UNIVERSITÀ DEGLI STUDI DI PADOVA

Asymptotical Analysis

Gianmaria Silvello

Department of Information Engineering
University of Padua

gianmaria.silvello@unipd.it

http://www.dei.unipd.it/~silvello/

Running Time

- In some cases we can define the exact running time of an algorithm
  - See Insertion Sort

Running Time

- In some cases we can define the exact running time of an algorithm
    - See Insertion Sort

- Nevertheless, the extra precision is not useful
    - Constants do not have a real impact on running times

Running Time

- In some cases we can define the exact running time of an algorithm
  - See Insertion Sort
- Nevertheless, the extra precision is not useful
  - Constants do not have a real impact on running times
- For input “large enough” we can get rid of multiplicative factors and <u>lower order terms</u>
  - Remember when we discussed linear vs quadratic running time

Asymptotic Analysis

- We are concerned about how the running time of the algorithm grows as the size of the input goes to the limit

Asymptotic Analysis

- We are concerned about how the running time of the algorithm grows as the size of the input goes to the limit
- Say the input has size n then we care about how the algorithm performs when $n \rightarrow +\infty$

Asymptotic Analysis

- We are concerned about how the running time of the algorithm grows as the size of the input goes to the limit
- Say the input has size *n* then we care about how the algorithm performs when $n \rightarrow +\infty$
- An algorithm A which is asymptotically better than an algorithm B will perform better for all, but very small inputs
  - $n \geq n_0$ where n_0 is a small value selected ad-hoc

Asymptotic Analysis

- Let us consider two algorithms:
  - Algorithm A with $T_A(n) \sim n^2$
  - Algorithm B with $T_B(n) \sim n$

- Algorithm B is better than Algorithm A for all $n \geq n_0$

Asymptotic Notation

- We use asymptotic notation to describe the running time of algorithms
  - For insertion sort we said that the running time is
    - $T(n) = an^2 + bn + c$
    - Using asymptotic notation we might say that $T(n) = \Theta(n^2)$
  - With asymptotic notation we abstract away some details of functions

Asymptotic Notation

- We use asymptotic notation to describe the running time of algorithms
  - For insertion sort we said that the running time is
    - $T(n) = an^2 + bn + c$
    - Using asymptotic notation we might say that $T(n) = \Theta(n^2)$
  - With asymptotic notation we abstract away some details of functions
- With asymptotic notation we mainly focus on time, but it can be used to characterise any other aspect of an algorithm (e.g., space)

