CP312
Algorithm Design and Analysis I

LECTURE 3: CHARACTERIZING RUNNING TIME

Simplifying the Running Time Expression

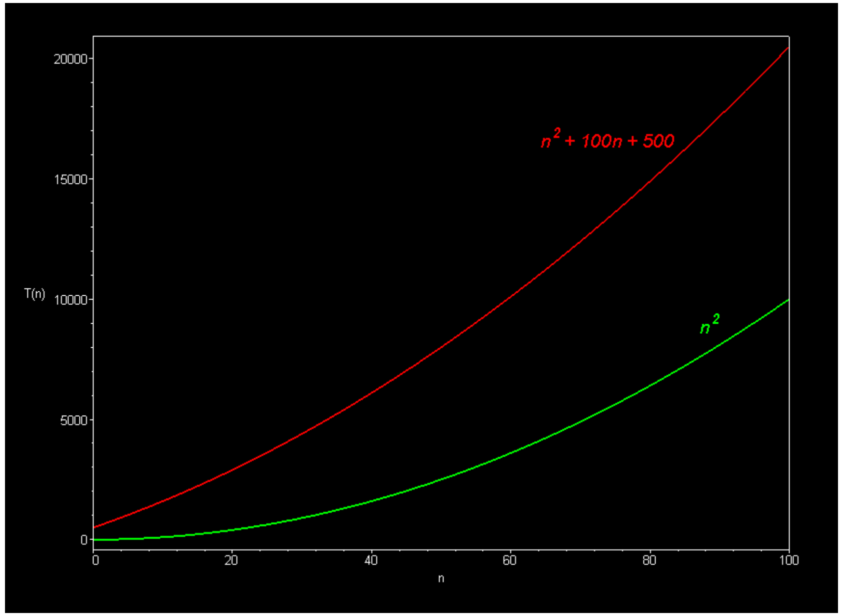
- Consider the following running time:

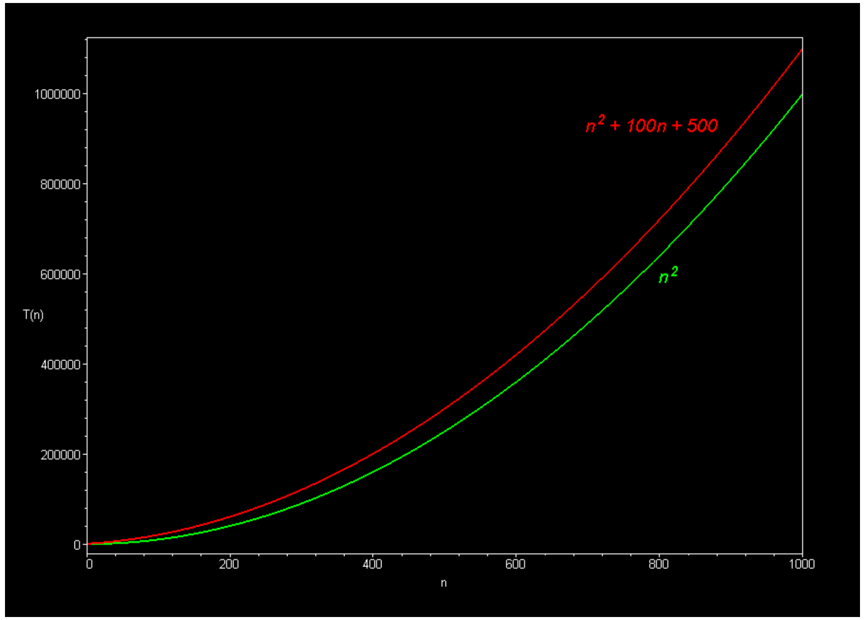
$$T(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3 + \cdots + a_d n^d$$

