NP-Hard and NP-complete

Polynomial Time	Exponential Time
Linear Search - n	0/1 knapsack - 2^n
Binary Search - $\log n$	Traveling SP - 2^n
Insertion Sort - n^2	Sum of subsets - 2^n
Merge Sort - $n \log n$	Graph Coloring - 2^n
matrix multiplication - n^3	Hamiltonian Cycle - 2^n

Here we have categorized the algorithms into two, one is polynomial time taking algorithms and another is exponential time taking algorithms.

Scope of doing research ?

- See for searching, we have linear search algo which takes $O(n)$ time & faster than that is Binary search algo which takes $O(\log n)$ time. We are in search of algo which is much faster than these i.e. $O(1)$ time.

- Similarly the fastest sorting algo. takes $O(n \log n)$ time, so we want much faster than this i.e. $O(n)$ time.

- Similarly, for these exponential time algo's, we want polynomial time algos.

- So we want the easy method which can solve the exponential time algo's in polynomial time.

So polynomial time could be n^c, c is the constant.

if $c = 2$ & $n = 1000$

1000^2 Vs 2^{1000}

Both are different

- framework that is made to do the research on exponential time algo's, is known as NP-Hard & NP-complete.

- when you are unable to solve exponential time problems, that means not able to find polynomial time solutions. Then at least try to show the similarity between them (set of problems), so that if one problem is solved, all other problems should also be solved.

- Here we will be not doing the research work individually on each and every problems. So try to find the relationship between them.

Deterministic ⇒ Each and every statement how it works, we know it clearly. We know the working of algo.

Nondeterministic ⇒ We can write nondeterministic algo also, that means we don't know how they are working.

- If I am trying to write polynomial time algo for 0/1 knapsack problem, then I may be knowing most of the statement but some of the statement that I may not be figuring out i.e. how to make them polynomial? So leave them blanks & say that this is nondeterministic.

- In nondeterministic algo. most of the statement may be deterministic, but some of the statement are non-deterministic. in future

- So Research point of view — someone may convert the non-dete--- part into deterministic.

- How to write non-deterministic algo?

Non deterministic algo:

Ex: Nondeterministic Search algo: Searching a key from an array A of size n.

```
Algorithm NSearch (A, n, key)
{
    j = choice ( );
    if (key = A[j])        // if key element is present in array
    {
        write (j);         // search is successful & write the output
        success ( );
    }

    write (0);
    Failure ( );
}
```

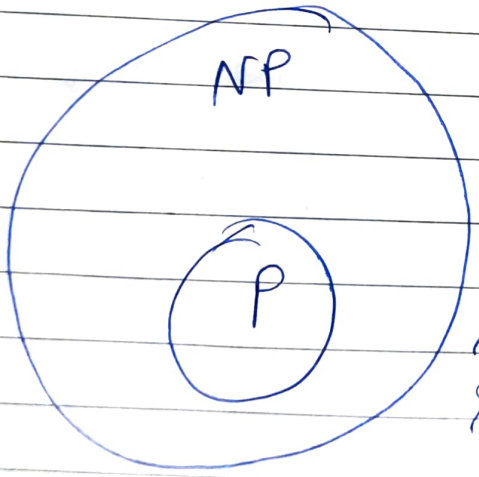
index

- Choice, Success & Failure are non-deterministic statement.

P → is the set of deterministic algo's which are taking polynomial time.

Ex: linear search
Binary "
Bubble sort
Merge "
Single source sortest path
Huffman coding etc---

NP → these ~~algos~~ are nondeterministic polynomial time taking algo's.



P is the subset of NP

So deterministic is the subset of non-deterministic

- Some problems may be in class of NP today but may be tomorrow in P class.

As we discussed that if we are unable to solve exponential time problems then at least we should show the relationship between them such that if one problem is solved then it should be easy to solve other problems also in polynomial time.

How to relate them together?

- For relating them together, we need some problem as a base problem.

Exponential time
- 0/1 Knapsack
- Traveling SP
- Graph coloring
etc.

- So which problem we take as base problem?
- The base problem is the "Satisfiability problem"

CNF - Satisfiability

using boolean variables $\{x_1, x_2, x_3\}$

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

formula; $\vee$ = OR; $\wedge$ = AND; C_1; C_2

What is satisfiability problem?

The satisfiability problem is to find out for what values of x_i the above formula is true?

- So what are the possible values of x_i.

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 we have eight possibility.

- So solving the above formula we can use these x_i values. So how many values we should try i.e. eight.

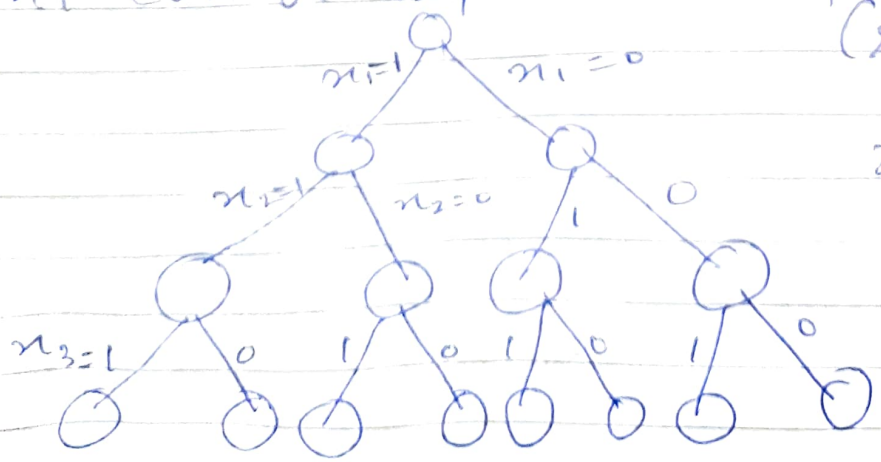
so eight $\Rightarrow 2^3 \Rightarrow 2^n$

$\downarrow$

So for n variables.

