NP-Hard and NP-Complete

Polynomial Time	Exponential Time
Linear Search = $O(n)$	0/1 Knapsack = $O(2^n)$
Binary Search = $O(\log n)$	TSP = $O(2^n)$
Insertion Sort = $O(n^2)$	Sum-of-Subsets = $O(2^n)$
Merge Sort = $O(n\log n)$	Graph Coloring = $O(2^n)$
Matrix Multiplication = $O(n^3)$	Hamiltonian Cycle = $O(2^n)$

⇒ So, for Searching, we want faster algorithm than Binary Search i.e Time to search should be $< O(\log n)$ or we can say $O(1)$ time.

⇒ For Sorting, we want algorithm taking less than $O(n\log n)$ time in worst case i.e Time to search should be $< O(n\log n)$ or we can say $O(\log n)$ or $O(n)$ time.

These are all polynomial time algorithms.

⇒ Similarly, for 0/1 Knapsack or TSP or SOS or Graph Coloring or Hamiltonian cycle, we want polynomial time algorithms to solve these problems.

⇒ These are research areas of Computer Science and Mathematics.

⇒ Guidelines or framework made for doing research on exponential time taking algorithms is called NP-Hard and NP-Complete

Approach used for solving Exponential Time Algorithms

(1) Show relationship between all exponential time taking algorithms.

This is because if one problem is solved, we can solve similar type problems easily.

(2) Build polynomial time non-deterministic algorithms for these exponential time taking problems.

Example of Non-Deterministic Algorithm

```
Algorithm NSearch (A, n, Key)
{
    j = choice(); // non-deterministic  => should take O(1)
    if (Key = A[j])
    {
        write (j);
        success (); // non-deterministic  => O(1)
    }
    write (0);
    Failure (); // non-deterministic  => O(1)
}                                         ______
                                           O(1)
```

A	12	7	5	8	2

Key = 8

Note: Deterministic polynomial time algorithm means that all the steps of the algorithm is known to us and is unambiguous. Also the algorithm is taking polynomial time for running.

e.g. Linear Search, Binary Search, Insertion Sort, Merge Sort etc.

Non-deterministic polynomial Algorithm means that all the steps of the algorithm are <u>not</u> known to us but we can write few steps that are deterministic.

Also, we want polynomial time for such algorithm.

∴ So research is going on "how to write non-deterministic polynomial time algorithm for exponential time taking problems.

P = Deterministic Polynomial Time Algorithms

NP = Non-Deterministic Polynomial Time Algorithms
↳ mostly written for Exponential problems.

Establishing Relationship between Exponential problems

Base Problem used is "Satisfiability" (SAT)

CNF-Satisfiability is a propositional calculus logic for boolean variables. $x_i = \{x_1, x_2, x_3\}$

$$CNF = \underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{\text{clause}} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}_{\text{clause}}$$

$\wedge \rightarrow$ Conjunction
$\vee \rightarrow$ Disjunction

Here, Satisfiability problem is "For what values of x_i, the given CNF formula is True".

Possible values can be

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

So, to check for True output, we have to try these 8 possible values.

$8 = 2^3 \leftarrow$ No. of boolean variable

$\therefore$ For n-boolean variables, we have to try 2^n values i.e exponential time

$\therefore$ CNF-Satisfiability is also an exponential time taking problem.

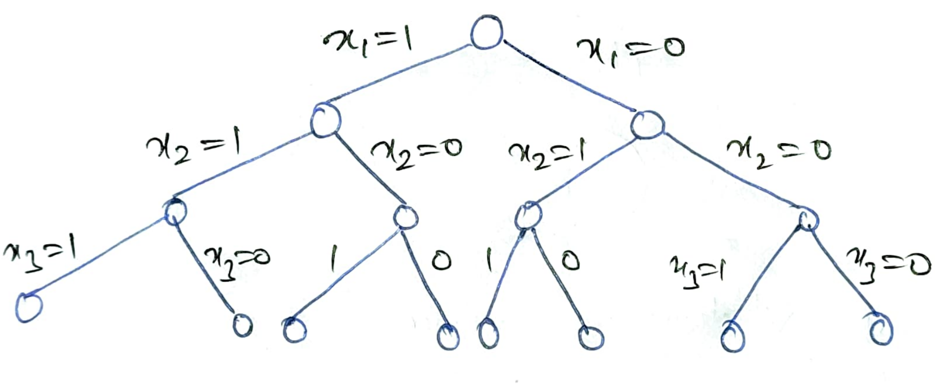


fig. State-space tree for 3-SAT

So, our objective is to show all exponential time problems (0/1 Knapsack, TSP, SOS, Graph Coloring, Hamiltonian cycle) similar to 3-SAT.

