Introduction to Algorithms and Data Structures

Lesson 6: Foundation of Algorithms (3)
Big-O notation

Professor Ryuhei Uehara,

School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

http://www.jaist.ac.jp/~uehara

Big-O notation

- Big-O notation (Bachmann-Landau notation)
  - Big-O notation: O(f(n))
  - Big-Ω notation: Ω(f(n))
  - Θ notation: Θ(f(n))

Paul Bachmann
1837–1920

Edmund Landau
1877–1938

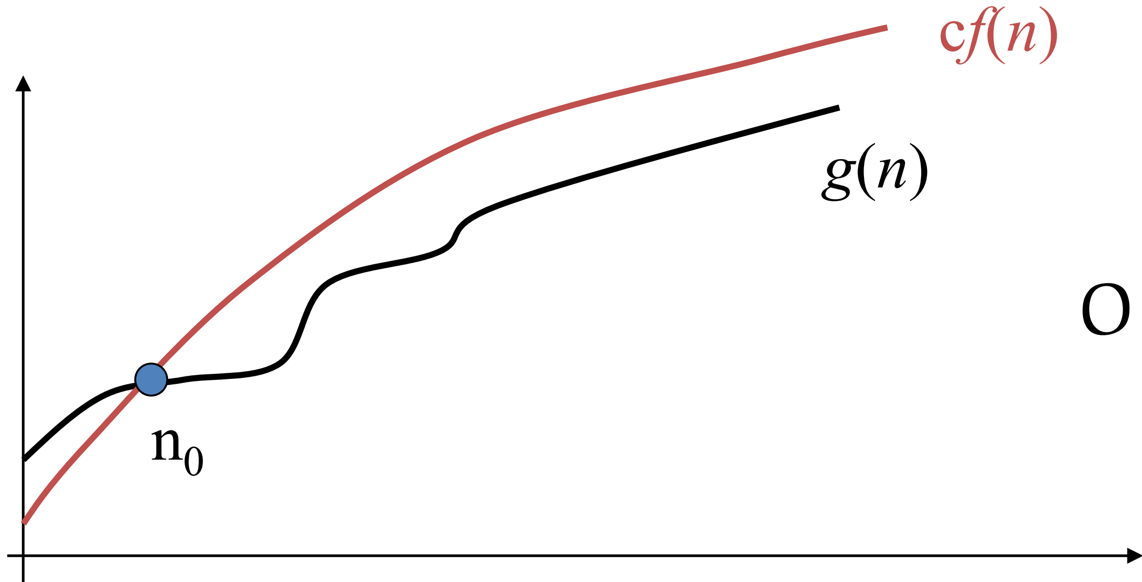
- We have three more, small-o notations, but we don't use in this lesson.

Asymptotical Complexity

- It indicates the behavior of complexity when the size n of input grows quite huge.
- We'd like to check how complexity grows (independent to machine model and/or programming techniques)➔
  - It is enough to consider main/major term
  - Coefficients are not essential from this viewpoint
- Three types:
  - Upper bound
  - Lower bound
  - Both of them

Big-O notation: O(f(n))
Upper bound of complexity

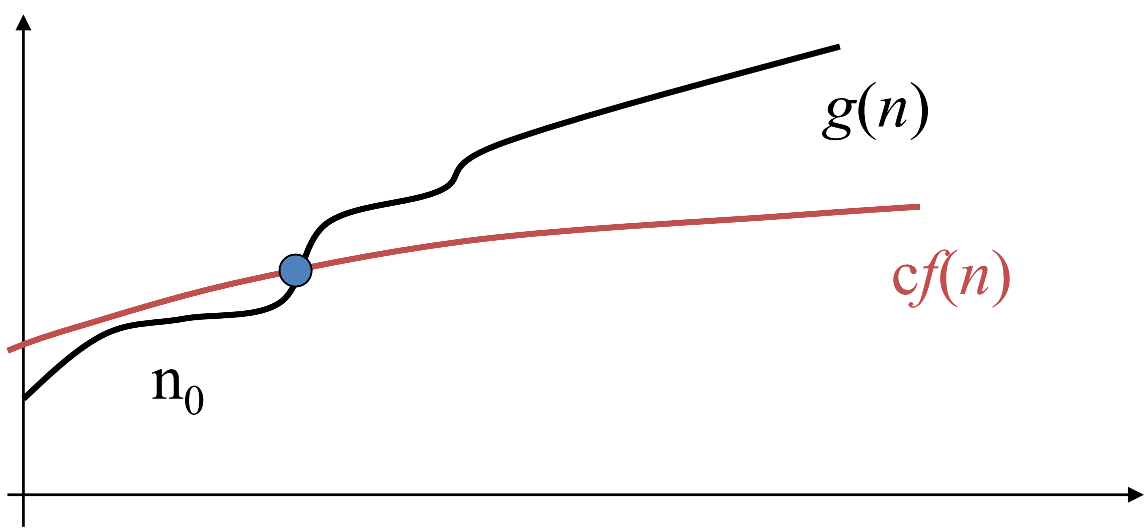
- $O(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0, \forall n \geq n_0, g(n) \leq cf(n)\}$
  - There exist two positive constants c and n_0 such that $g(n) \leq cf(n)$ for every $n \geq n_0$
  - Sometimes $g(n) = O(f(n))$ is used as $g(n) \in O(f(n))$
- Example of $f(n)$: $\log_2 n,\ n^2,\ 2^n,\ \ldots$



