Algorithm efficiency and asymptotic analysis: Big-Oh, Big-Omega, Big-Theta

1. Introduction to asymptotic analysis

Suppose algorithm *X* exists and *X* has input data whose size is *n*.

Q: What is the goal of the asymptotic analysis of *X* produce?

A: To describe how the running time of *X* depends on *n*.

Remark: ’running time’ is traditional and misleading since

• asymptotic analysis used mostly for analyzing pseudocode

• pseudocode cannot be ’run’ executed

• no ’times’ can be measured from executing pseudocode

We will still use the traditional expression of 'running time'.

Q: Why ’asymptotic’?

A: We are interested in the behaviour of *X* as *n* becomes ’large’.

Let running time function $f(n)$ be count of simple operations needed when *X* computes from start to finish.

More precise first question:

Q: What is the goal of the asymptotic analysis of *X* produce?

A: To describe how $f(n)$ depends on *n* when *n* gets 'large'.

Example

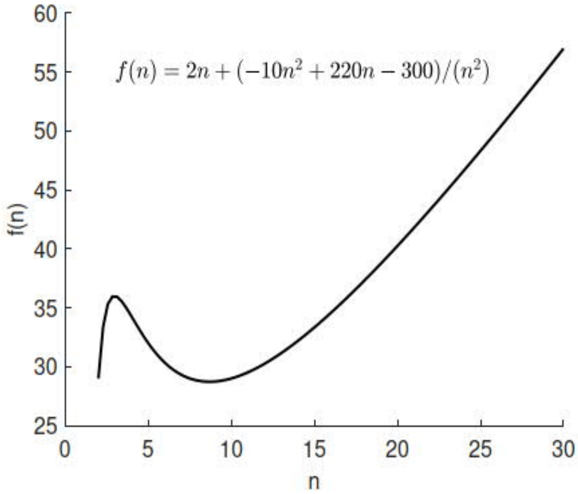
$$f(n) = 2n + \frac{-10n^2 + 220n - 300}{n^2}, \quad n > 0$$

When is *n* large?

For n large

$$\frac{-10n^2 + 220n - 300}{n^2} \longrightarrow -10$$

□



Asymptotic: we can ignore all terms in *f*(*n*) except the fastest growing

Example

function *f*(*n*)	fastest growing term	function of interest for asymptotic analysis
$f(n) = 2n + 100$	$2n$	$\hat{f}(n) = 2n$
$f(n) = n^2/100 + 100n$	$n^2/100$	$\hat{f}(n) = n^2/100$
$f(n) = n/10^3 + 200\log_2(n)$	$n/10^3$	$\hat{f}(n) = n/10^3$
$f(n) = n^2 + 2^n$	2^n	$\hat{f}(n) = 2^n$

□

For some algorithm we might have

$$f(n) = f_1(n) + f_2(n)$$

- for small n, $f_1(n)$ dominates $f_2(n)$
- for n large enough, $f_2(n)$ always dominates $f_1(n)$

In asymptotic analysis: ignore f_1

Remarks on asymptotic analysis

- other names: run time analysis, time-complexity analysis, growth rate analysis
- for two alternative algorithms *X* and *Y*, if *fX*(*n*) grows slower than *fY*(*n*), then *X* is 'better'
- we want lower and upper bounds on *f*(*n*)

2. Big-Oh

$O(g(n))$ is a set that contains functions.

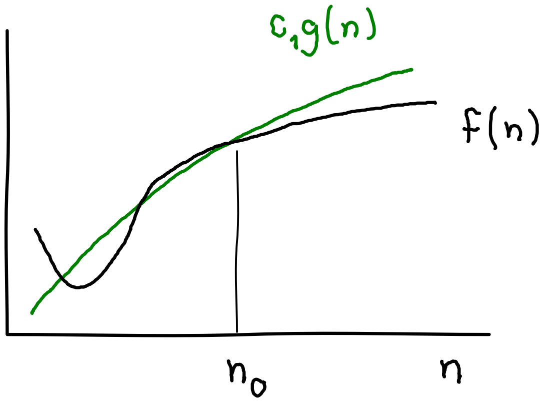
Definition

$$O(g(n)) = \{f(n) : \text{there exist constants } c_1 > 0, n_0 > 0 \text{ such that } f(n) \leq c_1 g(n), \text{for all } n > n_0\}$$

Function $f(n)$ belongs to $O(g(n))$, when there exists positive constants c_1 and n_0, such that $f(n) \leq c_1 g(n)$ for all $n > n_0$.

