Chapter - 6

Computational Complexity

Till now, we have been discussing about computability. Problems that can be solved and problems that are impossible to solve were our matter of discussions. Some issues we will discuss in this chapter:-

* Are all theoretically computable problems, practically computable?
* How much complex can a problem be, even if they are solvable?
* How to measure and compare the complexities of the problems??

Some terminologies:-

* Time Complexity:- Duration of time required by an algorithm to complete.
* Space Complexity:- Number of memory cells that an algorithm requires.

→ Algorithms that has minimum time and space complexity are better algorithms

* Big-O Notation:- (Big-Omicron or Big-Oh)
  - A notation used to describe time or space complexities of the algorithm.

Space complexities of the algorithm

- It specially describes the worst-case scenario.
- We will mainly focus on time complexity of algorithms in this chapter.

* Time complexity of any sequential algorithms (without loops) is said to be "Order 1" written in Big-O notation as $O(1)$.

Algorithms with single loop whose complexity increases linearly with numbers of inputs (n) has complexity of order n written as $O(n)$.

Algorithms can have polynomial complexity.

$O(n^2)$ ⇒ one nested loop

$O(n^3)$ ⇒ two level nested loop

Complexity of algorithm can also be exponential.

$O(2^n)$, $O(10^n)$ ⇒ which is BAD for computation

Algorithms with logarithmic time complexity are very very fast.

$O(\log n)$ [very much preferred]

The lower order terms can be ignored considering only the highest order. Also, constant coefficient can be ignored.

eg: $O(3n^2+5n+1) = O(3n^2) = O(n^2)$

Examples:-

In sorting algorithm, complexity of

Insertion sort is $O(n^2)$

Bubble sort is $O(n^2)$

Quick sort is $O(n \log n)$

Note:-

Big-omicron, $O()$

⇒ upper bound asymptotically

Big-omega, $\Omega()$

⇒ lower bound asymptotically

Big-Theta, $\Theta()$

⇒ Tight bound

⇒ both upper & lower bound asymptotically

Class P problems

→ Also called "Polynomial time problems" are the class of problems solvable by a Deterministic Turing Machine in polynomial time.

i.e, Time complexity = $O(P(n))$ ∵ $P(n)$ → polynomial function

OR

A language L is said to be in class P if there exist a polynomially bounded Turing machine (deterministic) that accepts L.

Polynomially bounded turing machines are guaranteed to halt after almost $P(n)$ steps where n is length of input and $P(n)$ is some polynomial function of n.

→ Languages in class P are also called Polynomially decidable.

class NP [Non- Deterministic Polynomial Time]

→ class of problems solvable by a non-deterministic turing machine in polynomial time.

Language L is said to be in class NP, if there exist a polynomially bounded non-deterministic turing machine that accepts L.

→ Languages in class NP are polynomially verifiable.

Clearly, we ~~sto~~ see that all class P problems are also class NP i.e. $P \subseteq NP$, But the question is $NP \subseteq P$? is still unknown.

Tractable Problems:-

Problems that have algorithm to solve such that it takes no more time than some polynomial function of input size (n) are called tractable. All class P problems are tractable.

→ Problems that are solvable in theory, but can't be solved practically are called intractable problems.

→ Exponential time complete (Exp-Time complete) problems are known to be intractable problems

If it is proved that P ≠ NP then NP problems are also intractable.

class P, NP and Exp-Time: Example Problems:-

All problems that our computers can solve with polynomial time complexity are problems in complexity class P. Example:- Sorting given numbers. Find prime numbers etc.

Examples of class NP (which are not known to be class P till now).

* Travelling Salesman Problem
  [finding shortest path for salesman to travel, visiting each node only once]
* The Hamilton Circuit Problem
  [finding a path visiting each node only once and back to start node)
* Equivalence of Regular Expression
  (NFA → DFA ⇒ NP problem)
* Intersection of finite automata
  [M1, M2, ... Mn]
* Linear Programming Problems (optimizing problems)

All these problems have polynomial time solution on a Non-Deterministic T.M, but Exp. time on Deterministic Turing Machine.

-> A famous exponential time complexity problem of "Tower of Hanoi" is not even NP.

NP-Complete Problems :-

A decision problem C is NP-complete if

1) C is in NP and
2) Every problem in NP is reducable to C in polynomial time.

i.e, NP complete problems are atleast as hard as any other problems in NP and if any one of NP complete problem can be solved in Deterministic Polynomial time. Then, P = NP.

Examples of NP-complete Problems:-

* Travelling Salesman Problem
* Bounded Tiling problem
* Hamilton Path Problem, Hamilton Circuit Problem
* Linear Programming Problem

etc.

Reduction:

→ Expressing one problem in terms of other.

We say, problem A is atleast as hard as B and write $B \leq A$ if we can express B in terms of A (some subroutine call to A in algorithm for B, so if A can be solved in polynomial time and B can be reduced to A in polynomial time, then B can also be solved in polynomial time.

Dealing with NP:-

Many practical problems are NP-complete and we cannot ignore them. We can't quickly solve them either. So, some approaches to solve them could be

(i) Heuristic approach:-
→ See if you can solve a reasonable fraction of the common cases.

(ii) Approximate solution:-
Some NP-complete problems can be approximately solved in shorter time.

(iii) Use exponential solution anyway:- if you need exact solution. But may take too long to complete.

(iv) Choose better abstraction:-
The details considered or ignored of real world might be making the difference.