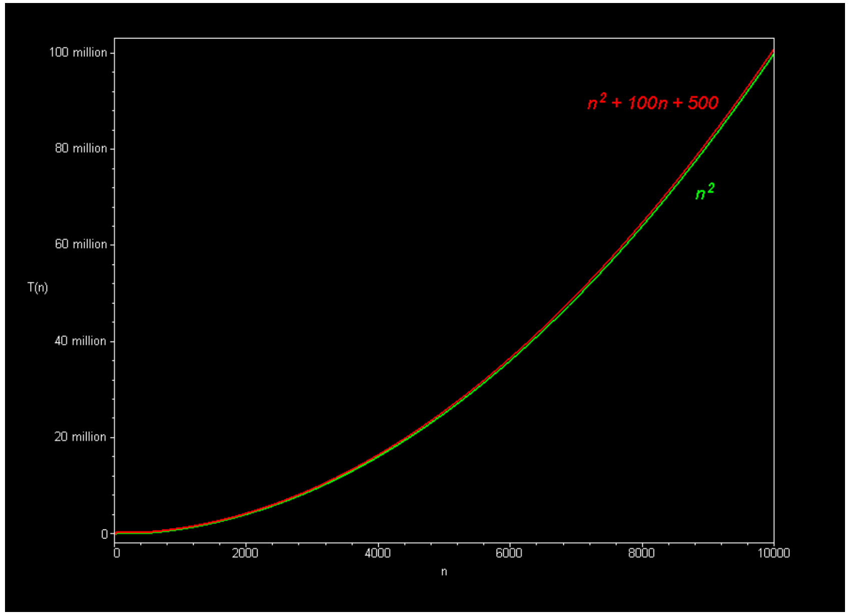
  - Too complicated
  - Too many terms
  - Difficult to compare two expressions of the same form
  - Do we really need that many terms?

- Example: $T(n) = 10n^3 + n^2 + 40n + 800$
  - If $n = 1000$, then $T(n) = 10{,}001{,}040{,}800$ whereas $10n^3 = 10{,}000{,}000{,}000$
  - If we approximate and drop all but the n^3 term the error is 0.01%
  - So, it is worth simplifying a complexity expression to get the core factor in that expression

Simplifying the Running Time Expression







- Only the **dominant terms** of a polynomial matter in the long run
- Lower-order terms fade to insignificance as the problem **input size increases**
- We care more about **scalable** algorithms than those for specific small-size inputs

Growth Rate of Running Time

- For any given running time function, in order to write it in the best way that represents the general growth rate.

1. We consider only the most dominant term
   - Keep the fastest growing term and remove the lower-order terms
2. Constant coefficients are removed
   - These constants represent language- or machine-dependent overhead
   - Growth rate not affected by constant coefficients

- Examples:
  - $T(n) = 100n + 10^5$ is considered a linear function
  - $T(n) = 80n^2 + 50n + 10$ is considered a quadratic function

Asymptotic Complexity

For large enough n:
$T(n) \approx g(n)$

- Finding the **exact** complexity, $T(n) =$ number of **primitive operations**, of an algorithm is difficult.
- Therefore, we **approximate** $T(n)$ by a function $g(n)$ in a way that does not substantially change the magnitude of $T(n)$
  - The function $g(n)$ is sufficiently close to $T(n)$ for **sufficiently large values of the input size n**.
- This "approximate" measure of efficiency is called **<u>asymptotic complexity</u>**.
- Thus, the asymptotic complexity measure does not give the exact number of operations of an algorithm, but it **shows how that number grows with the size of the input**.
  - This gives us a measure that will work for different operating systems, compilers and CPUs.

Asymptotic Complexity

- Three main types of asymptotic complexity expressions:

1. Big-O
   - Express asymptotic upper bounds
2. Big-Ω
   - Express asymptotic lower bounds
3. Big-Θ
   - Express asymptotic tight bounds

$$T(n) = O(g(n))$$
$\Rightarrow$ For large enough $n,\ T(n) \leq cg(n)$

$$T(n) = \Omega(g(n))$$
$\Rightarrow$ For large enough $n,\ T(n) \geq cg(n)$

$$T(n) = \Theta(g(n))$$
$\Rightarrow$ For large enough $n,\ c_1 g(n) \leq T(n) \leq c_2 g(n)$

- The term asymptotically means "for large enough n"

