Up to now the notion of algorithm that we have been using has the property that the result of every operation is uniquely defined. Algorithms with this property are termed *deterministic algorithms*.

A deterministic algorithm is one where, given a particular input, the algorithm will always produce the same output and follow the same sequence of states. It operates under the principle of predictability and consistency. The outcome of a deterministic algorithm can be precisely determined based on its input.

Characteristics of Deterministic Algorithms:

•**Predictability**: The same input will always result in the same output.

•**Fixed Steps:** The algorithm follows a clear set of rules or steps.

•**No Randomness:** There is no element of randomness or probability in the process.

•**Easy to Debug**: Predictable behavior makes them easier to test and debug.

Applications of Deterministic Algorithms:

•**Sorting and Searching**: Algorithms like QuickSort, MergeSort, and Binary Search.

•**Mathematical Calculations:** Algorithms used in calculating functions in mathematics and physics.

•**Data Compression**: Algorithms like Huffman coding and Run-length encoding.

•**Cryptography**: Algorithms for encryption and decryption like AES (Advanced Encryption Standard).

What is a Non-Deterministic Algorithm?

A non-deterministic algorithm is one where the same input can lead to multiple possible outcomes. Unlike deterministic algorithms, these do not follow a single clear path through execution. Instead, they explore various paths or possibilities, often simultaneously.

Characteristics of Non-Deterministic Algorithms:

•**Multiple Possible Outcomes:** The same input might result in different outputs on different runs.

•**Probabilistic Elements**: Often involves randomness or probability in decision-making.

•**Parallel Path Exploration**: Can explore multiple solution paths simultaneously.

•**Complexity:** Generally more complex than deterministic algorithms.

algorithms to contain operations whose outcomes are not uniquely defined but are limited to specified sets of possibilities. The machine executing such operations is allowed to choose any one of these outcomes subject to a termination condition to be defined later. This leads to the concept of a *nondeterministic algorithm*. To specify such algorithms, we introduce three

Any problem for which the answer is either zero or one is called a *decision problem*. An algorithm for a decision problem is termed

a *decision algorithm*. Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an *optimization problem*. An *optimization algorithm* is used to solve an optimization problem. □

Definition of Decision Based Problem: - A problem is called a decision problem if its output is a simple "yes" or "no" (or you may need this of this as true/false, 0/1, accept/reject.) We will phrase many optimization problems as decision problems. For example, Greedy method, D.P., given a graph G= (V, E) if there exists any Hamiltonian cycle.

- The problem whether a given number is odd (or even).
- The problem whether a given number is a prime number.
- The problem whether a given given number is in a specified finite or cofinite subset of natural numbers.

Definition of P class Problem: - The set of decision-based problems come into the division of P Problems who can be solved or produced an output within polynomial time. P problems being easy to solve

Definition of Polynomial time: - If we produce an output according to the given input within a specific amount of time such as within a minute, hours. This is known as Polynomial time.

Definition of Non-Polynomial time: - If we produce an output according to the given input but there are no time constraints is known as Non-Polynomial time. But yes output will produce but time is not fixed yet.

ne complexity of an algorithm, we use the input length as

An algorithm A is of *polynomial complexity* if there exists a polynomial $p()$ such that the computing time of A is $O(p(n))$ for every input of size n.

$\mathcal{P}$ is the set of all decision problems solvable by deterministic algorithms in polynomial time. $\mathcal{NP}$ is the set of all decision problems solvable by nondeterministic algorithms in polynomial time. □

Definition of NP class Problem: - The set of all decision-based problems came into the division of NP Problems who can't be solved or produced an output within polynomial time but verified in the **polynomial time**. NP class contains P class as a subset. NP problems being hard to solve.

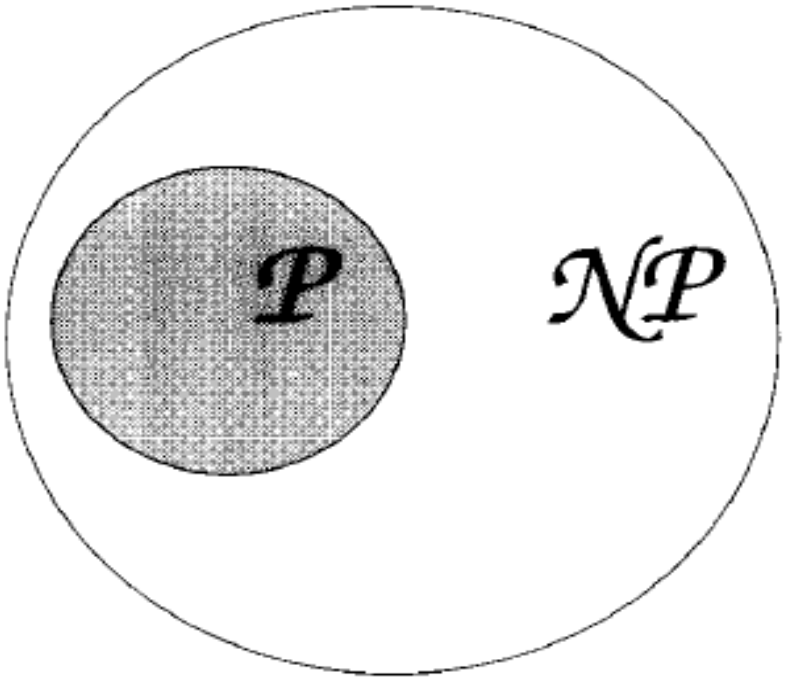
Note: - The term "NP" does not mean "not polynomial." Originally, the term meant "non-deterministic polynomial. It means according to the one input number of output will be produced.