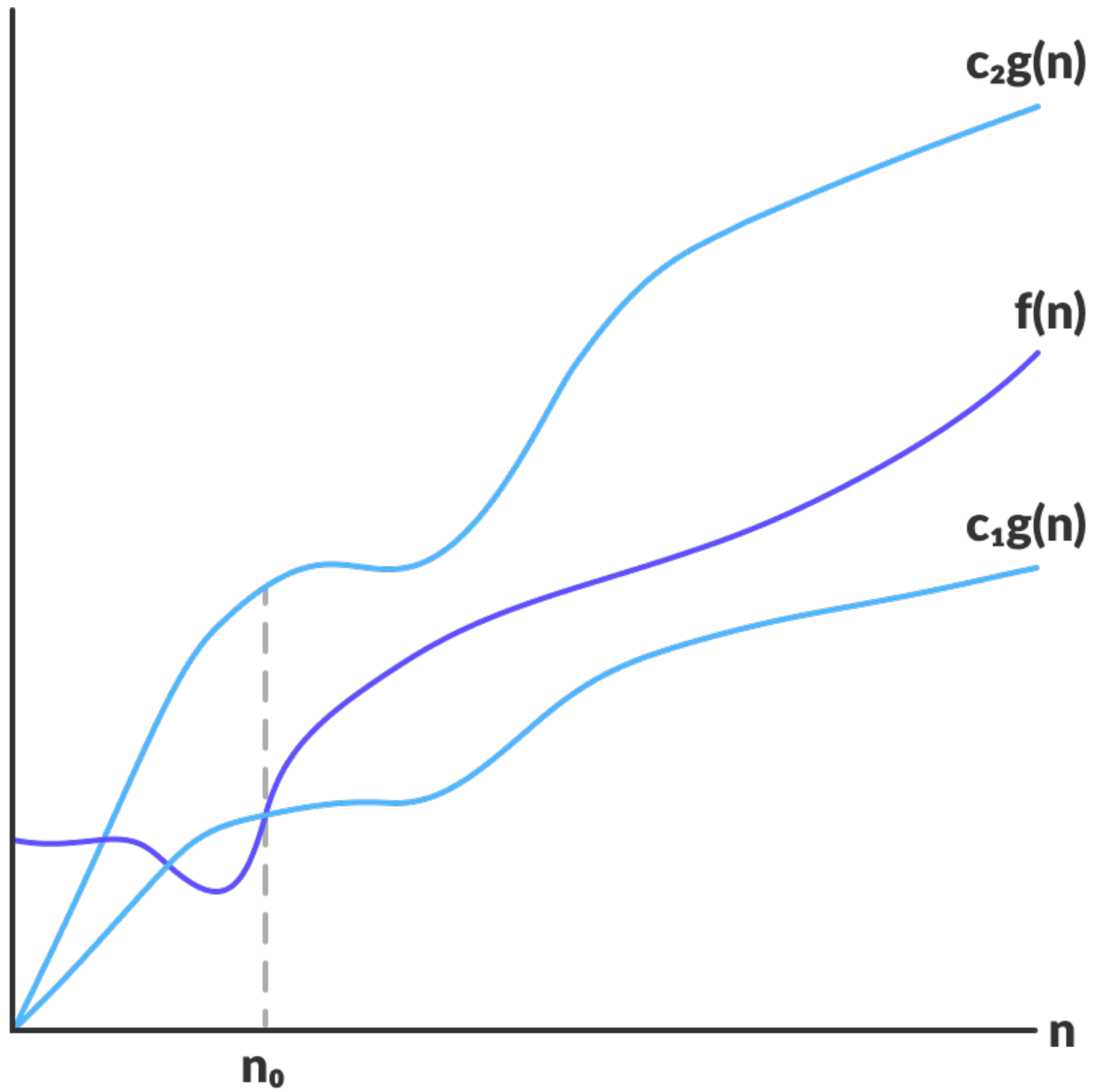
Big-Theta Notation $\boldsymbol{\Theta}$

- Insertion sort has $\boldsymbol{\Theta}$(n²) worst-case running time
  - What does this mean?

$$\Theta(g(n)) = \{f(n) : \text{ there exists positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

Big-Theta Notation Θ

$$\Theta(g(n)) = \{f(n) : \text{ there exists positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

