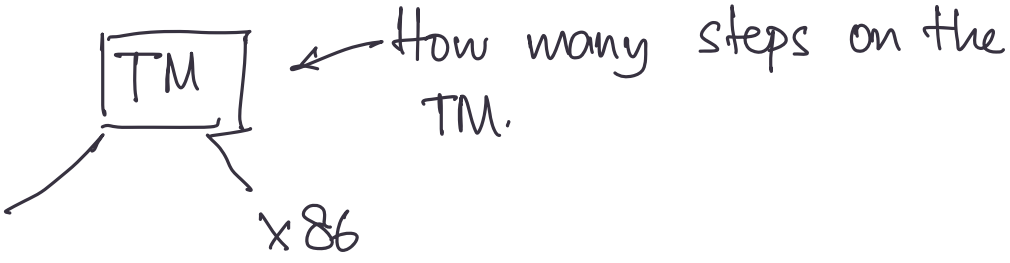
Efficiency of computation: P vs NP.

- Turing machines: Abstract machines to model computations.

"Turing machines simulate any computation".

Abstraction helps us to better express our runtime complexity.



Polynomial time algorithms: Algorithms that can be simulated on a TM in polynomial steps.

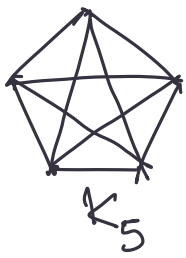
"Implicitly encodes that it is polytime bit operations".

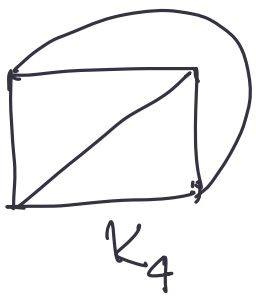
→ Does a given graph contain a clique of size k?

↳ Clique of size 3 is a triangle.

$\binom{n}{3} \cdot O(1)$.

Given k vertices by an "oracle/wizard" we can check if they form a clique.





$\binom{n}{k} \cdot \text{poly}(k)$ ← Brute force.

$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k$

↳ Exponential algorithm in k.

Vertex cover: Set of vertices such that for every edge one of its end points is in the set.

Qn: Is there a Vertex Cover of size at most k?

Brute force → $\binom{n}{k} \cdot m$.

Given a possible solution, we can verify if it is in fact a solution.

Integer factorization: Given an integer n (in binary repr) compute its prime factors.

Brute force: for i in $[1, \lfloor\sqrt{n}\rfloor]$:

Check if i divides n.

↳ Check for multiplicity.

Running time → $\sqrt{n} \cdot \text{poly}(\log n)$ $\quad N = \log n$

$= 2^{\frac{N}{2}} \cdot \text{poly}(N)$

Given a bunch of prime numbers with their multiplicities

we can check if they multiply to n.

$$(p_1, e_1), (p_2, e_2), \ldots, (p_k, e_k) : \longrightarrow \quad n \stackrel{?}{=} p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$$