If 3-SAT is solved, means all these problems are solved in polynomial time.

How to show 0/1 Knapsack problem similar to 3-SAT ?

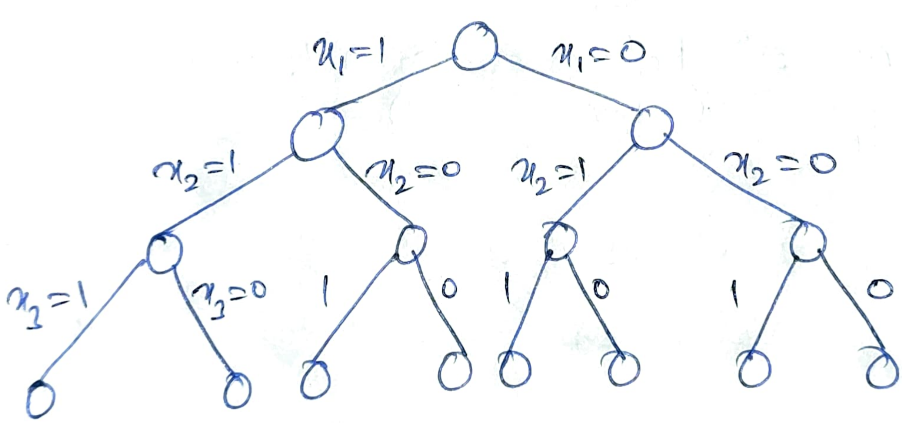
Profits = $\{10, 8, 12\}$ $n = 3$

weights = $\{5, 4, 3\}$ $m = 8$

Solutions can be written as $x_i = \{0/1, 0/1, 0/1\}$

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

8 values.



So, this is similar to 3-SAT.

NP-Hard and NP-Complete

Exponential time

$$\left.\begin{array}{ll} \text{0/1 Knapsack} & = O(2^n) \\ \text{TSP} & = O(2^n) \\ \text{Sum-of-Subset} & = O(2^n) \\ \text{Graph Coloring} & = O(2^n) \\ \text{Hamiltonian Cycle} & = O(2^n) \end{array}\right\} \text{Hard problems}$$

Now, we know, 3-SAT is NP-Hard problem.

Reduction :⇒ The process of showing relationship between any of the exponential time problem with 3-SAT is called Reduction.

(or)

SAT ⇒ Reduces to ⇒ 0/1 Knapsack
(in polynomial-time)

I_1 (Instance) I_2 (Instance)

(Example of SAT is used to 0/1 Knapsack)

Formula used in SAT is converted in 0/1 Knapsack

here, if we can solve SAT in polynomial time, then same algorithm can be used to solve 0/1 Knapsack problem. or vice-versa.

SAT ⇒ Reduces to ⇒ 0/1 Knapsack

↑ ∴ NP-Hard ↑ ∴ NP-Hard

$\text{Sat} \Rightarrow \text{Reduces to} \Rightarrow \text{Any problem } L$

$\uparrow$ NP Hard (Sat) $\qquad$ $\uparrow$ NP Hard (problem)

$\therefore$ $\text{Sat} \Rightarrow \text{Reduces to} \Rightarrow L_1$

$\uparrow$ NP Hard (Sat) $\qquad$ also $\qquad$ $\uparrow$ NP-Hard (L_1)

$L_1 \Rightarrow \text{Reduces to} \Rightarrow L_2$

$\uparrow$ NP-Hard (L_1) $\qquad$ $\uparrow$ NP-Hard (L_2)

NOTE:-> Since SAT is NP-Hard and we have non-deterministic polynomial algorithm for SAT.

