Brief Introduction to complexity Classes P, NP, NPC, NP-Hard

Dr. Priyadarshan Dhabe,
Ph.D (IIT Bombay)

Types of problems

- **Tractable problems:-** problems which are solvable in reasonable (polynomial) time
  - Efficient algorithm exists to solve them

    (e.g. sorting, searching)
- **Intractable problems:-** Some problems are intractable, as they grow large, we are unable to solve them in reasonable time.
  - Efficient algorithm do not exists to solve them
  - Algorithm exists but are not efficient

Intractable- difficult to manage/solve

Notion of polynomial time

- A problem is solvable in **polynomial time** if the number of arithmetic operations required to solve it can be defined by (bounded from above by) a polynomial in **n**, for **input size n**.

for a matrix multiplication of 2 (nxn) matrices

- we need to compute n^2 elements

- for computation of 1 element of result we need

n - multiplications

(n - 1) - additions

$$O(n^3)$$

$$\text{total} = n^2(n + (n-1))$$

$$= \boxed{2n^3 - n^2} \; (\Leftarrow \text{polynomial in n})$$

Notion of polynomial time

- Polynomial time solvable problems are problems whose worst case time complexity is $\mathbf{O}(\boldsymbol{n^k})$, where n-is input size and k-is constant and k<<<n (if k>n it behaves like exponential)
- Following are polynomial time problems
- $O(1)$,$O(n), \mathrm{O}(n^2), \mathrm{O}(n^3)$
- But, we also recognize following problems as polynomial time $O(logn)$, $\mathrm{O}(\mathrm{n}\, logn)$, since they are bounded from above by polynomials like n, n^2, respectively.

Non polynomial time problems

- Problems whose worst case time complexity is $\mathbf{O}(\boldsymbol{k}^{\boldsymbol{n}})$, where n-is input size and k is constant, are called **non-polynomial time** problems.
- k^n -**is genuinely exponential**
- But we also recognize following functions as **non polynomial (exponential)**

  $O\left(n^{logn}\right) - \text{not quite exponential}$

  $O(n!)$,- not exactly exponential, grows faster than $\mathbf{O}(\mathbf{2}^{\boldsymbol{n}})$

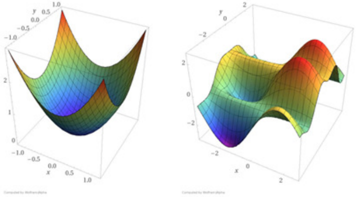
  $O(n^n)$,- super exponential
- So we wrapped problems in two broad classes.

Some common complexity functions

constant	$O(1)$	
logarithmic	$O(\log n)$	
linear	$O(n)$	
n-log-n	$O(n \times \log n)$	Polynomial time
quadratic	$O(n^2)$	
cubic	$O(n^3)$	
exponential	$O(k^n)$, e.g. $O(2^n)$	
factorial	$O(n!)$	Non-Polynomial time
super-exponential	e.g. $O(n^n)$	

Types of problems-Optimization problems

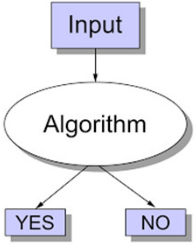
- **Optimization Problems** – An optimization problem is one which asks, "What is the optimal solution to problem X?"
- **Examples:**
  - 0-1 Knapsack
  - Fractional Knapsack
  - Minimum Spanning Tree (MST)



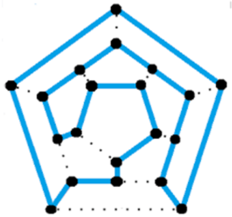
- Optimization algorithms are used for solving them.

Types of problems-Decision problems

- **Decision Problems:-**
- A decision problem is one with **yes/no answer**
- **Examples:**
  - Does a graph G have a MST of weight <= **W**?



- Many problems will have both **decision** and **optimization** versions
- Eg: Traveling salesman problem
  - **optimization:** find **Hamiltonian cycle** (cycle containing each vertex in V) of minimum weight
  - **decision:** is there a Hamiltonian cycle of weight <= k ???



P- Class problems

- **P class:-** the class of problems that are solvable in polynomial-time using **deterministic** algorithms.
- That is, they are solvable in **O(n^k),** where k is constant (highest power of n) in the polynomial on n (input size).
- In computer science, a **deterministic algorithm** is an algorithm which, given a particular input, will always produce the same output, with the underlying machine always passing through the **same sequence of states**. (Ref:- wikipedia)
- Everything is **determined/decided/defined** without confusion.
- **E.g.** Sorting, searching, matrix multiplication and etc.