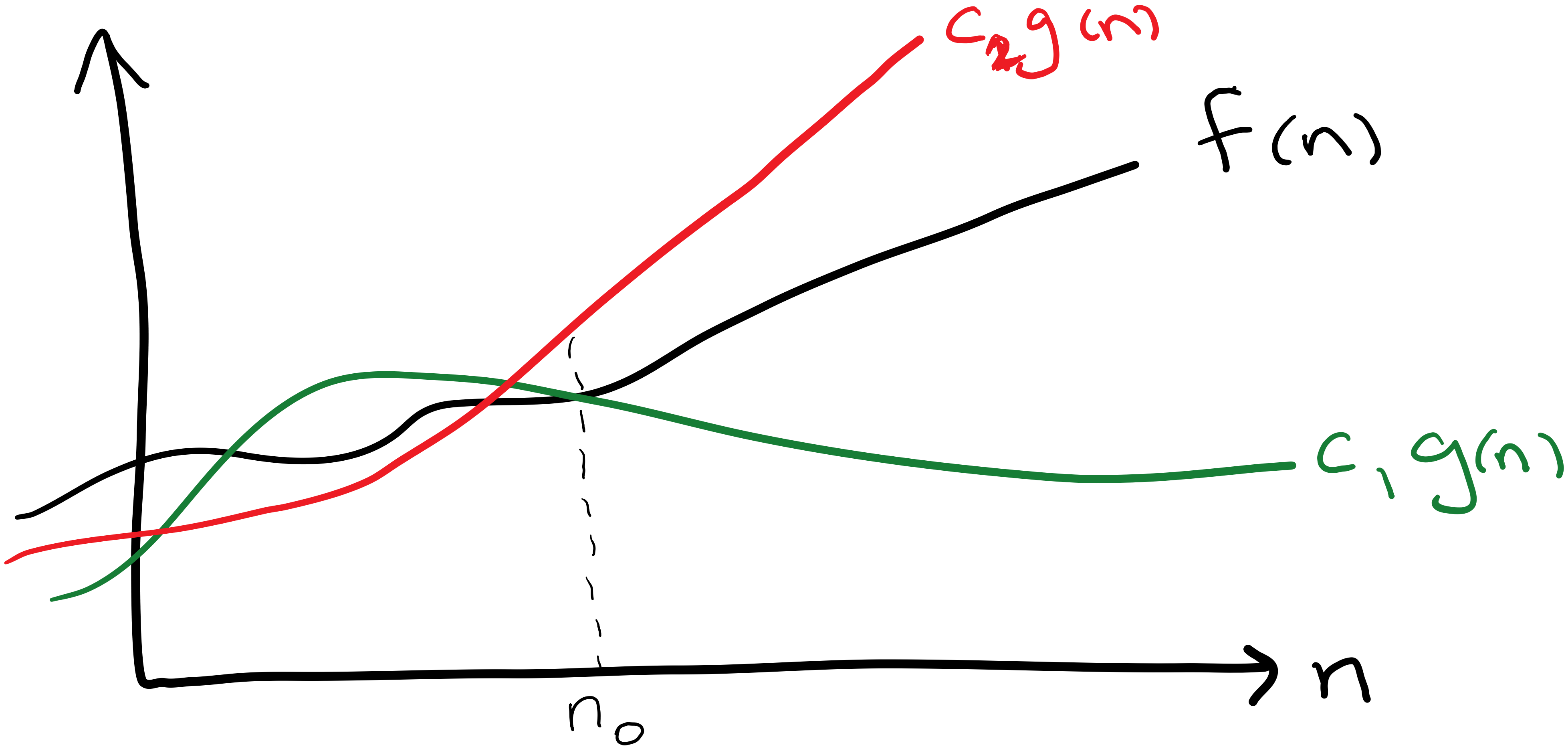
Asymptotic Notation

- Recall: Informally, we said that $f(n) = \Theta\big(g(n)\big)$ if $f(n) = g(n)$ after removing lower order terms and constant factors.

