Computational complexity.
NP- Completeness

An Intro:

→ ∃ many real life practical problems for which no efficient algorithm is known. [whose intrinsic difficulty no one has yet managed to prove]

E.g: TSP, Optimal graph coloring, the knapsack problem, Hamiltonian cycles, integer programming, Finding the longest simple path in a graph, Satisfying a boolean formula.

To prove: An efficient algorithm to solve any one of the problems listed above would automatically provide us with efficient algorithms for all of them!

→ They are all of similar complexity [but we don't know whether they are easy or hard to solve]

→ New problem → if u r not able to come up with an efficient solution

{If u can show that it is computationally equivalent to one of these problems → convincing evidence that at the moment no one else knows how to efficiently solve the problem.

* E.g. of problems that are hard to solve, but the validity of a solution can be verified efficiently.

* Factorization:
- Given a composite number
- hard to find a non-trivial divisor
- but any purported divisor can be verified easily.

* Hamiltonian cycle :

→ Given an undirected graph $G:\langle V,E\rangle$, to find a path that starts with some node, visits each node exactly once and returns to the starting node.

→ The graph is 'Hamiltonian' if such a cycle exists.

P and NP

→ An algorithm is efficient if ∃ a polynomial $p(n)$ such that the algorithm can solve any instance of size 'n' in a time $O(p(n))$. Such algorithms are called 'polynomial-time' algorithms.

Decision Problems:

- the answer to these problems is either yes/no
- Can be thought of as defining a set X of instances on which the correct answer is "yes" - yes-instances, any other instance is a no-instance.
- Correct algorithm that solves a decision problem accepts "yes-instances" and rejects "no-instances".

e.g.: Find a Hamiltonian cycle in G
-not a decision problem.
Is graph 'G' Hamiltonian?
- a decision problem.

P : P is the class of decision problems that can be solved in polynomial time (or) that can be solved by a polynomial-time algorithm.

{ Probabilistic algorithms are not covered by this definition }

X → a decision problem

Q → a set - the proof space for X

Proof system for X → a set F of pairs $\langle x, q \rangle$.

$(\forall x \in X), \; (\exists q \in Q) \mid (x, q) \in F$

→ no such q must exist when $x \notin X$.

$$(\forall x \notin X)\,(\forall q \in Q)\,[\,(x,q) \notin F\,]$$

For. e.g:

① If X → set of all Hamiltonian graphs,
Q → set of sequences of graph nodes and
define $(G, \sigma) \in F$ iff the sequence 'σ'
specifies a Hamiltonian cycle in G.

② If X → set of all composite numbers,
$Q = \mathbb{N}$ → proof space.
$F = \{(n, q) \mid 1 < q < n$ and q divides $n\}$
as the proof system

(or)

$F' = \{(n, q) \mid 1 < q < n \text{ and } \gcd(n, q) \neq 1\}$

Class NP → corresponds to the decision problems that have an efficient proof system (i.e) each yes-instance must have atleast one succinct certificate, whose validity can be verified quickly.

NP → stands for "non-deterministic polynomial time"

(i) Any polynomial-time solvable problem is also in NP

(ii) We donot know how to prove the existence of even a single problem in NP that can't be solved in polynomial time.

(iii) NP-definition is asymmetric.
(i.e.) succinct certificates for yes-instances but no such requirement for no-instances. (e.g): non-Hamiltonian graph? (not in NP)

NP- Definition:

- NP is the class of decision problems X that admit a proof system $F \subseteq X \times Q$ such that $\exists$ a polynomial $p(n)$ and a polynomial-time algorithm 'A' such that:
  - For all $x \in X$ $\exists$ a $q \in Q$ such that $(x,q) \in F$ and moreover $|q| = O(p(n))$ where 'n' is the size of x.
  - For all pairs $\langle x,q \rangle$, algorithm 'A' can verify whether or not $\langle x,q \rangle \in F$. In other words $F \in P$.

Theorem: $P \subseteq NP$.

Proof:
- No evidence / proof needed, as the decision problem can be handled by ourselves.
- let X - an arbitrary decision problem $X \in P$
- let $Q = \{0\}$ - a trivial proof space.

Define: $F = \{\langle x,0 \rangle \mid x \in X\}$

- $\therefore$ An yes-instance admits one succinct certificate '0' and no-instances have no certificates at all.
- It suffices to verify that $x \in X$ and $q = 0$ to establish that $(x,q) \in F$.
- can be done in polynomial time because we assumed that $X \in P$.

IS $P = NP$? (open question)

Conjecture $P \neq NP$

We will study the consequences of this!!!

Polynomial Reductions

Defn: Let A and B be two problems. We say that A is polynomially turing reducible to B if there exists an algorithm for solving A in a time that would be polynomial if we could solve arbitrary instances of problem B at unit cost. This is denoted $A \le_T^P B$. When $A \le_T^P B$ and $B \le_T^P A$ both hold, we say that A and B are polynomially turing equivalent and we write $A \equiv_T^P B$.

$\Rightarrow$ If $A \le_T^P B$ and $B \le_T^P C$ then $A \le_T^P C$.

HAM $\Rightarrow$ problem of finding a Hamiltonian cycle in a graph. (if one exists)

HAMD $\Rightarrow$ Deciding whether or not a graph is Hamiltonian.

Theorem: HAM $\equiv_T^P$ HAMD.