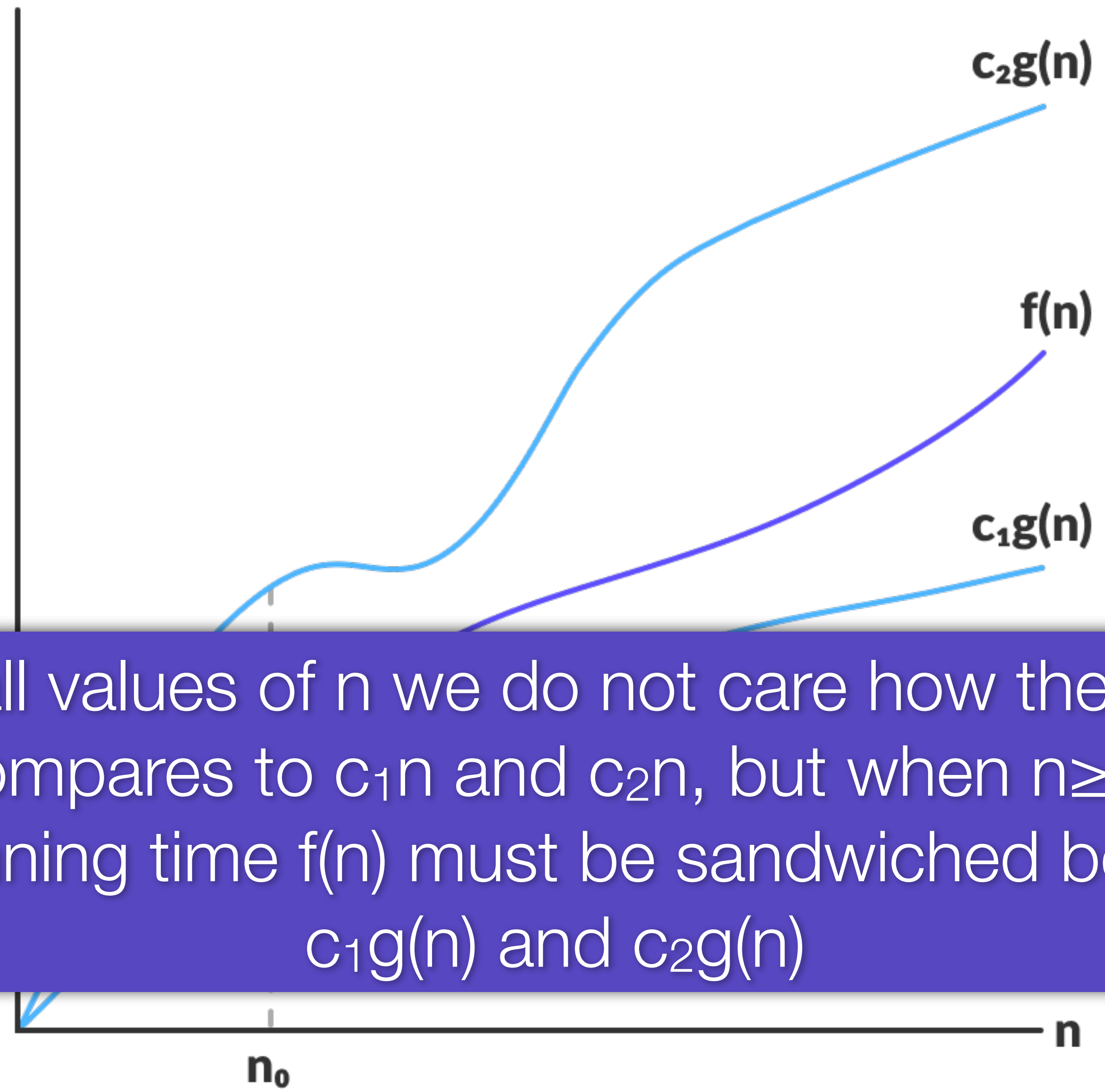


Source: https://www.programiz.com/dsa/asymptotic-notations

f(n) = Θ(g(n))

Big-Theta Notation Θ

$$\Theta(g(n)) = \{f(n) : \text{ there exists positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}$$



For small values of n we do not care how the running time compares to c_1n and c_2n, but when $n \geq n_0$ then the running time f(n) must be sandwiched between $c_1g(n)$ and $c_2g(n)$

Source: https://www.programiz.com/dsa/asymptotic-notations

$f(n) = \Theta(g(n))$

Big-Theta Notation Θ

- We write $f(n) = \Theta(g(n))$ with a little abuse of notation
  - we should write $f(n) \in \Theta(g(n))$ or f(n) *is* $\Theta(g(n))$
- We say that g(n) is an asymptotically TIGHT BOUND for f(n)

Big-oh notation

- *Big-oh* (or only *oh*) notation defines an upper bound
  - It says that an algorithm cannot go <u>slower</u> than big-oh

Source: https://www.programiz.com/dsa/asymptotic-notations

Big-oh notation

- *Big-oh* (or only *oh*) notation defines an upper bound
  - It says that an algorithm cannot go <u>slower</u> than big-oh