- **Definition**: For a given function $g(n)$, we denote by $\Theta\big(g(n)\big)$ the set of functions:

$$\Theta\big(g(n)\big) = \left\{ f(n) \,\middle|\, \begin{matrix} \exists c_1, c_2, n_0 > 0 \text{ such that } \forall n \geq n_0 \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \end{matrix} \right\}$$

Asymptotic Notation Θ



Asymptotic Notation Θ

- Example: $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

Pick $c_1 = \frac{1}{14}, c_2 = \frac{1}{2}, n_0 = 7$

Asymptotic Notation Θ

- Example: Is $4n^4 = \Theta(n^2)$? NO

Proof by Contradiction:

- Assume there exists c_2 and n_0 such that $4n^4 \leq c_2 n^2$ for all $n \geq n_0$
- Then $n^2 \leq c_2/4$ for all $n \geq n_0$
- Which is NOT TRUE since c_2 is a constant → Contradiction

Asymptotic Notation O

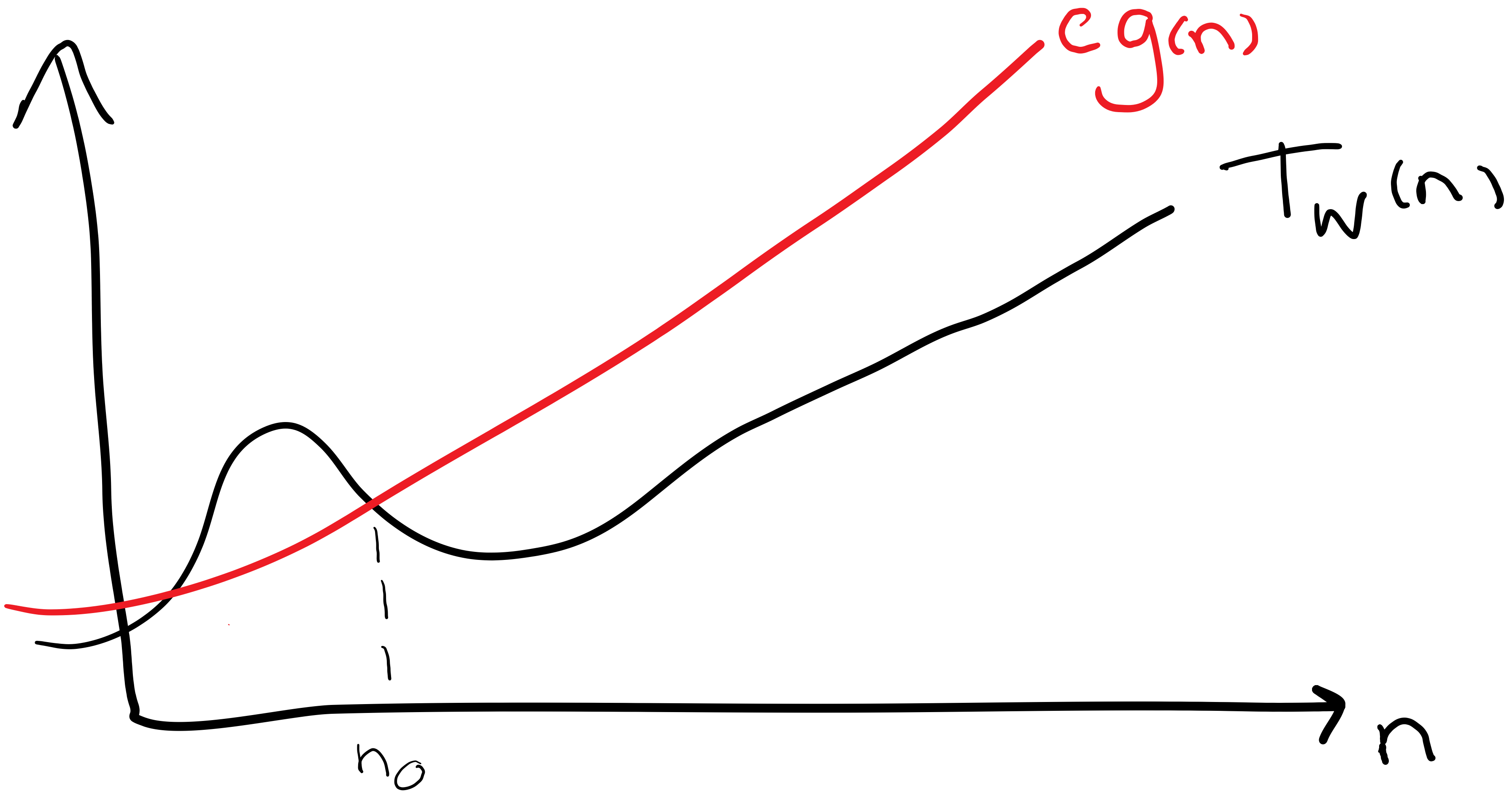
- **Definition**: For a given function $g(n)$, we denote by $O\big(g(n)\big)$ the set of functions:

$$O\big(g(n)\big) = \left\{ f(n) \,\middle|\, \begin{matrix} \exists c, n_0 > 0 \text{ such that } \forall n \geq n_0 \\ f(n) \leq cg(n) \end{matrix} \right\}$$

Asymptotic Notation O

- We use O-notation to give an **upper bound** on a function.
- $f(n) = \Theta\big(g(n)\big) \implies f(n) = O\big(g(n)\big)$
- Suppose $T_w(n)$ is the **worst-case** running-time of an algorithm (on input w) and $T_y(n)$ is the running-time of an algorithm on **any** input y. Then $T_w(n) = O\big(g(n)\big) \implies T_y(n) = O\big(g(n)\big)$

Asymptotic Notation O



Asymptotic Notation O

- Examples:
- Is $2^{n+1} = O(2^n)$? YES

Asymptotic Notation O

- Examples:
- Is $2^{2n} = O(2^n)$? NO

Asymptotic Notation O

- Examples:
- Is $2^{2n} = 2^{O(n)}$? YES

Asymptotic Notation O

- Examples:
- Is $\log_{10}(n) = O(\log_2(n))$? YES

Asymptotic Notation O

- Examples:
- Is $n^{2.5} = O\left(n^{2.8}\right)$? YES

Asymptotic Notation O

- Examples:
- Is $n^{\log n} = O\left(n^5\right)$? NO

Asymptotic Notation Ω

- **Definition**: For a given function $g(n)$, we denote by $\Omega\big(g(n)\big)$ the set of functions:

$$\Omega\big(g(n)\big) = \left\{ f(n) \,\middle|\, \begin{array}{c} \exists c, n_0 > 0 \text{ such that } \forall n \geq n_0 \\ 0 \leq cg(n) \leq f(n) \end{array} \right\}$$

