Computational Complexity

Hao Chung (鍾豪)

July 18, 2019

Self Introduction

台大電機碩士

前 DEXON foundation 區塊鏈研究員

2016 暑期密碼學課程學生

2017 暑期密碼學助教

研究興趣：quantum cryptography, blockchain

Outline

1. Big-Oh notation
2. Polynomial-Time Reduction
3. P v.s. NP

How to Compare Algorithms

Suppose we want to compute a function $f(n)$

- Algorithm 1 runs in $100{,}000n$ seconds
- Algorithm 2 runs in n^2 seconds

Which algorithm is better?

- Algorithm 2 runs faster if $n < 100{,}000$
- Algorithm 1 runs faster if $n > 100{,}000$

The ratio of running times goes to infinity

How to Compare Algorithms

Suppose we want to compute a function $g(n)$

- Algorithm 1 runs in $20n$ seconds
- Algorithm 2 runs in $10n + 30$ seconds

Which algorithm is better?

- Algorithm 2 runs faster when $n > 3$
- However, it is only twice as fast when $n \to \infty$

Big-Oh

Definition (Big-Oh)

Let $f(n)$ and $g(n)$ be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$. Then,

$$O\big(f(n)\big) = \{g(n) : \exists c, n_0 \text{ such that } 0 \leq g(n) \leq c \cdot f(n) \text{ for all } n > n_0\}$$

In human language, $g(n) \in O\big(f(n)\big)$ if

1. there exists a constant c and a threshold n_0 such that
2. $g(n) \leq c \cdot f(n)$ for all n larger than n_0

Remark: Sometimes we abuse the notation and write $g(n) = O\big(f(n)\big)$; however, you should aware that $O\big(f(n)\big)$ is a set.

Big-Oh Examples

Example

Let $f(n) = 10n + 30$ and $g(n) = 20n$.

Prove that $f(n) \in O\big(g(n)\big)$ and $g(n) \in O\big(f(n)\big)$.

Proof:

Let $c = 1, n_0 = 3$. Then,

If $n \geq n_0$, then $f(n) = 10n + 30 \leq 20n = cg(n)$.

Let $c = 2, n_0 = 1$. Then,

If $n \geq n_0$, then $g(n) = 20n \leq 2(10n + 30) = cf(n)$.

Big-Oh Examples

Example

Let $f(n) = 100{,}000n$ and $g(n) = n^2$.

Prove that $f(n) \in O\big(g(n)\big)$.

Big-Oh Examples

Example

Let $f(n) = n^4 + 3n + 500$.

Prove that $f(n) \in O(n^4)$.

Properties of Big-Oh

Big Oh allows us to ignore constant factors:

- For any constant c, we have $c \cdot f(n) \in O\big(f(n)\big)$.
- For example, addition costs 1 iteration and division costs 32 iteration on x86 CPU .

  But whether counting division as an elementary operation doesn't affect the time complexity.

Properties of Big-Oh

Big Oh allows us to ignore lower order terms:

- If $f(n) \in O(g(n))$, then $g(n) + f(n) \in O(g(n))$.
  - $O(n^4 + 3n^2 + 1) = O(n^4)$.
- If there are many parts in an algorithm, the most expensive part will dominate.

Big-Omega

Definition (Big-Omega)

Let $f(n)$ and $g(n)$ be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$. Then,

$$\Omega\big(f(n)\big) = \{g(n) : \exists c, n_0 \text{ such that } g(n) \geq c \cdot f(n) \geq 0 \text{ for all } n > n_0\}$$

In human language, $g(n) \in \Omega\big(f(n)\big)$ if

1. there exists a constant c and a threshold n_0 such that
2. $g(n) \geq c \cdot f(n)$ for all n larger than n_0

Big-Theta

Definition (Big-Theta)

Let $f(n)$ and $g(n)$ be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$. Then,

$$\Theta\big(f(n)\big) = \left\{ \begin{array}{c} g(n): \exists c_1, c_2, n_0 \text{ such that} \\ 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \text{ for all } n > n_0 \end{array} \right\}$$

In human language, $g(n) \in \Theta\big(f(n)\big)$ if

1. there exists two constants c_1, c_2 and a threshold n_0 such that
2. $c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$ for all n larger than n_0

Examples

Example

Let $f(n) = n^4 + 3n + 500$.

Prove that $f(n) \in \Omega(n^3 \log n)$ and $f(n) \in \Theta(n^4)$.

Example

Prove that $\log n \in O(n^k)$, for all $k > 0$.

Example

Prove that $n^k \in O(n^{\log n})$, for all $k > 0$.

Example

Let $f(n) = \frac{n}{\log n}$ and $g(n) = n^{1-\delta}$ for some $\delta \in (0,1)$.

Prove that $f(n) \in O(g(n))$ or $g(n) \in O(f(n))$.

Outline

1. Big-Oh notation
2. Polynomial-Time Reduction
3. P v.s. NP

Probabilistic Polynomial-Time (PPT)

In cryptography, we usually define the power of adversary as a probabilistic polynomial-time (PPT) algorithm.

