

Chapter 3 - Growth of functions [Cormen 2001]

1/

- Chapter 2 - defined the order of growth of an algorithm
 - simple characterization of an algorithm's efficiency
 - allows comparison of algorithmic performance
 - Exact running time (\neq order of growth):
 - usually not worth the effort of computing;
 - For large enough inputs:
the multiplicative constants and lower-order terms are dominated by the effects of the input size itself;
- We are interested in the asymptotic behaviour efficiency:
~~when~~ $n \rightarrow \infty$
- Let's see then the asymptotic notation

3.1 Asymptotic Notation

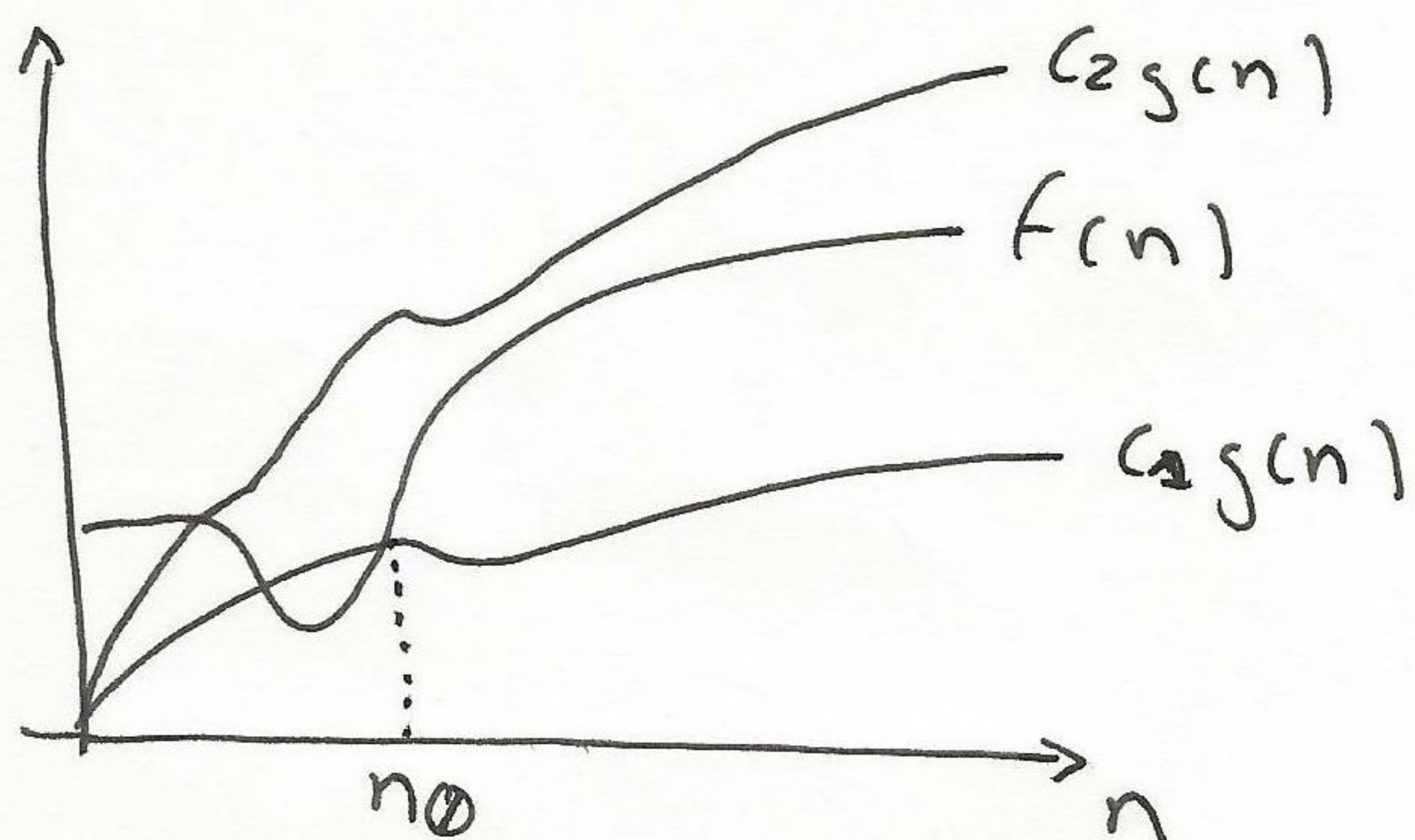
- functions with domain $\in \mathbb{N} = \{0, 1, 2, \dots\}$

3.1.1 Θ -notation

- Set of functions

- $\Theta(g(n)) = \{f(n) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0$
 $\exists c_1, c_2 > 0\}$