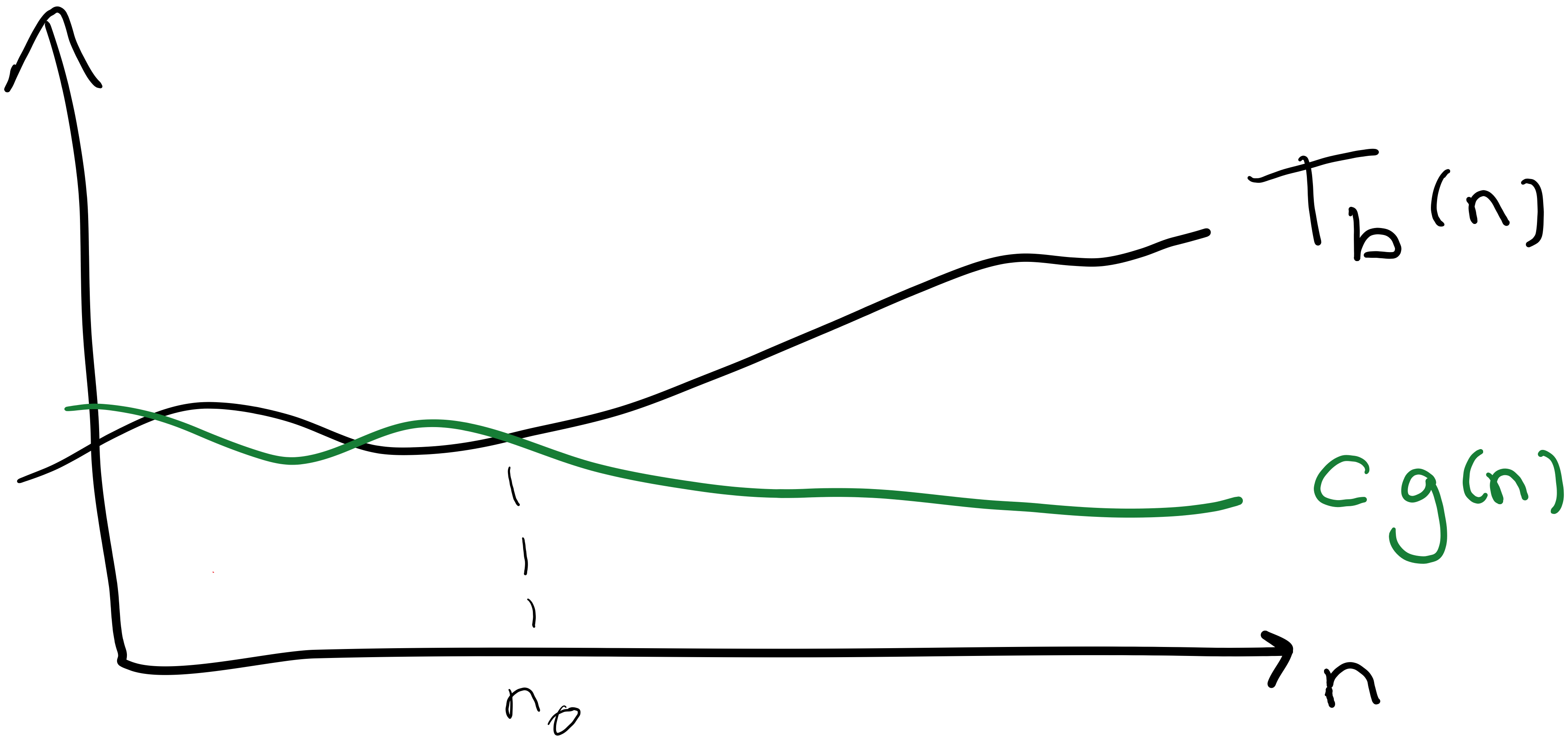
Asymptotic Notation Ω

- We use Ω-notation to give an **lower bound** on a function.

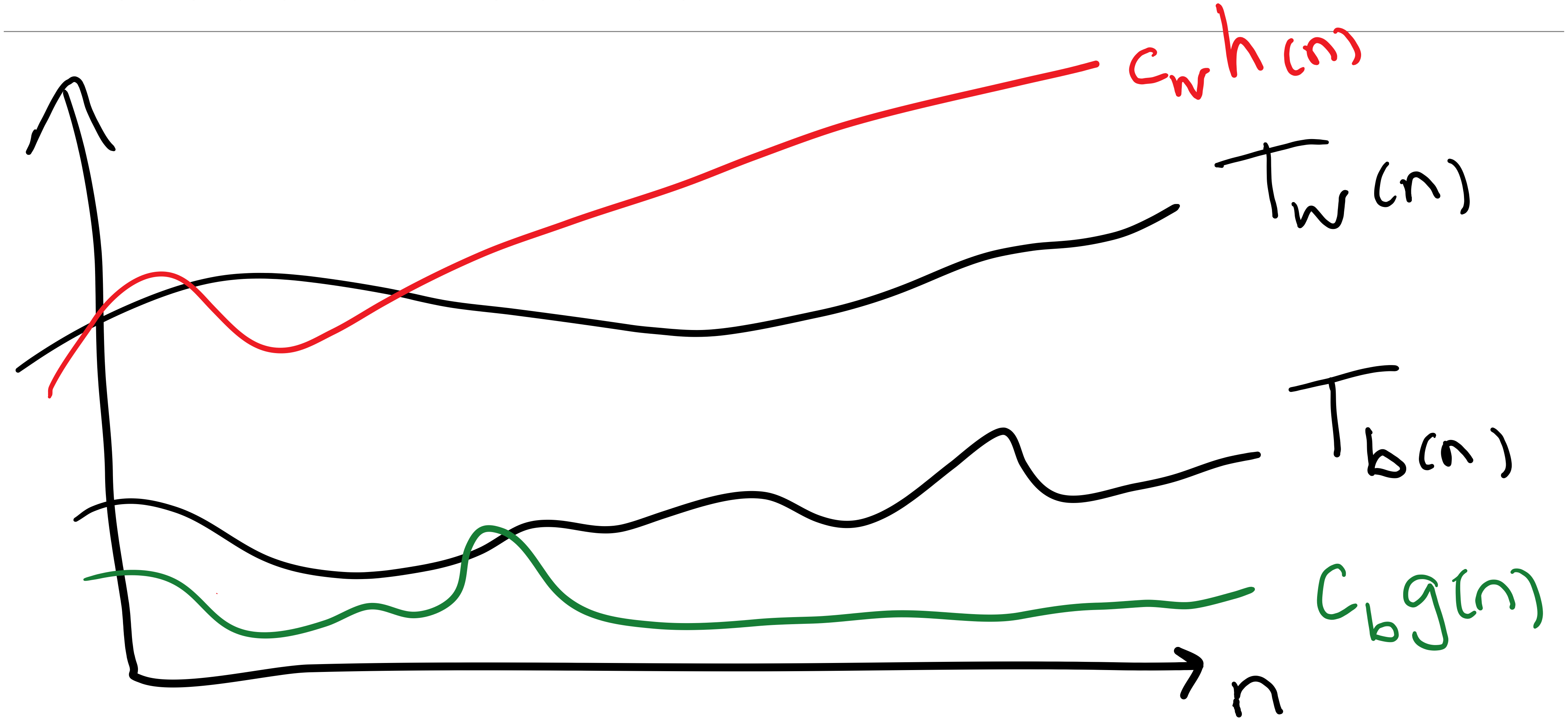
- $f(n) = \Theta(g(n)) \implies f(n) = \Omega(g(n))$