So it takes exponential time(2^n), so it belongs to same set of exponential time problems.

x_i can be represented in form of tree. (state space tree)



root to leaf node.

- If satisfiability can be solved in polynomial time then all can be solved in polynomial time.

- Let us consider 0/1 knapsack problem using the same tree.

0/1 Knapsack

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

profit

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

↓ Capacity of bag

- we have to fillup the bag such that the profit is maximized and should not exceed the capcity of the bag.

so Soluting $x_i = \{ \underline{0/1}, \underline{0/1}, \underline{0/1} \}$

x_1	x_2	x_3
0	0	0
0	0	1
⋮		

$2^{n=3}$ possibilities

So Solution that is requisite for knapsack problem is similar to the satisfiability problem.

How we solved the satisfiability in the

Same way we can solve knapsack problem.

In:

Satisfiability → we have to see formula is true or not

knapsack → whether the profit is maximized or not.

- So knapsack problem can be solved using state space tree.
- So this tree is showing the solutions of satisfiability & knapsack both. So we are trying to relate them.

"If we are able to solve this state space tree in polynomical time then you can get the solution for satisfiability & 0/1 knapsack in polynomial.

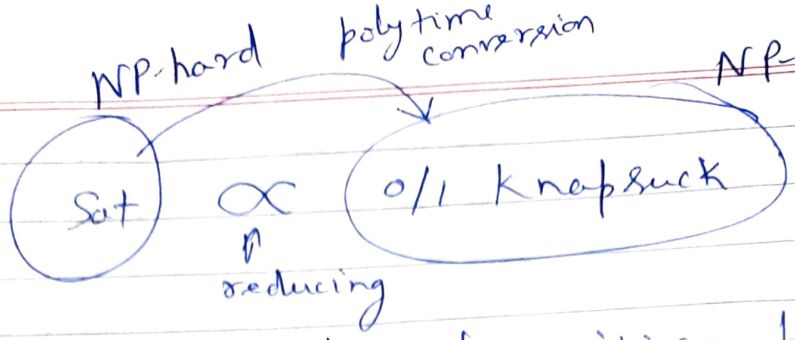
NP-hard

Let us consider Satisfiability (base problem) as NP hard and other simply hard problems.

- Exponential time algo's are the hard problems.
- we can say that hard problems (as per list) are related to satisfiability only, so Sat. can be solved then all can be solved.

Reduction:

Satisfiability reduces to 0/1 knapsack problem.



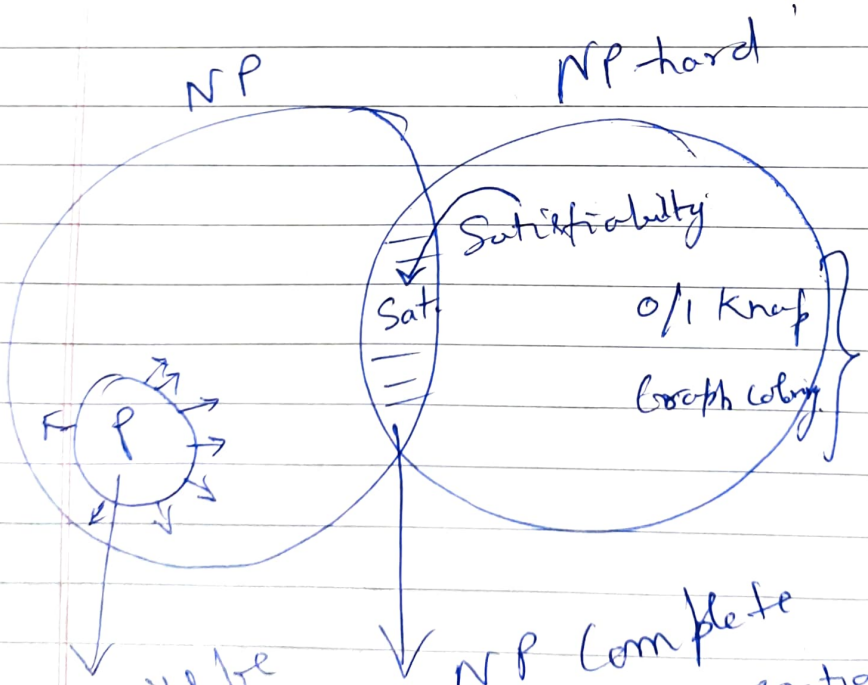
- Reduction has transitive property

Satisfiability $\propto L_1 \propto L_2$

Satisfiability: NP-hard; L_1: NP-Hard; L_2: NP-hard

- We know that there exist the non-deterministic algo for Satisfiability.

NP-hard
NP-complete → Satisfiability problem.



P will be keep on expanding & non-deterministic will be becoming deterministic.

NP Complete
(intersection of NP & NP-hard)

P ⊆ NP

- We need to proof that P = NP

- Cook's has tried to proof that.
- If we are able to proof this then Satisfiability will come into P, if P = NP.