- "Sandwich" of functions



$g(n)$ is an asymptotically tight bound for $f(n)$

All the functions involved in Θ must be non-negative for sufficiently large n . Why? By definition ;)

- Intuition behind throwing away lower-order terms and ignoring the leading coefficient of the highest-order term:

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

$$c_1 g(n) \leq \frac{1}{2}n^2 - 3n \leq c_2 g(n)$$

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

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

• There are a lot of possibilities for the choice of c_1 and c_2

- Assume $c_1 = \frac{1}{14}$ then $n = 7$ for the left inequality to hold

- Assume $c_2 = \frac{1}{2}$ then $n = 1$ for the right inequality to hold

- Don't forget that $n \in \mathbb{N} = \{0, 1, \dots\}$

• By choosing $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$, $n_0 = 7$ then $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Example:

- $6n^3 \neq \Theta(n^2)$

- To prove the contradiction:

$$c_1 n^2 \leq 6n^3 \leq c_2 n^2$$

$$c_2 n^2 \geq 6n^3 \quad (\text{lets focus on the right hand})$$

$$c_2 \geq 6n \Rightarrow n \leq c_2/6$$

This cannot hold when $n \rightarrow \infty$ since c_2 is a constant

Intuitively: lower-order terms of an asymptotically positive function can be ignored because they are insignificant for large n .