- Suppose $T_b(n)$ is the **best-case** running-time of an algorithm (on input b) and $T_y(n)$ is the running-time of an algorithm on **any** input y. Then $T_b(n) = \Omega(g(n)) \Longrightarrow T_y(n) = \Omega(g(n))$

Asymptotic Notation Ω



The Useful Bounds



Asymptotic Notation o and ω

$$o(g(n)) = \left\{ f(n) \,\middle|\, \begin{array}{c} \forall \boldsymbol{c} \\ \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \\ f(n) < cg(n) \end{array} \right\}$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

$$\omega(g(n)) = \left\{ f(n) \,\middle|\, \begin{array}{c} \forall \boldsymbol{c} \\ \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \\ cg(n) < f(n) \end{array} \right\}$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

Asymptotic Notation o and ω

Examples:

- Is $10n = o(n)$? NO
- Is $2n^2 = \omega(n)$? YES

Asymptotic Notation Properties

- Transitivity for $\Theta, O, \Omega, \mathrm{o}, \omega$
  - E.g. If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$
- Reflexivity for Θ, O, Ω
  - E.g. $f(n) = \Theta(f(n))$
- Symmetry for Θ
  - $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose Symmetry for $O, \Omega, \mathrm{o}, \omega$
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$
  - $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$

Asymptotic Notation Properties

- Using asymptotic notation in equations:

- $8n^2 + 7\text{n} + 10 = 8\text{n}^2 + O(n)$
  - $\Longrightarrow 8n^2 + 7n + 10 = 8n^2 + f(n)$ for ***some*** $f(n) = O(n)$

- $8n^2 + O(n) = O(n^2)$
  - $\Longrightarrow$ For any $g(n) = O(n)$, $8n^2 + g(n) = f(n)$ for ***some*** $f(n) = O(n^2)$

Exercises

- **EX:** Given that $f(n) = O\left(n^3\right) + O\left(n^2 \lg n\right)$, simplify $f(n)$ so that only a single big-O is used.

Exercises

- **EX:** List the following functions from slowest to fastest (c is an arbitrary constant):
  - $O(\log n)$
  - $O(n^2)$
  - $O(c^n)$
  - $O(1)$
  - $O((\log n)^c)$
  - $O(n^c)$
  - $O(n)$

Notation	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O((\log n)^c)$	Polylogarithmic
$O(n)$	Linear
$O(n^2)$	Quadratic
$O(n^c)$	Polynomial
$O(c^n)$	Exponential

$$O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(n^3) \subseteq O(2^n)$$