$O(\log n)$?

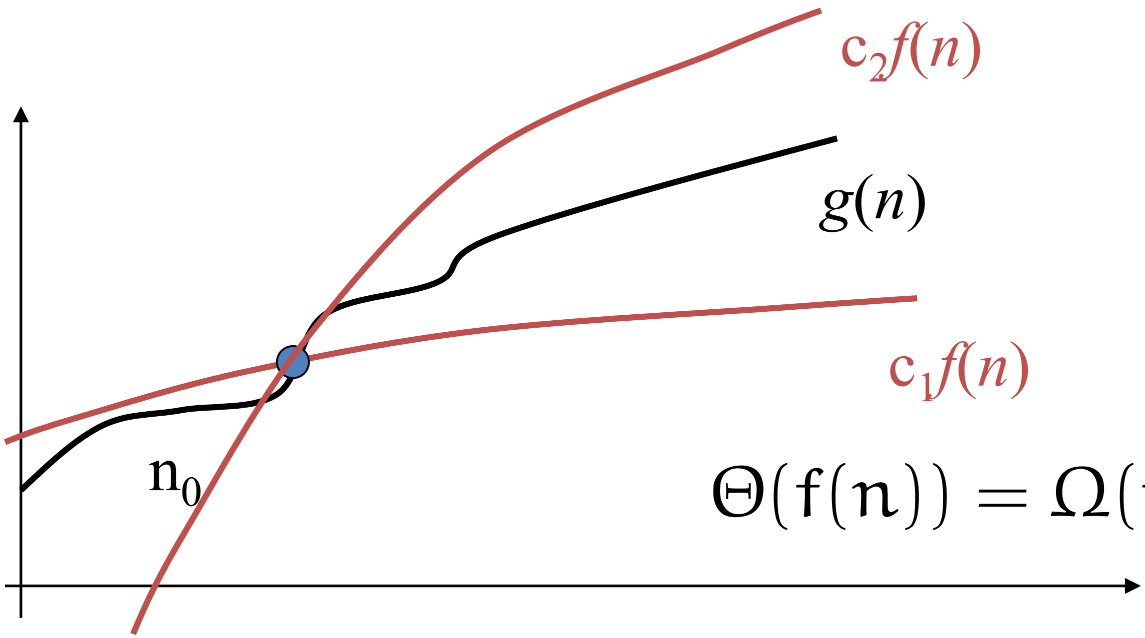
Big-Ω notation: Ω(f(n)) Lower bound of complexity

- $\Omega(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0, \forall n \geq n_0, cf(n) \leq g(n)\}$
  - There exist two positive constants c and n_0 such that $cf(n) \leq g(n)$ for every $n \geq n_0$



Θ notation: Θ(f(n))

- $\Theta(f(n)) = \{g(n) \mid \exists c_1, c_2 > 0, \exists n_0, \forall n \geq n_0, c_1 f(n) \leq g(n) \leq c_2 f(n)\}$
  - There exist three positive constants c_1, c_2, n_0 such that $c_1 f(n) \leq g(n) \leq c_2 f(n)$ for every $n \geq n_0$



$$\Theta(f(n)) = \Omega(f(n)) \cap O(f(n))$$

Report Problem 2

1. Choose functions in $O(n)$, $O(2^n)$
   – $0.1n$, $5n^{1000}$, 2.1^n, 2^{n+3}

2. Prove $23n^2 + n + 2018 \in O(n^2)$

3. Disprove $23n^3+n+2018 \in O(n^2)$

4. Prove $O(\log_2 n) = O(\log_{10} n)$

[Warning]
To (dis)prove, you need to follow the **definition**

Supplements: exponential, polynomial, and logarithm

1. A problem is solvable if there is an algorithm that solves the problem.
2. A problem is tractable if there is an algorithm that solves the problem in polynomial time of the length of the input.
3. A problem is intractable if we have no polynomial time algorithm.