NP completeness

problem < polynomial
Exponential

A problem is said to be polynomial if there exists an algorithm that solves the problem in time $T(n) = O(n^c)$, where c is constant

Examples → sorting : $O(n \log n)$ $O(n^2)$
searching $O(n)$ $O(\log n)$
All pair shortest path $O(n^3)$
Spanning tree (MCST) $O(E \log E) = O(E^2)$

A problem is said to be exponential if no polynomial time algorithm can be developed for it or algorithm solves the problem in time $T(n) = O(2^n) \approx O(k^n)$ where k is constant

Examples → 0/1 Knapsack problem
TSP problem
Hamiltonian problem
Graph coloring.

Algorithms that solves the problem in polynomial time are efficient

P class → that can be solved in polynomial time & that can be verified in polynomial time

NP class → that can be solved in exponential time & that can be verified in polynomial time.

P class → P is a set of problems that can be solved in polynomial time using deterministic algorithm.

(or) if a problem can be solved in polynomial time on a regular computer, that problem is known as class P problem.

class P problems can be verified in polynomial time (solutions)

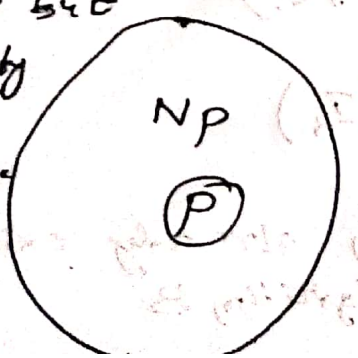
P problems

1. Solved & verify in polynomial time
2. Easy to solve, easy to verify
3. IT takes polynomial time to solve & verify
4. $P \subseteq NP$

NP class → NP is a set of problems that can be solved in polynomial time using non-deterministic (non deterministic polynomial) algorithm.

(or) NP problems are problems which can be solved in polynomial time on non-deterministic turing machine

- it can be verified in polynomial time
- Hard to solve but easy to verify
- it takes exponential time to solve



(We don't know how to works) Some steps are imaginary

```
Algorithm Search(A, n, key)
    i = choice()
    if (key == A[i])
    {
        print(i)
        success();
    }
    print(0)
    failure();
```

P is subset of NP

→ Any problem that can be solved by deterministic machine in polynomial time can also be solved by non deterministic machine in polynomial time.

NP problems can be solved by a polynomial time using "non deterministic" algorithm

↓

A magical algorithm that always makes a right guess among the given set of choices.

NP Hard and NP complete

To understand NP Hard & NP complete, we should first understand what is meant by reduction

Reduction

Let we assume that we have decision problem P_1 and P_2

↓

either answer is Yes or No

P_1 problem ← Input (I_1)

We need to design the algorithm

Algorithm (?)

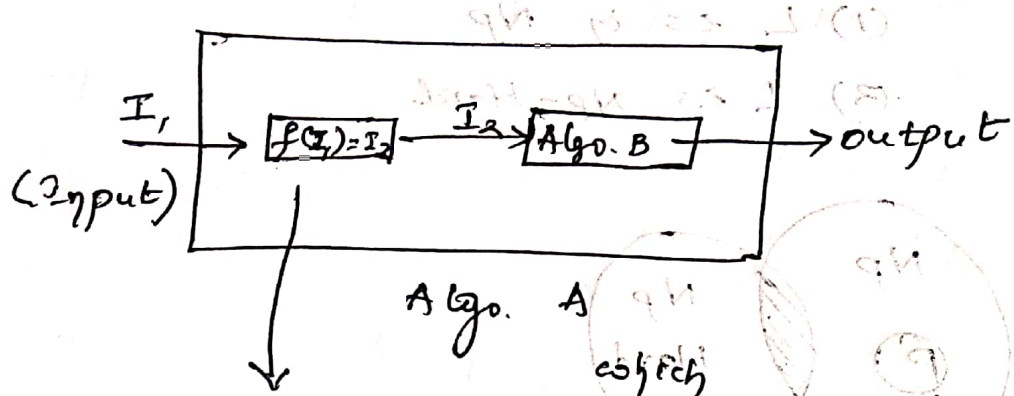
A

P_2 problem ← Input (I_2)

Algorithm already exists

✓ Algorithm B

Suppose Algorithm B can be used to solve problem P_1



which

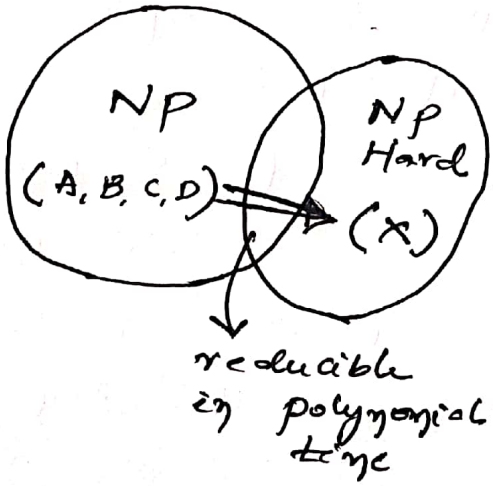
f is function transforms the input I_1 to I_2 in polynomial time

Problem P_1 is reducible to problem P_2 if there exists a function which converts input of A into input of B & solution of that instance provide solution to problem

$$P_1 \propto P_2$$

↳ we say P_1 is reducable to P_2

Np Hard → A problem is Np Hard if every problem in Np can be polynomial reduced to it



i.e $A \propto X$
$B \propto X$
$C \propto X$
$D \propto X$

$\therefore$ X is Np Hard

Np complete – A decision problem L is Np complete

if (1) L is in Np

(2) L is Np-Hard