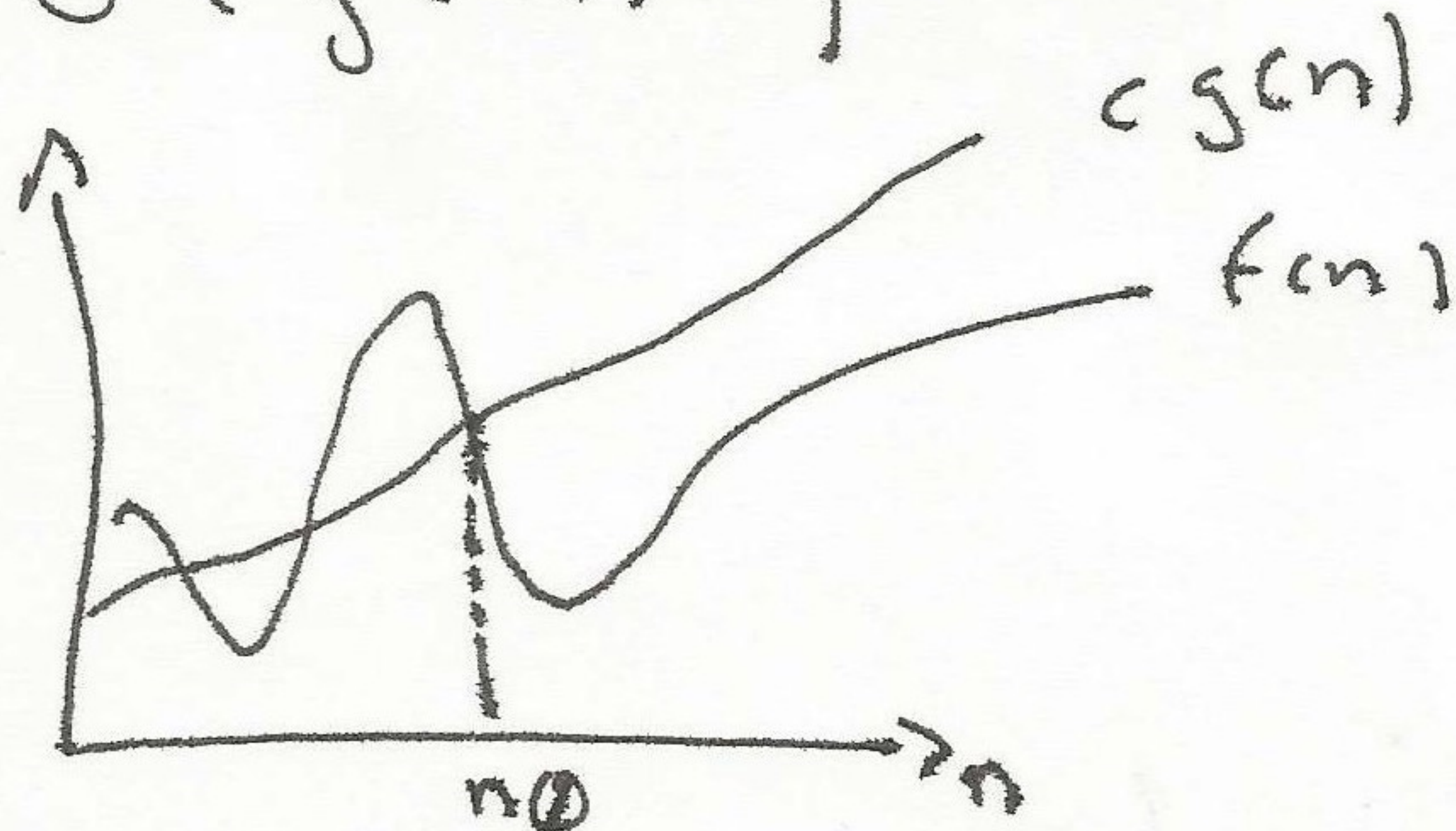
3.1.2

O -notation (big-oh)

- Asymptotic upper bound

- set of functions

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



- Note that $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(n)$, i.e.

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

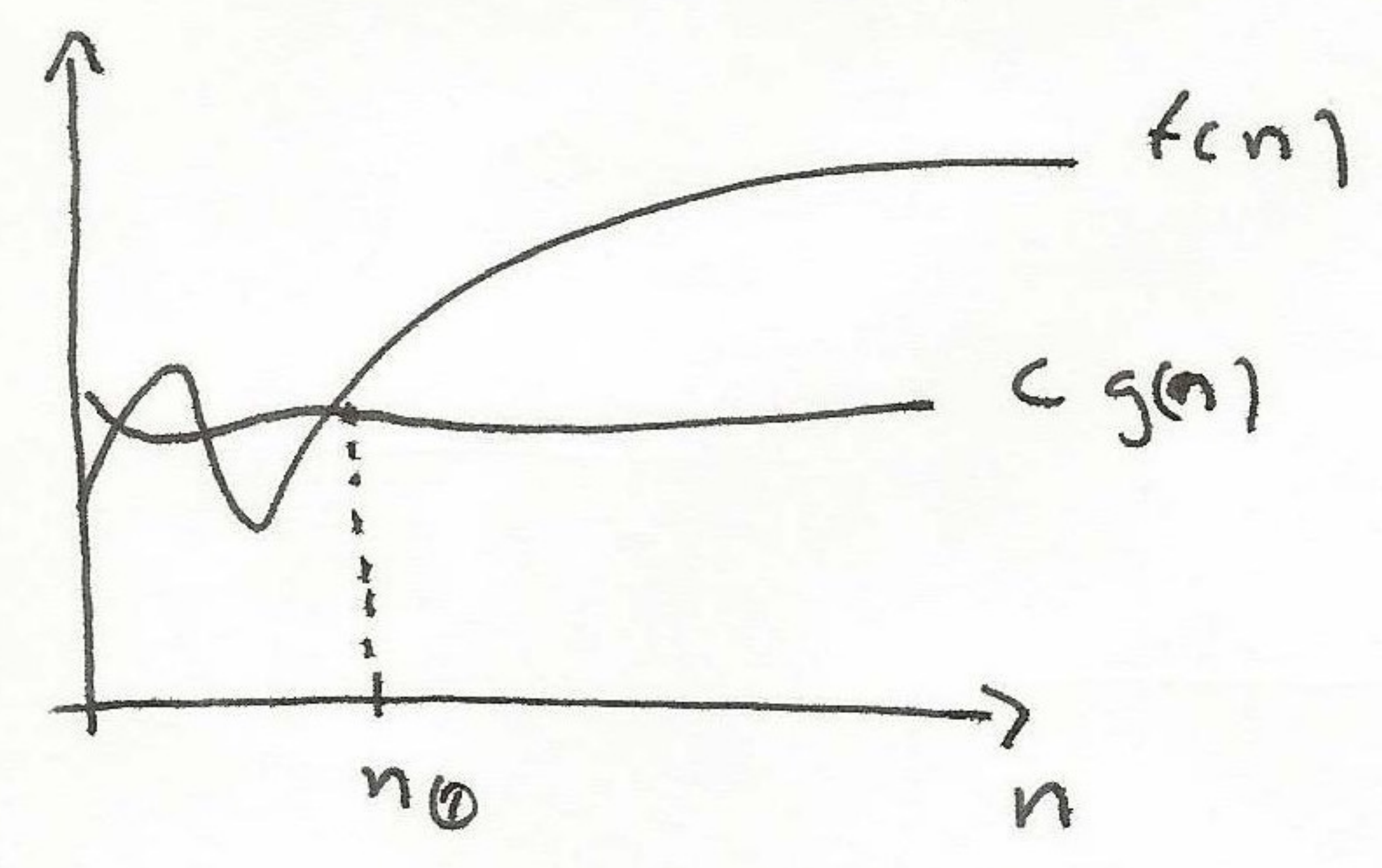
- Using O -notation makes it easy to describe the running time of an algorithm merely by looking at its structure, example:

for (i = 0; i < n; i++) \Rightarrow max n iterations
 for (j = 0; j < n; j++) \Rightarrow max n iterations
 \vdots
 for (z = 0; z < n; z++) \Rightarrow max n iterations
 \Rightarrow Worst running time: $\underbrace{n \times n \times \dots \times n}_{\text{number of for loops}}$