$\therefore$ SAT is called NP-Complete.

$\text{SAT} \Rightarrow \text{Reduces to} \Rightarrow \text{Any Exponential Problem}$

$\uparrow$ (SAT) $\left.\begin{matrix}\text{NP Hard} \\ + \\ \text{NP}\end{matrix}\right\}$ NP Complete

$\uparrow$ (Exponential Problem) NP Hard

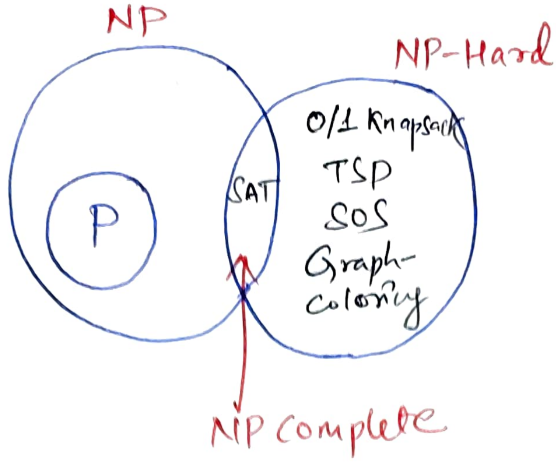
if Non-deterministic polynomial algorithm can be written, then this problem becomes NP-Complete.

So, for any exponential time taking algorithm, we must do two things

(1) Showing the relationship with SAT. (NP Hard)

(2) There exist Non-deterministic polynomial time algorithm for the exponential time problem.

If both points are achieved, then the problem becomes NP Complete.

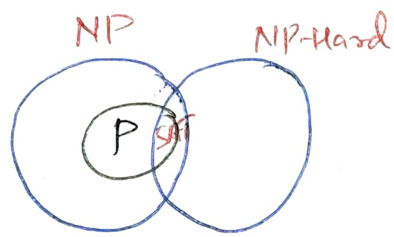


∴ we say, $P \subseteq NP$

If we can prove that $P = NP$, then we have discovered the solutions for all the problems.

Cook's Theorem

If SAT is in P iff $P = NP$.



P class Problem

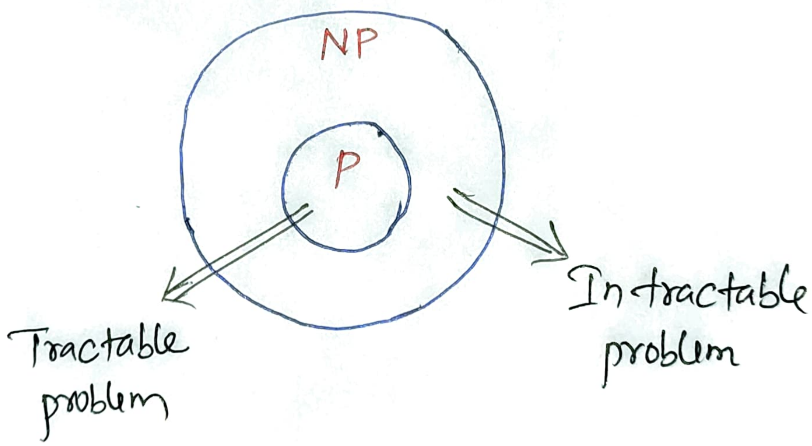
A problem which can be solved in polynomial time is known as P-class problem.

e.g. Searching algorithm
Sorting algorithm

NP Class (Non-Deterministic Polynomial time)

A problem which cannot be solved in polynomial time but can be verified in polynomial time.

e.g. 0/1 Knapsack problem
Su-do-ku
TSP
Prime-Factor



$$\text{A} \xrightarrow[\text{Reduce}]{\text{polynomial time}} \text{B}$$

Let A and B are two problems.

A reduces to B iff there is a way to solve A by deterministic algorithm that solve B in polynomial time

If A is reducible to B, then we denote $A \propto B$

NP-Hard :-> A problem is NP-Hard, if every problem in NP can be polynomially reduced to it.

NP-Complete :-> A problem is NP complete if it is in NP and it is in NP Hard.