Plot of big-oh

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



Typical situation:

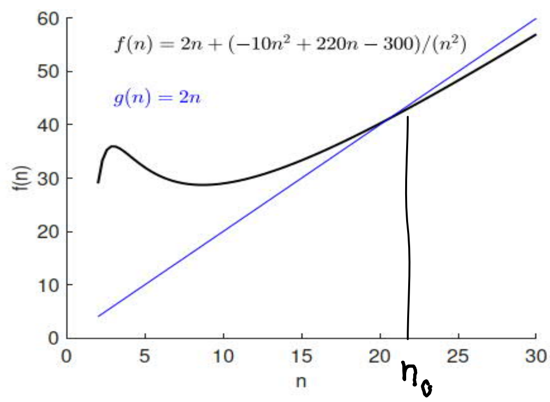
- we start $f(n)$ and we want to find $g(n)$ such that $f(n) \in O(g(n))$
- in principle we should find c_1 and n_0 to satisfy definition
- in practice we simply have to find fastest growing term in $\cancel{fn}$ $f(n)$

Example (contd)

$$f(n) = 2n + \frac{-10n^2 + 220n - 300}{n^2}\ , \quad n > 0$$

According to figure $f(n) \in O(n)$.

□



Example (contd)

function $f(n)$	fastest growing term	$O(g(n))$ for $f(n)$
$f(n) = 2n + 100$	$2n$	$f(n) \in O(n)$
$f(n) = n^2/100 + 100n$	$n^2/100$	$f(n) \in O(n^2)$
$f(n) = n/10^3 + 200\log_2(n)$	$n/10^3$	$f(n) \in O(n)$
$f(n) = n^2 + 2^n$	2^n	$f(n) \in O(2^n)$

$\square$

Example
Consider the following functions:

$f_1(n) = 100n + 10^6$ $\quad f_2(n) = n + 200\sqrt{n}$ $\quad f_3(n) = n^2 + 10^3 n$

$f_4(n) = n + n\log_2 n$ $\quad f_5(n) = n^{3/2} + 10^2 n + 10^7$ $\quad f_6(n) = n + 100\log_2 n$

$O(g(n))$	functions that belong to $O(g(n))$	functions that do not belong to $O(g(n))$
$O(n)$	f_1, f_2, f_6	f_3, f_4, f_5
$O(n\log_2 n)$	f_1, f_2, f_6, f_4	f_5, f_3
$O(n^2)$	$f_1, f_2, f_3, f_4, f_5, f_6$	—

$\square$

Q: In previous example

$$f_1 \in O(n) \text{ and } f_1 \in O(n \log_2 n) \text{ and } f_1 \in O(n^2)$$

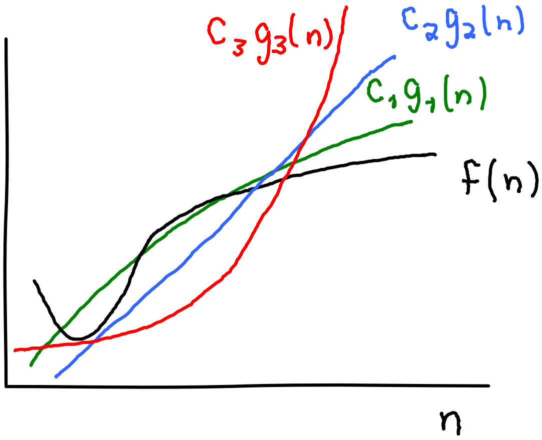
Are they all correct? Which is the best in other words provides the most information?

For n large enough

$c_1 g_1(n) < c_2 g_2(n)$

and

$c_1 g_1(n) < c_3 g_3(n)$



A: They are all correct. The most information is provided by

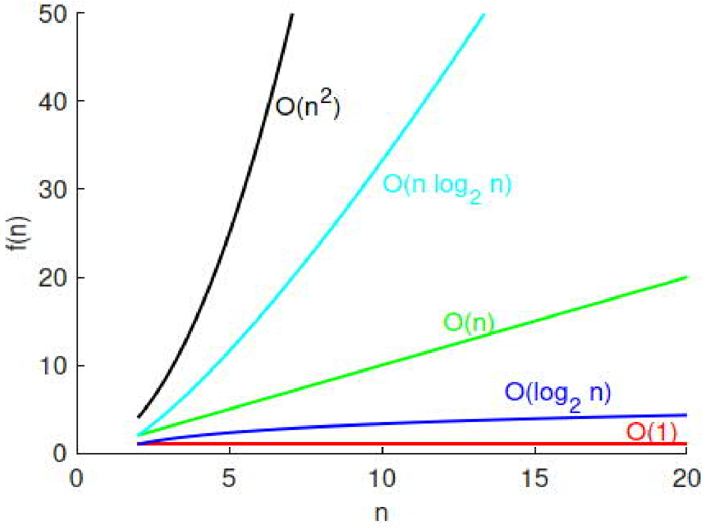
$$f_1 \in O(n)$$

$O()$ is an upper bound on growth and n grows slower than $n \log_2 n$ and n^2.

