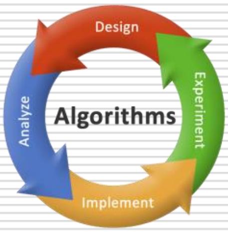
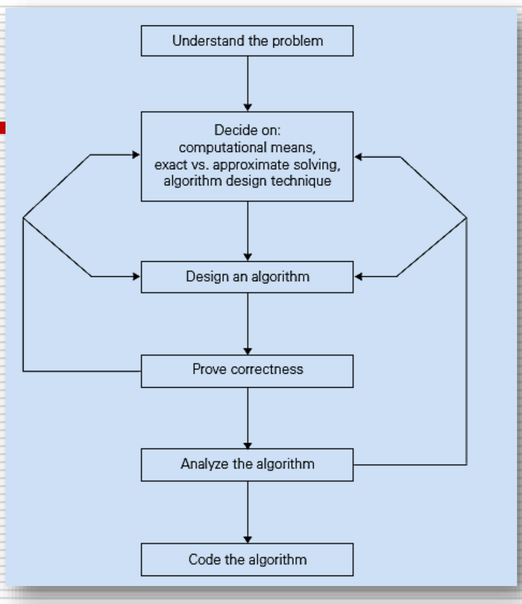
DR. Gatot F. Hertono, MSc.

# Design and Analysis of ALGORITHM (Week 3)




## Algorithm design and analysis process




*Source*: Introduction to the Design and Analysis of Algorithms, Anany Levitin

# Asymptotic Notation: Order of Growth

Background

Suppose, in worst case, a problem can be solved by using two different algorithms, with time complexity:

Algorithm A:

$f(n) = 400n + 23$

Algorithm B:

$g(n) = 2n^2 - 1$

Which one is better?

| n | $3n^2$ | 14n+17 |
|---|---|---|
| 1 | 3 | 31 |
| 10 | 300 | 157 |
| 100 | 30,000 | 1,417 |
| 1000 | 3,000,000 | 14,017 |
| 10000 | 300,000,000 | 140,017 |

$3n^2 > 14n+17$

∀ "large enough" n

**Solution:** Ignore the constants & low-order terms.

→ we use Asymptotic Notation ($\Omega$, $\Theta$ dan $O$)

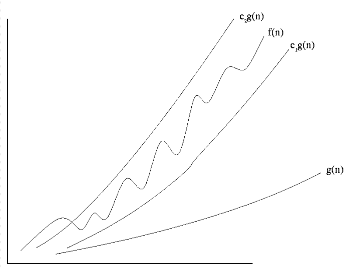
# Upper and Lower Bounds

→ Because it is so easy to cheat with the best case running time, we usually don't rely too much about it.

→ Because it is usually very hard to compute the average running time, since we must somehow average over all the instances, we usually strive to analyze the worst case running time.

→ The worst case is usually fairly easy to analyze and often close to the average or real running time.

We have agreed that the best, worst, and average case complexity of an algorithm is a numerical function of the size of the instances.




However, it is difficult to work with exactly because it is typically very complicated!

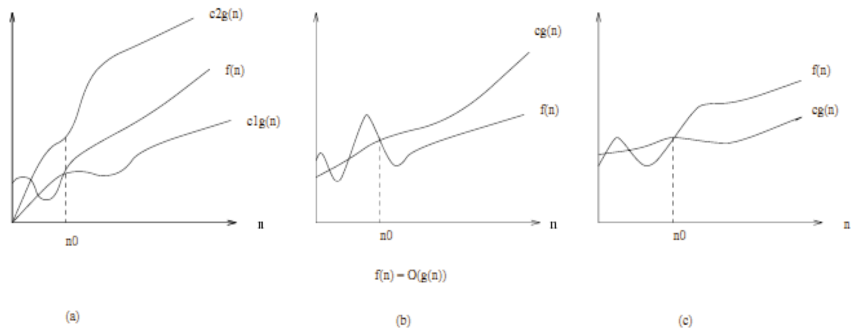
Thus it is usually cleaner and easier to talk about *upper and lower bounds* of the function.

This is where the dreaded big O notation comes in!

# Ω, Θ and *O* notations

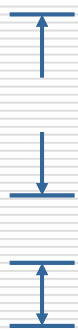
What does it mean?

## *O*, Ω, and Θ

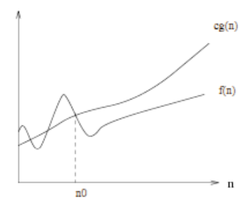



# Names of Bounding Functions

- $f(n) = O(g(n))$ means $C$ x $g(n)$ is an ***upper bound*** on $f(n)$.
- $f(n) = \Omega(g(n))$ means $C$ x $g(n)$ is a ***lower bound*** on $f(n)$.
- $f(n) = \Theta(g(n))$ means $C_1$ x $g(n)$ is an ***upper bound*** on $f(n)$ and $C_2$ x $g(n)$ is a ***lower bound*** on $f(n)$.



# $O$ (big *Oh*) Notation




$f(n) = O(g(n))$: $g(n)$ is an *asymptotically upper bound* for $f(n)$. *i.e. f* does not grow faster than *g*.

Formal definition:

$$O(g(n)) = \{f(n) : \exists\ c > 0 \text{ and } n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

**Remark:** "$f(n) = O(g(n))$" means that

$$f(n) \in O(g(n)).$$

**Examples:**

(1) $n^2 + 2n = O(n^2)$.
(2) $200n^2 - 100n = O(n^2)$.
(3) $n \log_2 n = O(n^2)$.
(4) $n^2 \log_2 n \neq O(n^2)$.
(5) $\forall a, b > 1$, $\log_a n = O(\log_b n)$.

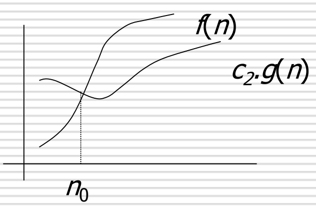
*Example*: $f(n) = n^3 + 20n^2 + 100.n$, then $f(n) = O(n^3)$.

*Proof*:

$\forall\ n \geq 0$, $n^3 + 20n^2 + 100.n \leq n^3 + 20n^3 + 100.n^3 = 121.n^3$.

Choose $c = 121$ and $n_0 = 0$, then it completes the definition.

# $\Omega$ (big Omega) Notation




$f(n) = \Omega(g(n))$: $g(n)$ is an *asymptotically lower bound* for $f(n)$.

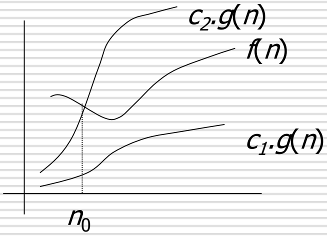
Formal definition:

$$\Omega(g(n)) = \{f(n) : \exists\ c > 0 \text{ and } n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$

**Examples:**

(1) $200n^2 - 100n = \Omega(n^2) = \Omega(n) = \Omega(1)$.
(2) $n^2 = \Omega(n)$.
(3) $n^2 \neq \Omega(n^2 \log n)$.
(3) Does $f(n) = O(g(n))$ imply $g(n) = \Omega(f(n))$?
(4) Does $f(n) = \Omega(g(n))$ imply $g(n) = O(f(n))$?

# $\Theta$ (Big Theta) Notation




$f(n) = \Theta(g(n))$: $g(n)$ is an *asymptotically tight bound* for $f(n)$.

Formal definition:

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

**Examples:**

(1) $5n^2 - 2n + 5 = \Theta(n^2)$

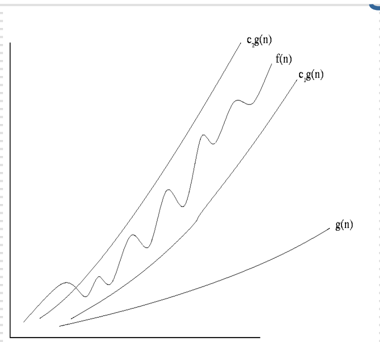
(2) $5n^2 - 2n + 5 = \Theta(n^2 + \log n)$

# More on Big-Theta $\Theta$

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

$$\exists c_1, c_2, \exists n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

**Constants c1 and c2 must be positive!**




- For some sufficiently small c1 (= 0.0001)
  For some sufficiently large c2 (= 1000)
- For all sufficiently large n
- For some definition of "sufficiently large"

f(n) is sandwiched between $c_1g(n)$ and $c_2g(n)$

# Examples of Θ

$3n^2 + 7n + 8 = \Theta(n^2)$ ?

True

$$\exists c_1, c_2, \exists n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$c_1 = 3$, $c_2 = 4$, $n_0 = 8$; $n \geq 8$

$$3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2$$

---

$n^2 = \Theta(n^3)$ ?

$$\exists c_1, c_2, \exists n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$c_1 = 0$

$$0 \cdot n^3 \leq n^2 \leq c_2 \cdot n^3$$

False, since C1,C2 must be >0

# Properties

Note that if

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

then

$$f(n) = \Omega(g(n)) \quad \text{and} \quad f(n) = O(g(n)).$$

In the other direction, if

$$f(n) = \Omega(g(n)) \quad \text{and} \quad f(n) = O(g(n)),$$

then

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

**Transitivity.**
- If f = O(g) and g = O(h) then f = O(h).
- If f = Ω(g) and g = Ω(h) then f = Ω(h).
- If f = Θ(g) and g = Θ(h) then f = Θ(h).

**Additivity.**
- If f = O(h) and g = O(h) then f + g = O(h).
- If f = Ω(h) and g = Ω(h) then f + g = Ω(h).
- If f = Θ(h) and g = O(h) then f + g = Θ(h).

*Assignment* 2a: Based on the definition of Ω, Θ and *O*, prove that

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

# Notes

Note that

$$\log(n^{473} + n^2 + n + 96) = O(\log n)$$

since $n^{473} + n^2 + n + 96 = O(n^{473})$, and $\log n^{473} = 473 * \log n$.

*Any* exponential dominates *every* polynomial. This is why we will seek to avoid exponential time algorithms.

Assignment 2b: *(a) Is* $2n + 1 = O(2n)$?

*(b) Is* $2^{2n} = O(2^n)$?

# Examples

$$\begin{aligned}
3n^2 - 100n + 6 &= O(n^2) \text{ because } 3n^2 > 3n^2 - 100n + 6 \\
3n^2 - 100n + 6 &= O(n^3) \text{ because } .01n^3 > 3n^2 - 100n + 6 \\
3n^2 - 100n + 6 &\neq O(n) \text{ because } c \cdot n < 3n^2 \text{ when } n > c
\end{aligned}$$

$$\begin{aligned}
3n^2 - 100n + 6 &= \Omega(n^2) \text{ because } 2.99n^2 < 3n^2 - 100n + 6 \\
3n^2 - 100n + 6 &\neq \Omega(n^3) \text{ because } 3n^2 - 100n + 6 < n^3 \\
3n^2 - 100n + 6 &= \Omega(n) \text{ because } 10^{10^{10}} n < 3n^2 - 100 + 6
\end{aligned}$$

$$\begin{aligned}
3n^2 - 100n + 6 &= \Theta(n^2) \text{ because } O \text{ and } \Omega \\
3n^2 - 100n + 6 &\neq \Theta(n^3) \text{ because } O \text{ only} \\
3n^2 - 100n + 6 &\neq \Theta(n) \text{ because } \Omega \text{ only}
\end{aligned}$$

Think of the equality as meaning *in the set of functions.*

## What does asymptotic property imply for an algorithm?

$t_1(n) \in O(g_1(n))$

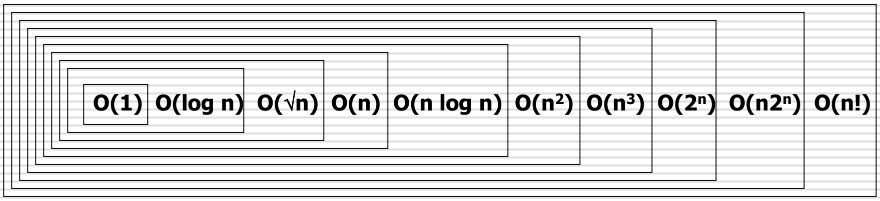
$t_2(n) \in O(g_2(n))$

How is the overall efficiency of this algorithm?

# Other Notations

| | | |
|---|---|---|
| Theta | f(n) = θ(g(n)) | f(n) ≈ c g(n) |
| BigOh | f(n) = O(g(n)) | f(n) ≤ c g(n) |
| Omega | f(n) = Ω(g(n)) | f(n) ≥ c g(n) |
| Little Oh | f(n) = o(g(n)) | f(n) << c g(n) |
| Little Omega | f(n) = ω(g(n)) | f(n) >> c g(n) |

# Order Hierarchy




*Note* : not all order are comparable

Two functions, $f(n)$ and $g(n)$, are not *comparable* if:

$$f(n) \notin O(g(n)) \text{ and } g(n) \notin O(f(n))$$

*Example*: $f(n) = n^3$ and $g(n) = n^4.(n \bmod 2) + n^2$

**Lesson Learn**

- *Algorithm Design*, as taught in this class, is mainly about designing algorithms that have small big $O()$ running times.

- Being able to do good algorithm design lets you identify the *hard parts* of your problem and deal with them effectively.

- "All other things being equal", $O(n \log n)$ algorithms will run more quickly than $O(n^2)$ ones and $O(n)$ algorithms will beat $O(n \log n)$ ones.

- Too often, programmers try to solve problems using brute force techniques and end up with slow complicated code! A few hours of abstract thought devoted to algorithm design could have speeded up the solution substantially *and* simplified it.