An algorithm is polynomial-time if its running time is $O(n^k)$ for some k.

An algorithm is probabilistic (or randomized) if we allow the algorithm to "toss a coin" at each step.

Example (Primality-Test)

1. randomly choose $a \in \{2, \cdots, N-1\}$
2. if $a^{N-1} \neq 1$, output N is a composite
3. run step 1. and step 2. many times
4. if none of $a^{N-1} \neq 1$ in previous steps, output N is likely a prime

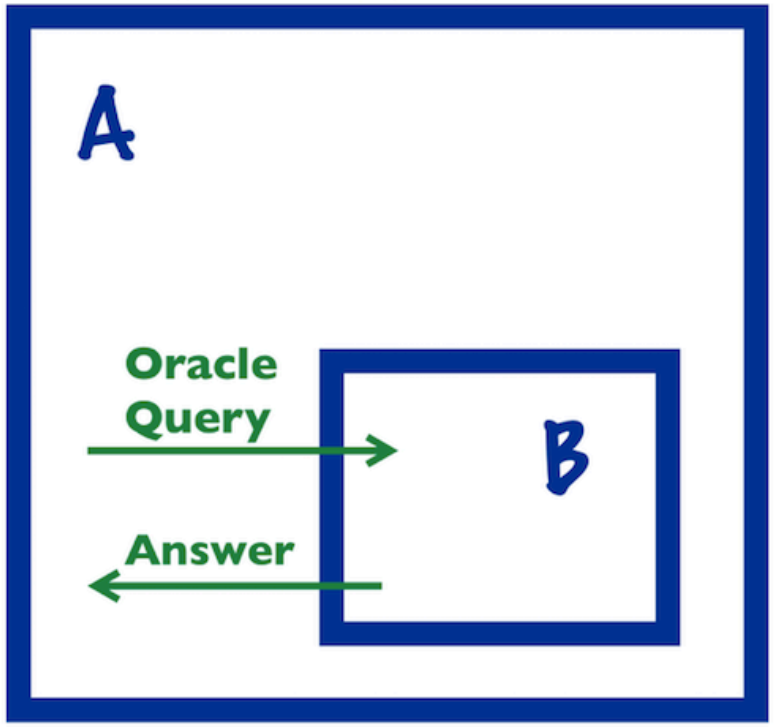
Polynomial-Time Reduction

Problem $\mathcal{A}$ is polynomial-time reducible to problem $\mathcal{B}$ if

1. given an oracle of $\mathcal{B}$
2. $\mathcal{A}$ can be solved in polynomial-time

We write "$\mathcal{A} \leq_p \mathcal{B}$" to denote that $\mathcal{A}$ is polynomial-time reducible to $\mathcal{B}$.

In this case, we say that problem $\mathcal{A}$ is no harder than problem $\mathcal{B}$.



RSA Encryption

ALICE

Message $x = 4$

BOB

1. Choose $p = 3$ and $q = 11$
2. Compute $n = p * q = 33$
3. $\Phi(n) = (3-1) * (11-1) = 20$
4. Choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$

$K_{pub} = (33,3)$ ←

$y = x^e \equiv 4^3 \equiv 31 \bmod 33$

y = 31 →

$y^d = 31^7 \equiv 4 = x \bmod 33$

Chapter 7 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

RSA Problem

Definition (RSA problem)

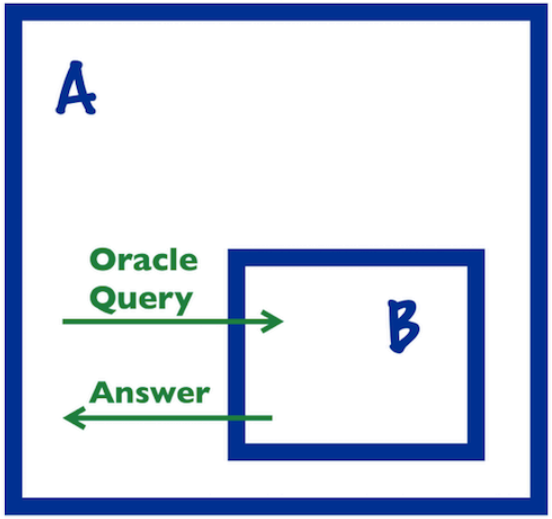
Given N, e, c such that $c \in \mathbb{Z}_N$ and $\gcd\big(e, \phi(N)\big) = 1$,

find m such that $m^e \equiv c \ (mod\ N)$.

Suppose we have a factoring oracle $\mathcal{O}$.

Then, we construct a RSA solver as follow.

1. query $\mathcal{O}$ and get $N = p \cdot q$
2. compute $\phi(N) = (p-1)(q-1)$
3. compute $d = e^{-1}\ \big(mod\ \phi(N)\big)$
4. compute $m = c^d\ (mod\ N)$



Hence, we can say that RSA problem $\leq_p$ factoring problem

Reduction in Security Proof

In cryptography, we often show the security of a scheme E by

assumption (known difficulty) $\leq_p$ **breaking E**

Outline

1. Big-Oh notation
2. Polynomial-Time Reduction
3. P v.s. NP

P and NP

Definition (**P**) (informal)

P is the set of problems that can be solved in polynomial time.