Principle: smallest upper bound conveys the most information

Typical *O*()-classes encountered in algorithm analysis

$O(g(n))$-class	name	if algorithm X's $f(n)$ belongs to $O(g(n))$
$O(1)$	constant time	Algorithm X runs in constant time.
$O(\log_2 n)$	logarithmic time	Algorithm X runs in log time.
$O(n)$	linear time	Algorithm X runs in linear time.
$O(n \log_2 n)$	linearithmic time	Algorithm X runs in linearithmic time.
$O(n^2)$	quadratic time	Algorithm X runs in quadratic time.



These $O()$-classes can be put in order:

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2)$$

fastest → → → slowest

Big-Oh is the most frequently quoted asymptotic result for different routines/procedures:

• C++ STL

https://alyssaq.github.io/stl-complexities/

https://github.com/gibsjose/cpp-cheat-sheet/blob/master/Data%20Structures%20and%20Algorithms.md

• Java Collections

https://gist.github.com/psayre23/c30a821239f4818b0709

• python

https://wiki.python.org/moin/TimeComplexity

3. Big-Omega and Big-Theta

Big-Omega

$\Omega(g(n))$ is a set that contains functions.

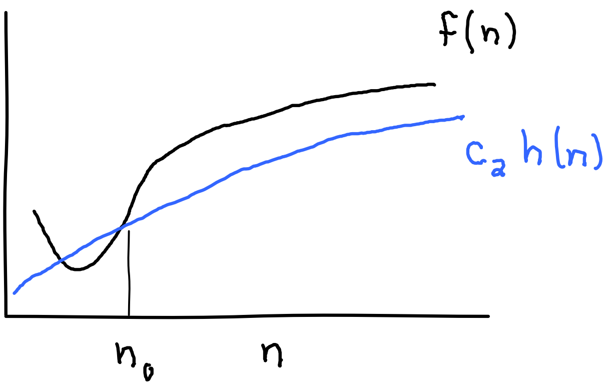
Definition

$\Omega(g(n)) = \{f(n) : \text{there exist constants } c_2 > 0, n_0 > 0 \text{ such that } f(n) \geq c_2 g(n), \text{for all } n > n_0\}$

Function $f(n)$ belongs to $\Omega(g(n))$, when there exists positive constants c_2 and n_0, such that $f(n) \geq c_2 g(n)$ for all $n > n_0$.

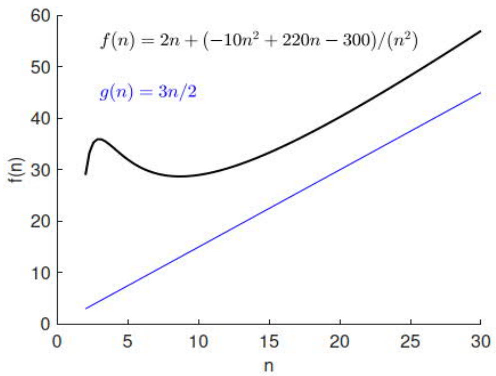
Plot of big-Omega significance.

$f(n) \in \Omega(h(n))$



Example (contd)

$$f(n) = 2n + \frac{-10n^2 + 220n - 300}{n^2}, \quad n > 0$$



According to figure $f(n) \in \Omega(n)$.

$\square$

Q: In the previous example we could have written

$$f(n) \in \Omega(n) \quad \text{or} \quad f(n) \in \Omega(\sqrt{n}) \quad \text{or} \quad f(n) \in \Omega(1)$$

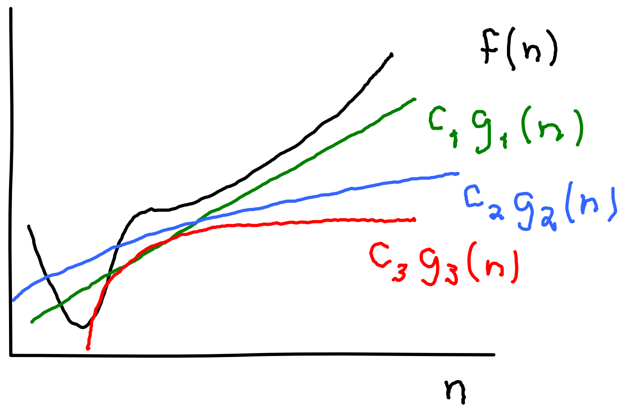
Are they all correct? Which is the best in other words provides the most information?