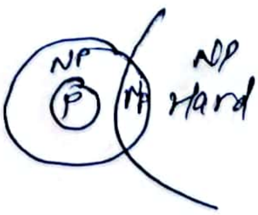
Properties of NP-complete problems:-

1) No polynomial time algorithm has been found for any one of them.
2) It is not proved that polynomial time algorithms for these problems do not exists.
3) If one of them can be solved in polynomial time, all of them can be too. [i.e, P=NP will be established]
4) If one of them cannot be solved in polynomial time, then none of them can be. [P ≠ NP]

NP-Hard Problems:-

→ similar to NP, except that they are not in complexity class NP.

NP-complete ≤ NP-Hard



Eg:-
- Tower of Hanoi → Exp. time complex
- Halting Problem → Undecidable.

→ even harder than NP-complete, but if solved in polynomial time ⇒ P = NP

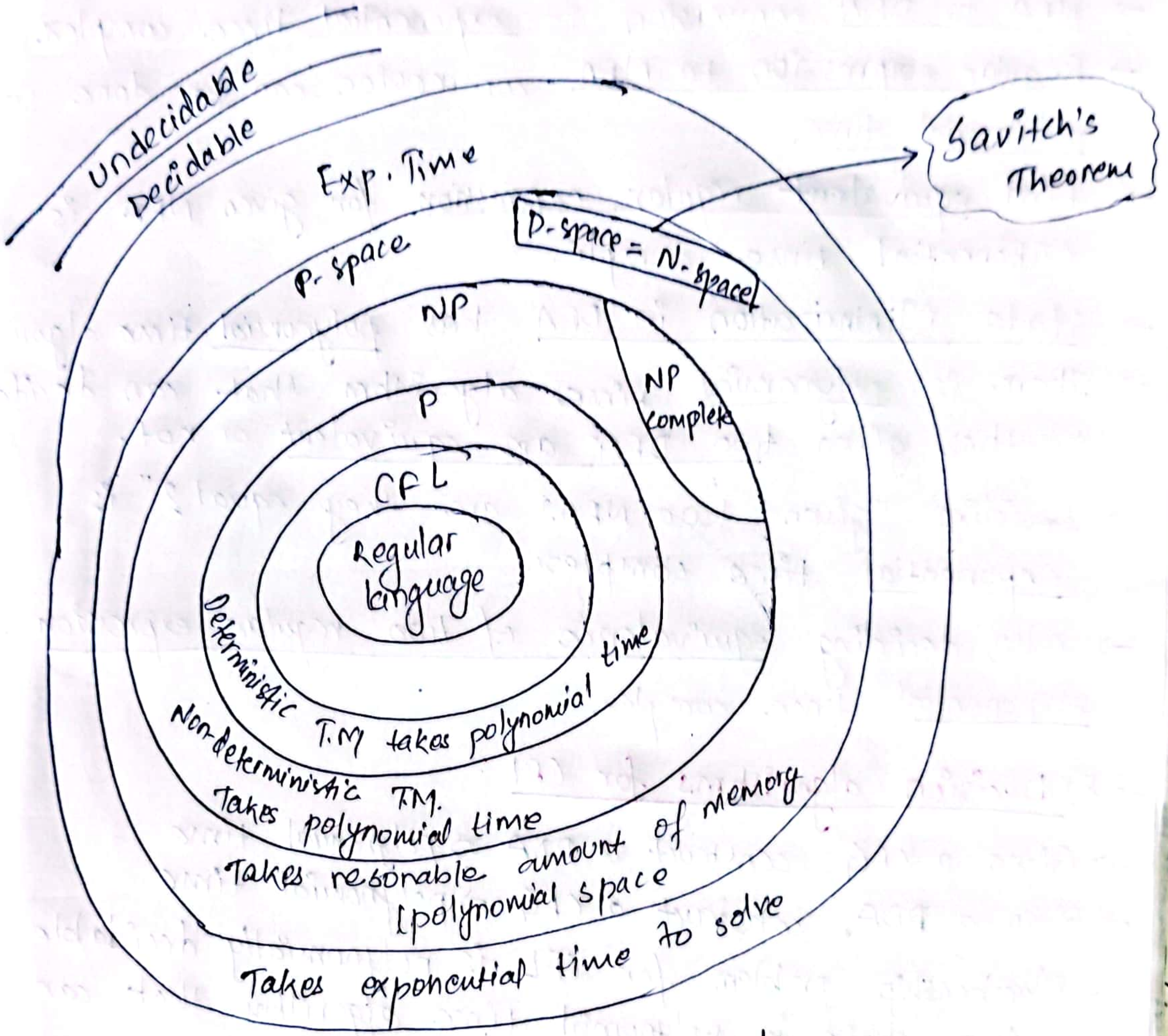


fig: classification of problems according to the complexity.

chapter-2

2.6) Decision Algorithms for regular language:-

→ Membership problem for DFA (Does given DFA accepts given input string?) has a polynomial time algorithm to solve it.

→ NFA to DFA conversion is exponential time complex.

→ Regular expression to NFA conversion can be done in polynomial time.

→ Find equivalent regular expression for given NFA is exponential time comple.

→ State Minimization in DFA has polynomial time algorithm.

→ There is polynomial time algorithm that can decide whether given two DFA's are equivalent or not.

→ Deciding, "given two NFAs are they equal?" is exponential time complex.

→ Also, deciding equivalence of two regular expression is exponential time complex.

3.8) Decision algorithms for CFL:-

→ Given a CFG, construct a PDA ⇒ polynomial time

→ Given a PDA, construct a CFG ⇒ polynomial time

→ Membership problem for CFL is polynomially decidable. (i.e. there is polynomial time algorithm that can decide if input $x \in L(G)$ or not).