Definition of NP-hard class: - Here you to satisfy the following points to come into the division of NP-hard

1. If we can solve this problem in polynomial time, then we can solve all NP problems in polynomial time
2. If you convert the issue into one form to another form within the polynomial time

Definition of NP-complete class: - A problem is in NP-complete, if

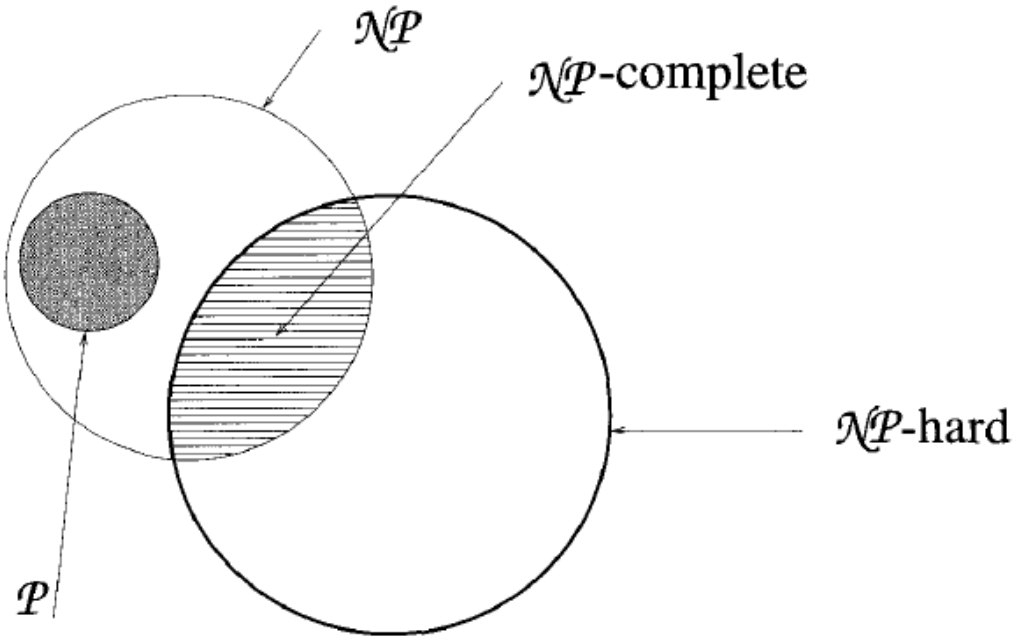
1. It is in NP
2. It is NP-hard


$\mathcal{P}$
$\mathcal{NP}$

Let L_1 and L_2 be problems. Problem L_1 *reduces* to L_2 (also written $L_1 \propto L_2$) if and only if there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time. □

This definition implies that if we have a polynomial time algorithm for L_2, then we can solve L_1 in polynomial time. One can readily verify that $\propto$ is a transitive relation (that is, if $L_1 \propto L_2$ and $L_2 \propto L_3$, then $L_1 \propto L_3$).

A problem L is $\mathcal{NP}$-hard if and only if satisfiability reduces to L (satisfiability $\propto L$). A problem L is $\mathcal{NP}$-complete if and only if L is $\mathcal{NP}$-hard and $L \in \mathcal{NP}$. □


$\mathcal{NP}$
$\mathcal{NP}$-complete
$\mathcal{NP}$-hard
$\mathcal{P}$

Pictorial representation of all NP classes which includes NP, NP-hard, and NP-complete

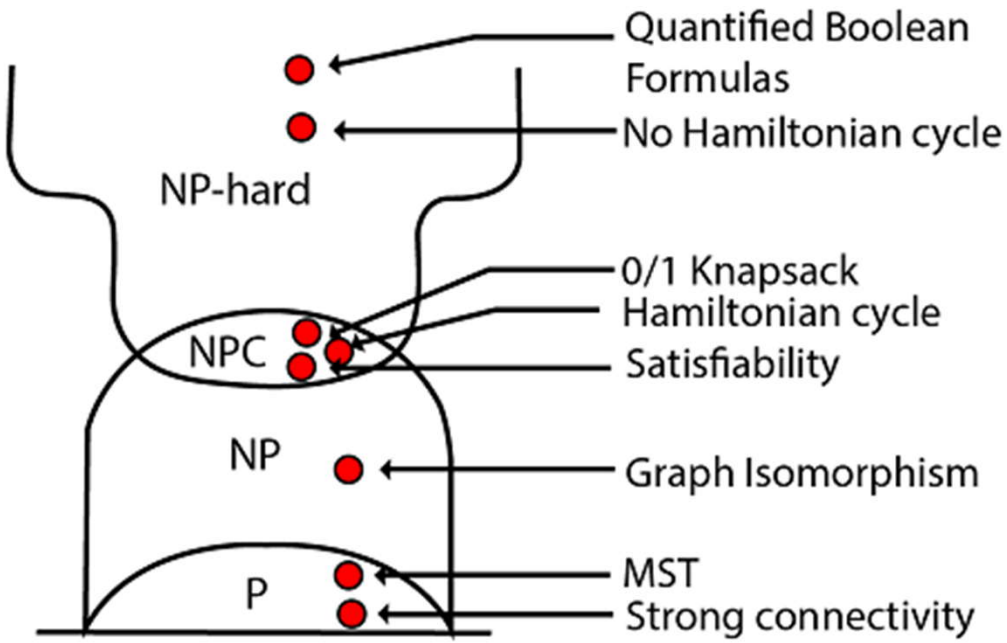


Fig: Complexity Classes