Proof: To prove: HAMD $\le_T^P$ HAM.

```
function HAMD (G: graph) {
    σ ← HAM(G)
    if σ defines a Hamiltonian cycle in G
            return true
    else    return false
}
```

↑ Clear that HAMD takes polynomial time provided we count the call on HAM at unit cost.

To prove: HAM $\le_T^P$ HAMD

- (i.e.) we find a Hamiltonian cycle assuming we know how to decide if such cycles exist.

```
function HAM(G=<N,A>) {
    if HAMD(G) == false
        return "No solution!"
    for each e ∈ A do {
        if (HAMD(<N, A\{e}>))
            A ← A\{e}
    }
    σ ← sequence of nodes obtained by
        following the unique cycle in G.
    return σ
}
```

- Clearly HAM takes polynomial time if we count each call to HAMD at unit cost.

$$\left.\begin{array}{l} - \text{HAM} \le_T^P \text{HAMD} \\ - \text{HAMD} \le_T^P \text{HAM} \end{array}\right\} \Rightarrow \text{HAM} \equiv_T^P \text{HAMD}.$$

Hence Proved.

<u>Theorem:</u> Consider two problems A and B. If $A \le_T^P B$ and if B can be solved in polynomial time, then A can also be solved in polynomial time.

<u>Proof:</u> A and B $\Rightarrow A \le_T^P B$

let $p(n) \to$ polynomial and the reduction algorithm for problem 'A' never requires the solution of more than $p(n)$ instances of problem 'B', such that none of those instances are of size larger than $p(n)$

let Solve(B) $\leftarrow$ $O(t(n))$ algorithm for solving 'B' $t(n) \to$ nondecreasing function

To Solve: 'A'

→ Run the reduction algorithm for 'A' calling SolveB( ) whenever necessary.

Time spent in SolveB $= O(p(n)\, t(p(n)))$

∴ A can be solved in $= O(p(n)\, t(p(n)) + q(n))$

$q(n) \rightarrow$ time spent outside solveB( ) calls.

If $t(n)$ is a polynomial then $O(p(n)\,t(p(n)) + q(n))$ is also a polynomial, as sums, products and compositions of polynomials are polynomials.

Hence proved.

→ We know from $HAM \equiv_T^P HAMD$ that, a polynomial time algorithm exists to find Hamiltonian cycles iff a polynomial time algorithm exists to decide if a graph is Hamiltonian.

(ie) $HAMD \in P$ [it is equivalent to saying that]

→ Typical of many interesting problems which are polynomially equivalent to a similar decision problem. ⇒ 'decision reducible'.

Defn: Let X and Y be two decision problems defined on sets of instances I and J respectively. Problem X is polynomially many-to-one reducible to problem 'Y' if ∃ a polynomial time computable fn. $f: I \rightarrow J$ such that $x \in X$ iff $f(x) \in Y$ $\forall x \in I$ of problem X. This is denoted as $X \leq_m^P Y$ and the function 'f' is called the 'reduction function'.

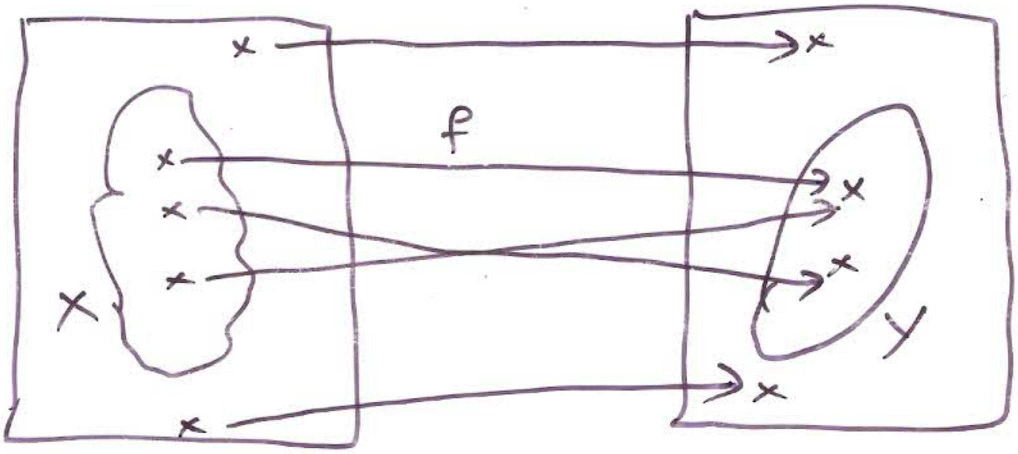
When $X \leq_m^p Y$ and $Y \leq_m^p X$ both hold, we say that X and Y are polynomially many-to-one equivalent and we write $X \equiv_m^p Y$.

(i.e.) The reduction function maps all yes-instances of X onto yes-instances of problem Y and all no-instances of X onto no-instances of Y.

$f \rightarrow$ to be computable in polynomial time
Size of $|f(x)|$ must be bounded above by some polynomial in the size of x $\forall x \in I$.

→ Useful in establishing Turing Reductions:

To decide if $x \in X$, we compute $y = f(x)$ and ask if $y \in Y$.



Many-one Reduction.

Theorem: If X and Y are two decision problems such that $X \leq_m^p Y$, then $X \leq_T^p Y$.

Proof:
DecideY → unit cost algorithm for Y.
f → reduction function from A to B computable in polynomial time.

Solves X in polynomial time ←

```
function DecideX(x) {
    y ← f(x)
    if DecideY(y) then return true;
                  else return false;
}
```