Time complexities of P-class problems

n - linear search

n^2 – Quadratic (bubble sort)

n^3 – matrix multiplication

$\log n$ - binary search

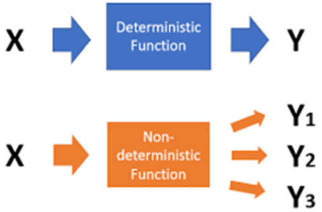
$n \log n$ - merge sort

note that $(\log n)$ is $O(n)$

and $(n \log n)$ is $O(n^2)$

NP- Class problems

- Note that **NP stands for "Nondeterministic Polynomial-time"**
- **NP Class Problems: -** the class of problems that are **solvable** in **polynomial time using** a **nondeterministic algorithm, whose solution can be verified in polynomial time**"
- NP- class problems are **decision** problems.
- A **nondeterministic algorithm/computer** is one that can "**guess**" the answer or solution .
- **nondeterministic algorithm- not known/determined today. We don't know that algorithm**



P and NP classes

Solvable and verified in polynomial time

Solvable in exponential time but verified in polynomial time e.g 0/1 knapsack

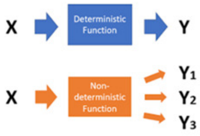
NP class examples :- Fractional Knapsack, MST , Traveling Salesman , Satisfiability (SAT) decision problem- the problem of deciding whether a given Boolean formula is satisfiable

What is Non-deterministic algorithm????

-In computer programming, a **nondeterministic algorithm** is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm.

-There are several ways an algorithm may behave differently from run to run. A concurrent algorithm can perform differently on different runs due to a race condition.

-The notion was introduced by Robert W. Floyd in 1967.



Non-deterministic algorithm

A is array of size n

```
Search(A, n, key)
{
     j=guess( );
     if(key==A[j])
       then write(j);
       return success();
     else
      return failure();
}
```

$O(1)$

Algorithm is, thus of O(1) and hence of polynomial time.

-I can not determine/decide /define algorithm **guess( )**

- Thus, it is non deterministic
- But solvable in polynomial time
- Today , we don't know algorithm **guess**, but may be able to define in future.

P and NP summary

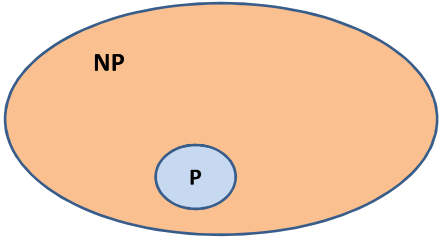
- P = set of problems that can be solved and verified in polynomial time using deterministic algorithm
  - Examples: sorting, searching, matrix multiplication
- NP = set of problems for which a solution can be verified in polynomial time but solvable in exponential time (includes O(n!) and $O(n^{logn})$ also) using deterministic algo. OR solvable in polynomial time using nondeterministic algo.
  - Examples: Fractional Knapsack, TSP, SAT (decision)

Suggested Reading -7 Difference Between P And NP Problems In Computer Science - Viva Differences

P and NP -Relationship

- Clearly $\mathrm{P} \subseteq NP$

Venn Diagram

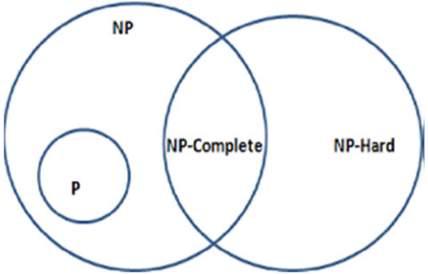


P-tractable
NP-intractable

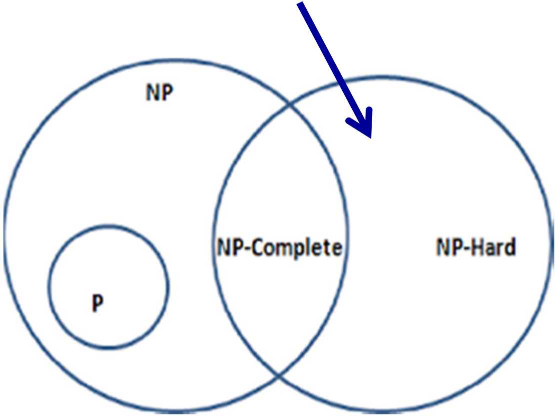
- As and when we know a deterministic polynomial time algorithm for a NP problem, it will be entered into P class region.
- Open question does $\boldsymbol{P = NP}$ or $\boldsymbol{P \neq NP}$?
- For some problem instance in NP if we can find polynomial time deterministic algorithm, then for that instance $\boldsymbol{P = NP}$
- But in general $\boldsymbol{P \neq NP}$

NP-Hard class Problems

- Non deterministic polynomial time hard problems
- NP-Hard problems are optimization problems.
- **NP–hard:-** A problem is NP-hard if an algorithm for solving it can be translated/Reduced into one for solving any NP-problem (nondeterministic polynomial time) problem.
- **NP-hard** therefore means "at least as hard as any NP-problem," although it might, **in fact, be harder**..