3.1.3 Ω -notation

- Asymptotic lower bound
- Set of functions

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



- Insertion Sort $\begin{cases} \Omega(n) & \text{best-case scenario where everything is sorted} \\ O(n^2) & \text{worst-case scenario where the elements of the array are reversed} \end{cases}$

Question:

- Do you know any other notations besides these ones?
- Do you see any way to extend the previous notations?

3.1.4 o-notation (little-oh)

51

- O -notation may or may not be asymptotically tight
 - $2n^2 = O(n^2)$ is asymptotically tight
 - $2n = O(n^2)$ is not asymptotically tight
 - Although true, a better bound would be $O(n)$
- We use o -notation to denote an upper bound that is not asymptotically tight
- Set of functions
- $o(g(n)) = \{ f(n) : 0 \leq f(n) < c g(n) \quad \forall n \geq n_0, n_0 > 0, c > 0 \}$

Example:

$$2n = o(n^2)$$

$$2n^2 \neq o(n^2)$$

- The definitions of O -notation and o -notation are similar:

Main difference:

$$f(n) = O(g(n)), \quad 0 \leq f(n) \leq c g(n) \text{ for some constant } \underline{c > 0}$$

$$f(n) = o(g(n)), \quad 0 \leq f(n) < c g(n) \text{ for all constants } \underline{c > 0}$$

Example:

$2n^2 \neq o(n^2)$ i.e. is not asymptotically tight because
for $c=2$ we would have $c g(n) = c n^2 = 2n^2$
that would be equal to the left hand side

6/

Intuitively, in o -notation the function $f(n)$ becomes insignificant relative to $g(n)$ as $n \rightarrow \infty$, i.e.:

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

3.1.5 ω -notation (little-omega)

- ω -notation is to Ω -notation what o -notation is to Θ -notation
- ω -notation: lower bound that is not asymptotically tight.

- Set of functions

$$\omega(g(n)) = \left\{ f(n) : 0 \leq c g(n) < f(n), \forall n > n_0, \forall c > 0 \right\}$$

- Example: $\frac{n^2}{2} = \omega(n)$, i.e. not asymptotically tight
 $\frac{n^2}{2} \neq \omega(n)$

- Implies that $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ if the limit exists

That is $f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity

3.1.6 Relational properties

Transitivity:

$$\begin{aligned} f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) &\Rightarrow f(n) = \Theta(h(n)) \\ f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) &\Rightarrow f(n) = O(h(n)) \\ f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) &\Rightarrow f(n) = \Omega(h(n)) \\ f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) &\Rightarrow f(n) = o(h(n)) \\ f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) &\Rightarrow f(n) = \omega(h(n)) \end{aligned}$$

↳

Reflexivity

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

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

implies

Symmetry:

if and only if

$$f(n) = \Theta(g(n)) \text{ iff } g(n) = \Theta(f(n))$$

Exercise 3.1.4

3.1.4.1

3.1.4.2

$$I_s \quad 2^{n+1} = O(2^n)?$$

$$I_s \quad 2^{2n} = O(2^n)?$$

[resolution 3.1.4.1]

$\rightarrow 2^{n+1}$ is $O(2^n)$, proof:

To show that $2^{n+1} = O(2^n)$ we must find constants $c, n_0 > 0$ such that

$$0 \leq 2^{n+1} \leq c \cdot 2^n \quad \forall n \geq n_0$$

Since $2^{n+1} = 2 \cdot 2^n \quad \forall n$ we can satisfy the definition with $c = 2$ and $n_0 = 1$

[resolution 3.1.4.2]

$\rightarrow 2^{2n}$ is not $O(2^n)$, proof:

To show that $2^{2n} \neq O(2^n)$ assume that there exist constants $c, n_0 > 0$ such that

$$0 \leq 2^{2n} \leq c \cdot 2^n \quad \forall n \geq n_0$$

Then $2^{2n} = 2^n \cdot 2^n \leq c \cdot 2^n \Rightarrow 2^n \leq c \dots$ But no constant is greater than all 2^n and so the assumption leads to a contradiction.