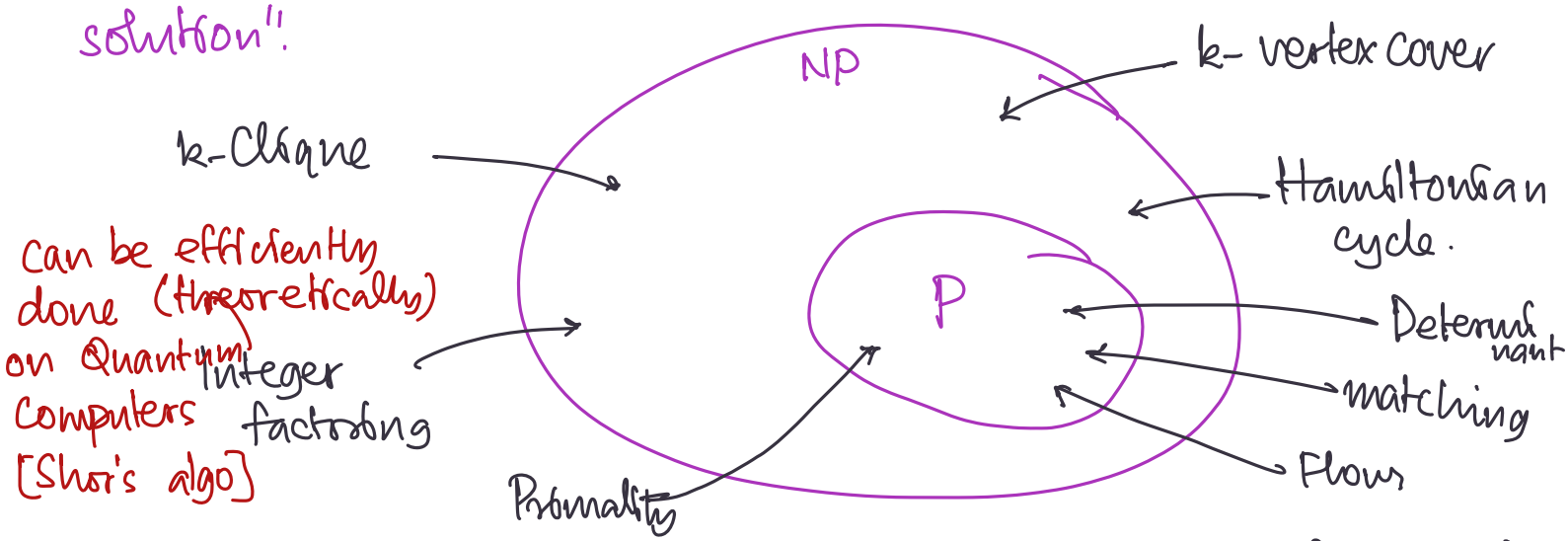
NP: Set of problems that have efficient verification
= algorithms.
↑ → polytime

P: Set of problems with efficient algorithms.

$P \subseteq NP$ ← Obtaining solution with an external help.
"polynomial length".

[Cook-Levin]

NP: External help provides a "witness/certificate/solution".



Hamiltonian cycle/circuit: Given a graph, check if there is a simple cycle that contains every vertex.

$v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ ← Check if they form a simple cycle.

Million-dollar problem:

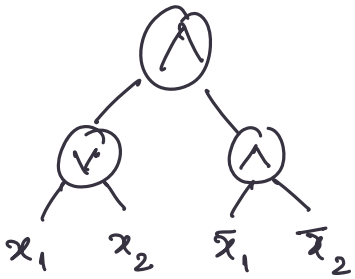
Are there problems in NP that do not have polynomial time algorithms.

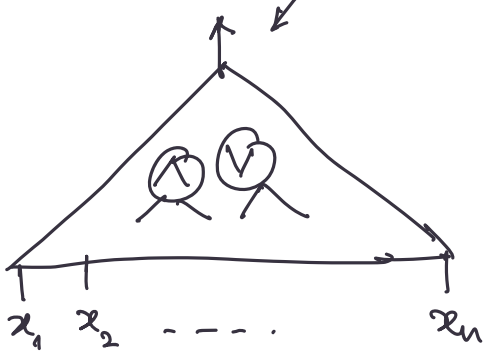
Circuit Satisfiability:

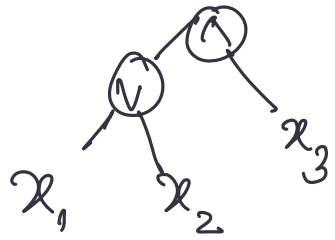
#gates = poly(n)

$\wedge, \vee, \neg$

$x_1, \ldots, x_n \in \{T, F\}$







Qn: Given a circuit ϕ, check if there is an assignment to the literals such that ϕ on that assignment outputs T.

Brute force: Run through all exponentially many assignments and check if any of them leads to an output of True.

Running time $\rightarrow$ $2^n \cdot |\phi|$.

Circuit-SAT $\in$ NP.

↳ "NP-hard"

↳ "NP-complete".

NP-hard: A problem Π is said to be NP-hard if any other problem $\Pi' \in NP$ can be solved with Π as a sub-routine.

"Ckt-SAT is NP-hard" [Cook-Levin theorem]

$\equiv$

"Every problem in NP can be solved efficiently if Ckt-SAT can be solved efficiently".

NP-complete: A problem Π is NP-complete if Π is NP-hard and $\Pi \in NP$.

"Ckt-SAT is NP-complete".