$O(n!)$ and $O(n^n)$

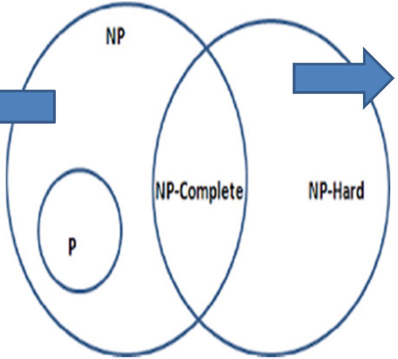


$$A\ problem\ R\ is\ polynomial\ time\ reduced\ to\ Q, dnoted\ as$$
$$R \leq_p Q$$

and if Q is in NP-class, then R is called NP-hard

Difference between NP and NP-Hard problems

-**Decision problems**
- e.g. is MST <w exits?(yes/No)
- Verifyable in polynomial time
- Fractional Knapsack, TSP, SAT (decision)



-**optimization problems**
- e.g. finding MST
- At least as hard as NP or harder than NP
- Turing machine halting, SAT, TSP

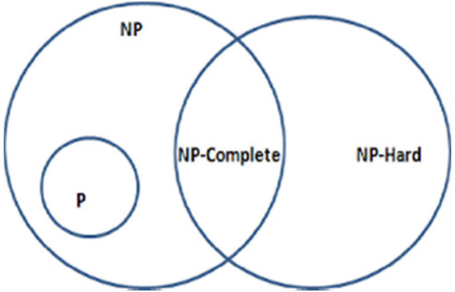
NP-Complete problems

$$We\ say\ that\ a\ problem\ Q\ belongs\ NP - Complete, if\ it\ is$$

1. NP-Hard and
2. Q ∈ NP

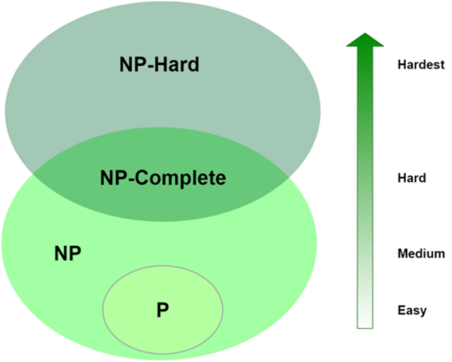
$$NPC = NP \cap NP - Hard$$

NPC problems are more hard problems in NP and thus are in NP-hard also



NP-Complete- is a decision as well as optimization problem
NP-complete problems can be verified in polynomial time but not NP-hard

Easy to Hard problem classification



https://www.baeldung.com/cs/p-np-np-complete-np-hard

Boolean satisfyability Problem

Two instances of circuit satisfiability problems

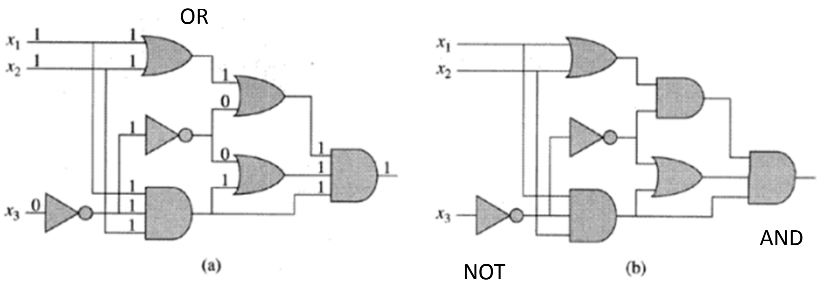


Figure 34.8 Two instances of the circuit-satisfiability problem. **(a)** The assignment $\langle x_1 = 1, x_2 = 1, x_3 = 0\rangle$ to the inputs of this circuit causes the output of the circuit to be 1. The circuit is therefore satisfiable. **(b)** No assignment to the inputs of this circuit can cause the output of the circuit to be 1. The circuit is therefore unsatisfiable.

Boolean satisfiability Problem

- **Circuit satisfiability problem** can be reduced to **formula satisfiability** problem

3CNF formula: literal **3-SAT**

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \underbrace{(x_3 \vee \overline{x_5} \vee x_6)}_{\text{clause}} \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

For N input variable it is of $O(c * 2^N))\ \ for\ c > 0$