Definition (**NP**) (informal)

NP is the set of problems that can be checked in polynomial time given a solution.

Strictly speaking, P and NP only include decisional problems.

Is primality test in P?

Definition (Primes problem)

Given an integer N, decide whether N is a prime or not.

In 2002, Manindra Agrawal, Neeraj Kayal and Nitin Saxena finally showed that Primes problem is in **P**.

If we can try the division up to $\sqrt{N}$, why Primes problem $\in$ **P** doesn't hold trivially?

Key: the running time is counted in input size.

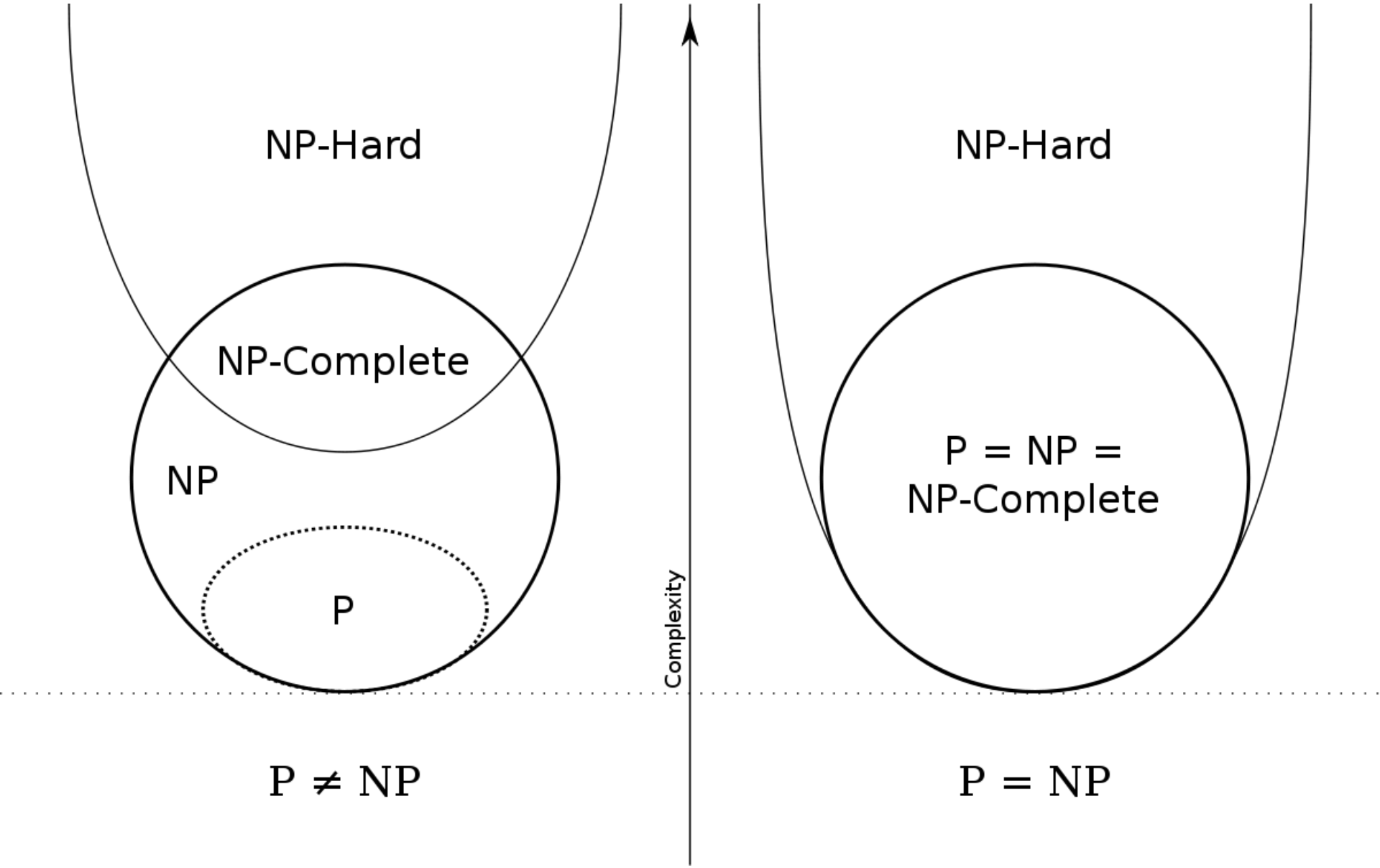
P and NP

Definition (P)

P is the set of decisional problems that can be solved in time $O\big(p(n)\big)$ for some polynomial $p(n)$ in the input size n.

Definition (NP)

NP is the set of decisional problems that can be checked in time $O\big(p(n)\big)$ for some polynomial $p(n)$ in the input size n given a solution.


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP