

CS203 (H): Data Structures & Algorithm Analysis (DSAA)

Lecture #2

➤ Runtime and Asymptotic Notation

Prof. Pietro Oliveto

Department of Computer Science and Engineering

Southern University of Science and Technology (SUSTech)

`olivetop@sustech.edu.cn`

<https://faculty.sustech.edu.cn/olivetop>

Office: Room 113

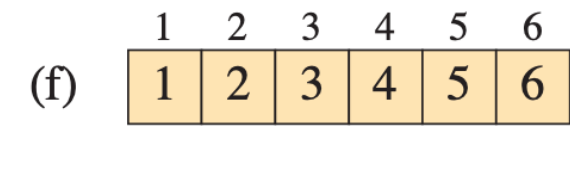
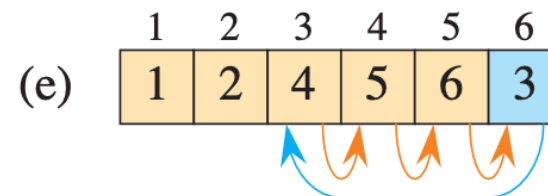
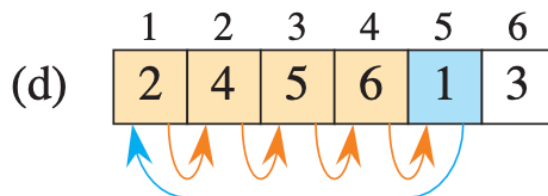
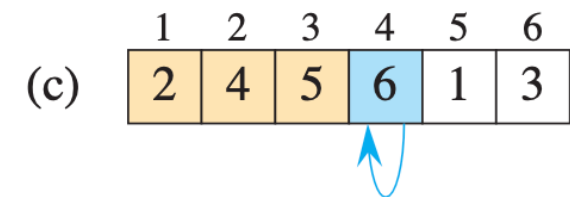
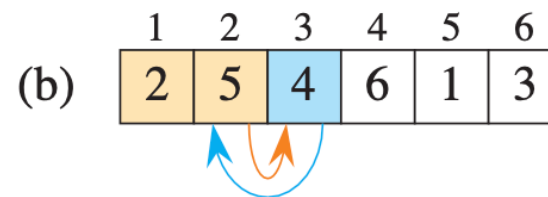
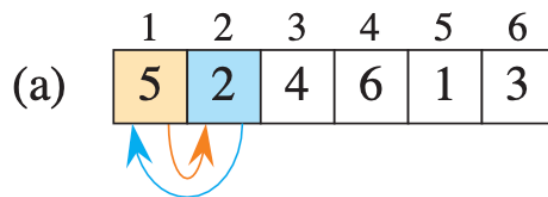
Reading: Section 3.1

(and flick through the treasure trove of formulas in Section 3.2, they might come in handy)

➤ Aims of this lecture

- To recap and simplify the runtime analysis of InsertionSort.
- To talk about growth of runtime with problem size.
- To introduce asymptotic notation (meet your Greek friends!)
- To show how to apply asymptotic notation

➤ Recap: Runtime of InsertionSort (1)



➤ Recap: Runtime of InsertionSort (2)

INSERTIONSORT(A)	Cost	Times
1: for $j = 2$ to $A.length$ do	c_1	n
2: $key = A[j]$	c_2	$n - 1$
3: // Insert $A[j]$ into ...		
4: $i = j - 1$	c_4	$n - 1$
5: while $i > 0$ and $A[i] > key$ do	c_5	$t_2 + t_3 + \dots = \sum_{j=2}^n t_j$
6: $A[i + 1] = A[i]$	c_6	$(t_2 - 1) + (t_3 - 1) + \dots = \sum_{j=2}^n (t_j - 1)$
7: $i = i - 1$	c_7	$(t_2 - 1) + (t_3 - 1) + \dots = \sum_{j=2}^n (t_j - 1)$
8: $A[i + 1] = key$	c_8	$n - 1$

Define t_j as the number of times the while loop is executed for that j .

➤ Recap: Runtime of InsertionSort (3)

- General formula:

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$$

- **Best case** simplifies to $T(n) = an + b$

for constants $a > 0, b$ composed of c_1, c_2 , etc.

- A **linear** function in n .

- **Worst case** simplifies to $T(n) = an^2 + bn + c$

for constants $a > 0, b, c$ composed of c_1, c_2 , etc.

- A **quadratic** function in n .

➤ On best case and worst case

- The running time of every instance is sandwiched between the best case and the worst case running time.
- ? Best case vs. worst case – which is more important?
Something about all possibilities
↓
Boundary
- Average case: performance on “average” input.
 - For sorting: assume each permutation is equally likely
 - For other problems it's not always clear what an average input is
- Why worst case is important:
 - Guarantee that the algorithm will never take longer
 - For some algorithms, the worst case is quite frequent
 - Often (not always) the average case is as bad as the worst case

➤ Comparison of two runtimes

- Let's compare two algorithms:
 - Algorithm A has runtime $2n^2$
 - Algorithm B has runtime $50n \log n$

Which one would you prefer?

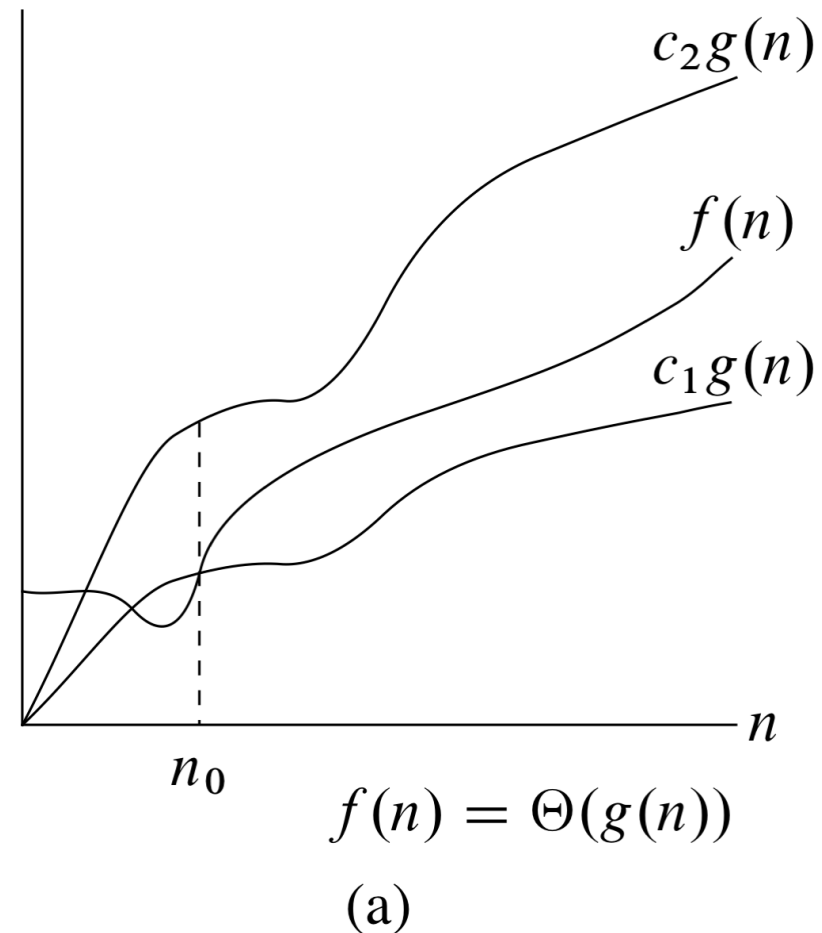
[Using Wolfram Alpha](#)

➤ Observations

- The biggest-order term (n^2 vs. $n \log n$) dominates the runtime as n grows.
- How the runtime scales with n is more important than constant factors (for large n).
- Additive smaller order terms (e.g. “ $+10n$ ” in “ $2n^2 + 10n$ ”) become **irrelevant** for large n .
- Care about large n , small problems (small n) are easy anyway.
- Recommendations:
 - If your problem is **always very small**, use the simplest algorithm.
 - Otherwise, use most **efficient** algorithm (**best growth** in n)

➤ Asymptotic Notation: Θ

- Idea: capture **asymptotic growth**
- Ignore constant factors
- Ignore small-order terms
- Ignore “blips” for tiny n
- Intuition: “ Θ ” **captures fastest growing term**
e.g. $2n^2 + 3n = \Theta(n^2)$.
- More details in the book, Section 3.1.



➤ Definition of $\Theta(g(n))$

For a given (non-negative) function $g(n)$ we denote by $\Theta(g(n))$ the set of functions

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

A function $f(n)$ belongs to the set $\Theta(g(n))$ if it can be “sandwiched” between $c_1 g(n)$ and $c_2 g(n)$, for sufficiently large n .

We could write: $f(n) \in \Theta(g(n))$.

However, the common notation is: $f(n) = \Theta(g(n))$, the equality being read from left to right!

We say that $g(n)$ is an asymptotically tight bound for $f(n)$.

➤ Example for Θ notation

- Example: $\frac{3}{2}n^2 + \frac{7}{2}n - 4 = \Theta(n^2)$.

To show this, we need to find constants c_1, c_2, n_0 such that for all $n \geq n_0$

$$0 \leq c_1 n^2 \leq \frac{3}{2}n^2 + \frac{7}{2}n - 4 \leq c_2 n^2$$

- Let's divide by n^2 :

$$0 \leq c_1 \leq \frac{3}{2} + \frac{7}{2n} - \frac{4}{n^2} \leq c_2$$

- This is true, e.g., for $c_1 = \frac{3}{2}, c_2 = 2, n_0 = 7$.

(Other choices are possible so long as the inequalities hold.)

➤ Examples (1)

Task: find constants $c_1, c_2, n_0 > 0$ from definition of Θ .

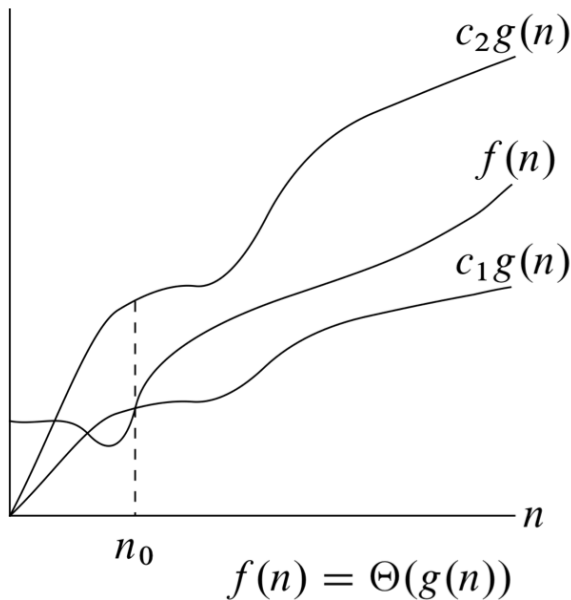
- $2n^2 = \Theta(n^2)$ since for all $n \geq n_0$
 $0 \leq c_1 n^2 \leq 2n^2 \leq c_2 n^2$
when choosing, say, $c_1 = 1, c_2 = 2, n_0 = 1$
- $2n^2 - 10n = \Theta(n^2)$ since for all $n \geq n_0$
 $0 \leq c_1 n^2 \leq 2n^2 - 10n \leq c_2 n^2$
when choosing, say, $c_1 = 1, c_2 = 2, n_0 = 10$
(as after division by n^2 we have $1 \leq 2 - 10/n \leq 2$ for $n \geq 10$)
- $50n \log n = \Theta(n \log n)$ since for all $n \geq n_0$
 $0 \leq c_1 n \log n \leq 50n \log n \leq c_2 n \log n$
when choosing, say, $c_1 = 50, c_2 = 50, n_0 = 1$

➤ Examples (2)

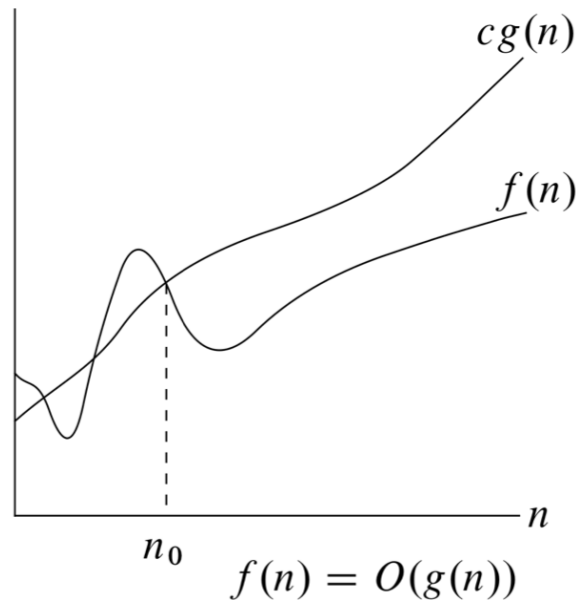
- but: $2n^2 \neq \Theta(n)$ since there is no constant c_2 such that $2n^2 \leq c_2 n$ **for all** $n \geq n_0$.
- and: $2n^2 \neq \Theta(n^3)$ since there is no constant c_1 such that $2n^2 \geq c_1 n^3$ **for all** $n \geq n_0$.

➤ Asymptotic Notation: Θ , O , Ω

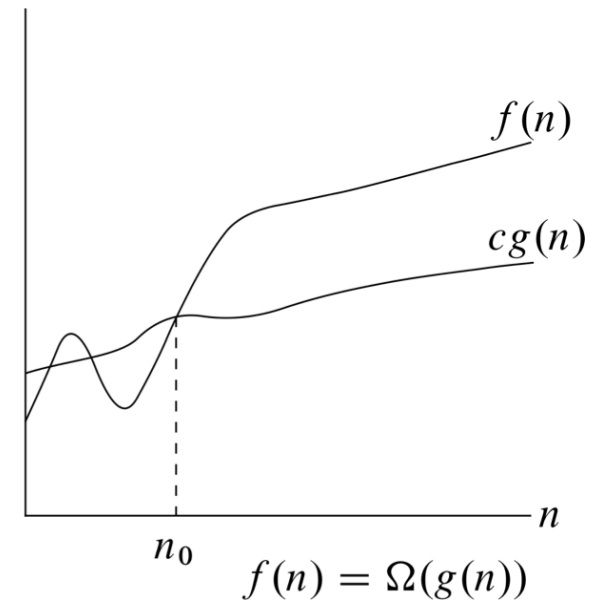
- Θ expresses tight upper and lower bounds on $f(n)$.
- Use O (“big-Oh”) if we only want to express an upper bound.
- Use Ω if we only want to express a lower bound.



(a)



(b)



(c)

➤ Definition of $O(g(n))$, $\Omega(g(n))$

For a given (non-negative) function $g(n)$ we denote by $O(g(n))$ and $\Omega(g(n))$ the following sets of functions:

$$O(g(n)) = \{f(n) : \text{there exist constants } 0 < c \text{ and } n_0 \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

$$\Omega(g(n)) = \{f(n) : \text{there exist constants } 0 < c \text{ and } n_0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

O and Ω are weaker than Θ . Together, they give Θ :

For any $f(n)$ and $g(n)$ we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

➤ Faster and slower growth

- Little-Oh “o” and little omega “ ω ” indicate strictly slower and faster growth, respectively:

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

$$f(n) = \omega(g(n)) \text{ if } g(n) = o(f(n))$$

➤ Asymptotic Notation: Overview

Notation	Meaning	Analogy
$f(n) = O(g(n))$	f grows at most as fast as g	“ $f \leq g$ ”
$f(n) = \Omega(g(n))$	f grows at least as fast as g	“ $f \geq g$ ”
$f(n) = \Theta(g(n))$	f grows as fast as g	“ $f = g$ ”
$f(n) = o(g(n))$	f grows slower than g	“ $f < g$ ”
$f(n) = \omega(g(n))$	f grows faster than g	“ $f > g$ ”

- Equalities are to be **read from left to right** – think of $f(n) = O(g(n))$ as actually meaning $f(n) \in O(g(n))$
- So $n = O(n^2)$ is **true** but $O(n^2) = n$ is **false**!
- We can chain equalities, e. g. $n = O(n) = O(n^2)$

➤ Common runtimes

$\Theta(1)$	constant time
$\Theta(\log n)$	logarithmic time
$\Theta(n)$	linear time
$\Theta(n^2)$	quadratic time
$\Theta(n^3)$	cubic time
n^k for $k = \Theta(1)$	polynomial time
2^n	exponential time

- Every polynomial of $\log n$ grows strictly slower than every polynomial of n , e. g. $(\log n)^{100} = o(n^{0.01})$
- Every polynomial of n grows strictly slower than every exponential function 2^{n^ε} , e. g. $n^{100} = o(2^{n^{0.01}})$

➤ Examples

Examples of using the various symbols:

- $2n + 1 = O(n)$
- $42 = O(n)$ (but not $\Theta(n)$!)
- $n - 9 = \Omega(n)$
- $n^2 + n = \Omega(n)$ (but neither $O(n)$, nor $\Theta(n)$!)
- $n^3 = o(n^4) = o(2^n)$
- $\sqrt{n} = \omega(\log n)$

➤ How to read asymptotic notation

How to read „The runtime of Algorithm XYZ is $O(n^2)$ “?

“The runtime of Algorithm XYZ is some (anonymous) function that grows at most as fast as n^2 .“

Or, more briefly,

“The runtime of Algorithm XYZ grows at most as fast as n^2 .“

Think of asymptotic notation as a **placeholder** for some anonymous function from the specified class.

- „runtime is $\Theta(n^2)$ “ \rightarrow „runtime grows as fast as n^2 “
- „runtime is $\Omega(n^2)$ “ \rightarrow „runtime grows at least as fast as n^2 “
- „runtime is $o(n^2)$ “ \rightarrow „runtime grows slower than n^2 “
- „runtime is $\omega(n^2)$ “ \rightarrow „runtime grows faster than n^2 “

➤ Asymptotic runtime of InsertionSort

- The runtime of InsertionSort is ...

$$\Omega(n) \quad \text{and} \quad O(n^2)$$

(grows at least as fast as n and at most as fast as n^2)

- This is because:
 - The best-case runtime is $\Theta(n)$
 - The worst-case runtime is $\Theta(n^2)$
 - So for every input, the runtime is at least $\Omega(n)$ and at most $O(n^2)$

➤ How to find c_1, c_2, n_0

- It is often helpful (though not compulsory) to divide by $g(n)$, e.g.

$$c_1 n \leq 10n + 5 \leq c_2 n \quad \Leftrightarrow \quad c_1 \leq 10 + \frac{5}{n} \leq c_2$$

Then try c_1, c_2 **sandwiching the constant term**, e.g. $c_1 = 10, c_2 = 15$.

- Remember that **$c_1 > 0$** : to show that $1 - \frac{3}{n} = \Omega(1)$ we cannot use $n_0 = 3$ as then there is no suitable $c_1 > 0$!

However, say, $n_0 = 6$ and $c_1 = \frac{1}{2}$ works as $1/2 \leq 1 - \frac{3}{n}$ for all $n \geq 6$.

- Also remember that inequalities need to hold **for all $n \geq n_0$** .

For instance, to show $1 - \frac{3}{n} = O(1)$ we cannot use $c_2 = \frac{1}{2}$ as

$1 - \frac{3}{n} \leq \frac{1}{2}$ is false for $n > 6$! Need to choose $c_2 \geq 1$ (e.g. $c_2 = 1$).

- No need to invest time to find the best possible constants.

➤ Rules to make runtime analysis simple

- For two non-negative functions $f(n)$, $g(n)$:

1. Slower functions can be ignored:

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

2. Asymptotic times can be multiplied:

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

Foo
1: foo
2: foo
3: for $i = 1$ to n do
4: foo
5: foo
6: foo

Example of how to use this:

- First two lines take time $\Theta(1)$
- One iteration of the for loop takes time $\Theta(1)$
- The for loop is executed $\Theta(n)$ times
- Total time is:

$$\Theta(1) + \Theta(n) \cdot \Theta(1) = \Theta(n).$$

➤ Asymptotic Notation: Comparing Sets

- Is $2n^2 + \Theta(n) = \Theta(n^2)$ true or false?
(Think of $\Theta(n)$ as a placeholder for an anonymous function from the set $\Theta(n)$ of all functions that grow linearly in n .)
- Such a statement is true if **no matter how the anonymous functions are chosen on the left of the equal sign, there is a way to choose the anonymous functions on the right of the equal sign** to make the equation valid.
- Example: is $O(n) = O(n^2)$?
True, because $O(n) \subseteq O(n^2)$
- Example: is $O(n^2) = O(n)$?
False, for example n^2 is in $O(n^2)$ but not in $O(n)$!

➤ Summary

- We may consider best-case, average-case, and worst-case runtime. Often the focus is on **worst-case runtime**.
- The most important aspect of efficiency is **scalability**: how the runtime grows with the input size, n .
 - Asymptotic perspective: $n \geq n_0$ (smaller problems are easy)
 - Scalability is more important than constant factors
 - Small-order terms become more insignificant as n grows.
- **Asymptotic notation** ($O, \Omega, \Theta, o, \omega$) hides constant factors and small-order terms, revealing asymptotic runtimes.
- Asymptotic notation refers to **sets of functions**, but for convenience is written with equalities read from **left to right**.