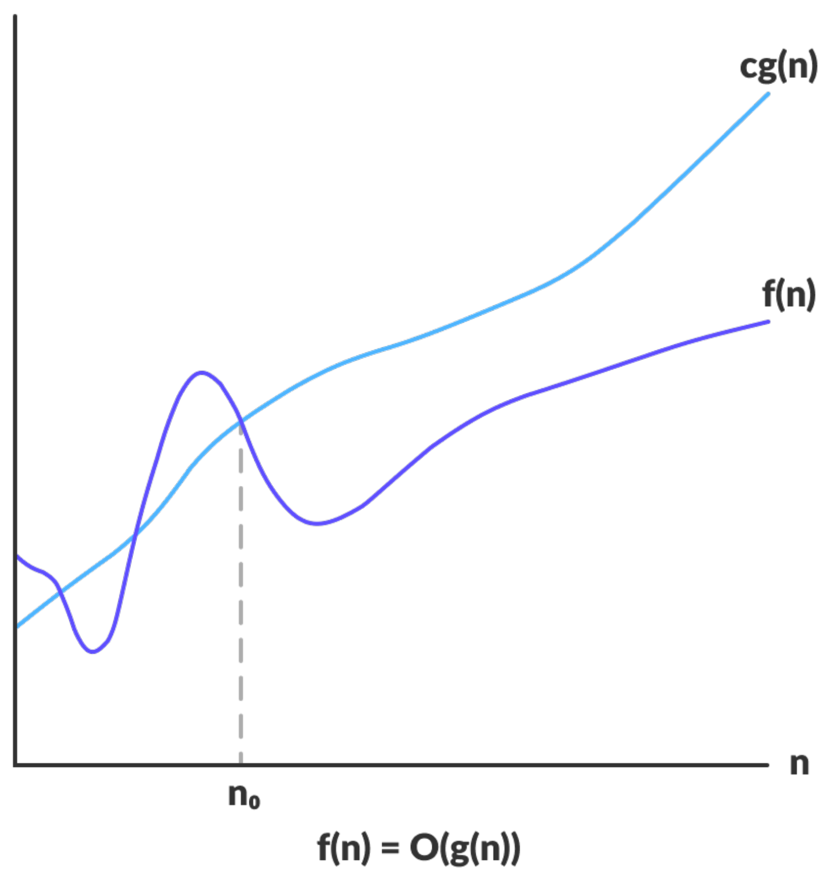
$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

Source: https://www.programiz.com/dsa/asymptotic-notations

Big-oh notation

- *Big-oh* (or only *oh*) notation defines an upper bound
  - It says that an algorithm cannot go <u>slower</u> than big-oh

$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

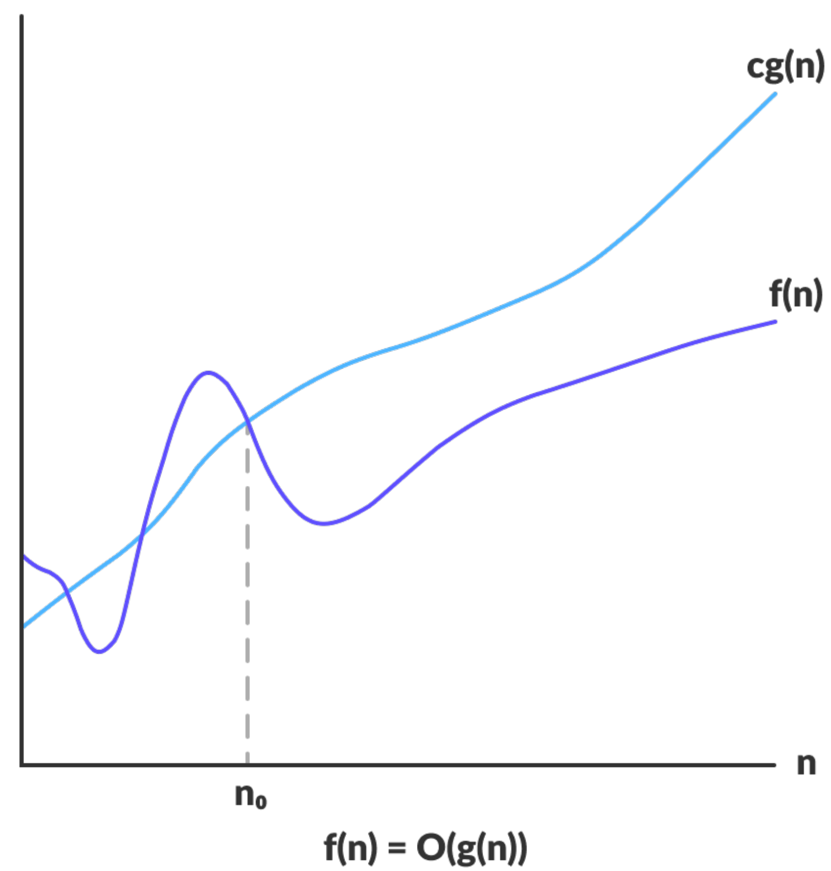


Source: https://www.programiz.com/dsa/asymptotic-notations

Big-oh notation

- *Big-oh* (or only *oh*) notation defines an upper bound
  - It says that an algorithm cannot go <u>slower</u> than big-oh

