, which is denoted $H \leq_p U$.