For n large enough

$c_1 g_1(n) > c_2 g_2(n)$

and

$c_1 g_1(n) > c_3 g_3(n)$



A: They are all correct. The most information is provided by

$$f(n) \in \Omega(n)$$

$\Omega()$ is a lower bound on growth and n grows faster than $\sqrt{n}$ and 1.

Principle: largest lower bound conveys the most information

Order of big-Omega classes:

$$\Omega(n^2) \subset \Omega(n \log n) \subset \Omega(n) \subset \Omega(\log n) \subset \Omega(1)$$

Big-Theta

$\Theta(g(n))$ is a set that contains functions.

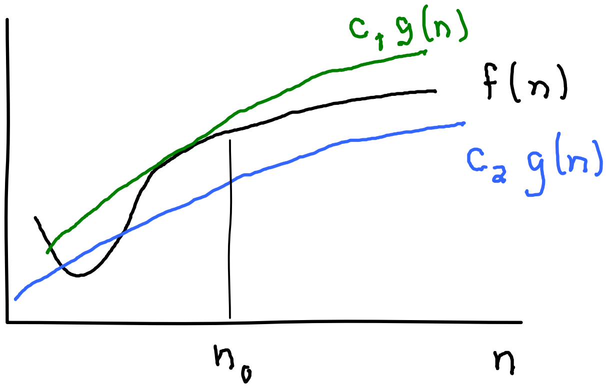
Definition

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

If $f(n) \in O(g(n))$ and $f(n) \in \Omega(h(n))$ and $g(n) = h(n)$, then $f(n) \in \Theta(g(n))$.

Plot of big-Theta significance.

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



Example (contd)

$$f(n) = 2n + \frac{-10n^2 + 220n - 300}{n^2}, \quad n > 0$$

From previous examples: $f(n) \in O(n)$ and $f(n) \in \Omega(n)$

Consequence: $f(n) \in \Theta(n)$ $\square$

Interpretations for big-oh, big-omega when *f*(*n*) is running time function of some algorithm.

- $f(n) \in O(g(n))$ means $f(n)$ will never grow faster than some multiple of $g(n)$ (worst growth)
- $f(n) \in \Omega(h(n))$ means $f(n)$ will never grow slower than some multiple of $h(n)$ (best growth)
- Often when doing asymptotic analysis for an algorithm we obtain the following result:

  $$f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(h(n)) \text{ and } g(n) \neq h(n)$$

  Consequence: there is no $\Theta()$ result.

4. Analysis of insertion sort

We will form two running time functions for INSERTSORT by counting simple operations.

Simple operations:

• arithmetic operations: +, -, *, /

• if-statement, else-statement

• one iteration of for or while or for-each

• variable assignment

• accessing a single item in memory

• a single call to a procedure (NOT the execution of the procedure itself)

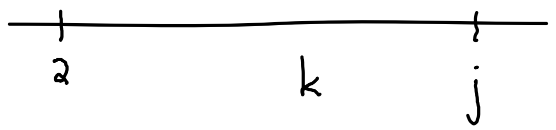
Assumption: each simple operation takes the same amount of time

$n = A.\text{length}$

In forming the counts we ignore nature of input data.

Pseudocode

```
1  INSERTSORT(A)
2  input: number array A output: sorted array A
3  /* The numbers in input A[1..n] may be in any order. On output the
4  numbers in A are sorted from smallest to largest. */
5  for j from 2 to A.length
6    key = A[j], k = j
7    while k ≥ 2 and A[k − 1] > key
8      A[k] = A[k − 1], k = k − 1
9    end
10   A[k] = key
11 end
```



line	lower bound on count	upper bound on count
5	$n-1$	$n-1$
6	$3(n-1)$	$3(n-1)$
7	$3(n-1)$	$3(1+2+3+\dots+n-1)$
8	0	$5(1+2+\dots+n-1)$
10	$n-1$	$n-1$
	$\sum = f_L$	$\sum = f_u$

$$\boxed{\sum_{i=1}^{n} i = \frac{(n+1)n}{2}}$$

$$f_L = (n-1) + 3(n-1) + 3(n-1) + (n-1) = 8n - 8$$

$$f_u = (n-1) + 3(n-1) + 8\sum_{k=1}^{n-1} k + n - 1 = \frac{8n(n-1)}{2} + 5(n-1) = 4n^2 + n - 9$$

Results: $f_L \in \Omega(n)$ $\quad f_u \in O(n^2)$