$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$



$$\Theta(g(n)) \subseteq O(g(n))$$

Source: https://www.programiz.com/dsa/asymptotic-notations

Big-oh notation

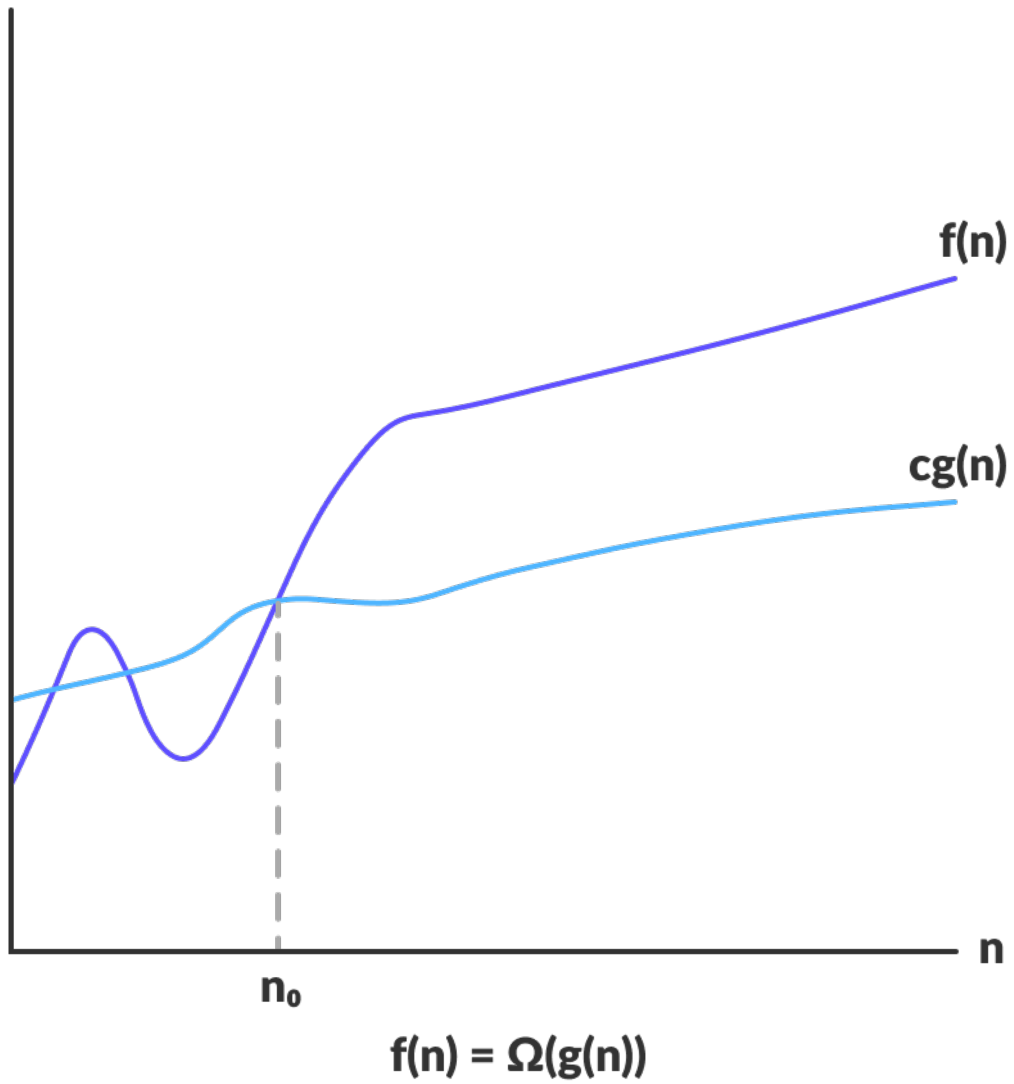
- *Big-oh* is used for the worst-case analysis because it defines a bound for <u>all input instances</u>

- Big-Theta does not imply an upper bound for every possible input
  - Indeed, $\Theta(n^2)$ for insertion sort is not a tight upper bound for all inputs since we have $\Theta(n)$ when the input instance is already sorted

Big-Omega notation

- It is complementary to big-Oh, it provides a lower bound

$$\Omega(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$



Source: https://www.programiz.com/dsa/asymptotic-notations