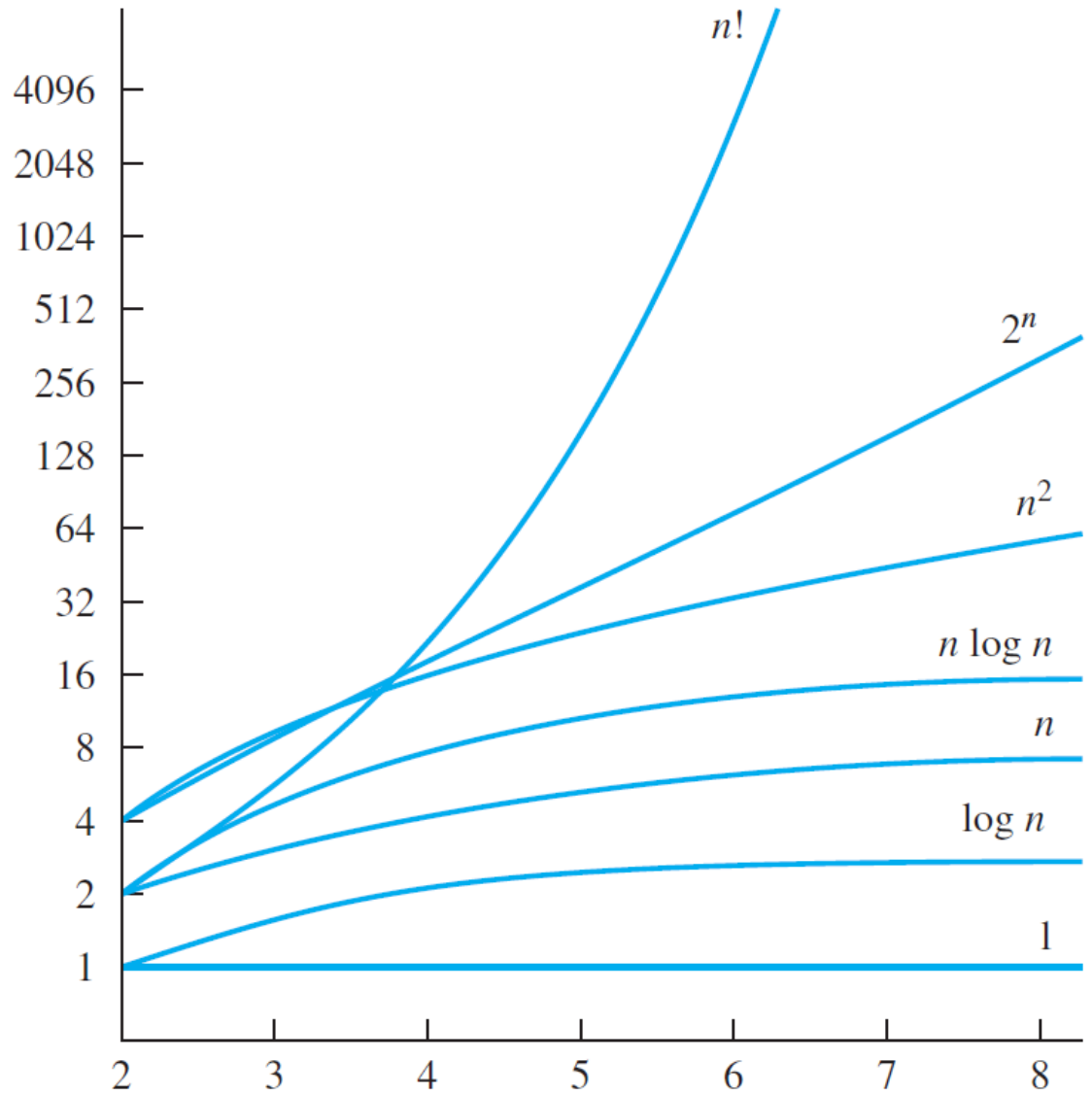
Design and Analysis of Algorithms

Analysis of Algorithm Efficiency

- A difference in running times on small inputs is not what really distinguishes efficient algorithms from inefficient ones.
- For example, the greatest common divisor of two small numbers, it is not clear how much more efficient Euclid's algorithm is compared to other algorithms.
- Algorithm speed isn't measured in seconds, but in growth of the number of operations.
- Instead, we talk about how quickly the run time of an algorithm increases as the size of the input increases.

Orders of Growth

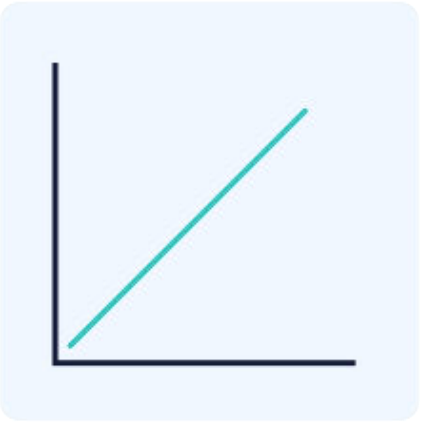


Orders of Growth

$\Theta(1)$

$\Theta(\log N)$

$\Theta(N)$



$\Theta(N \log N)$

$\Theta(N^2)$

$\Theta(2^N)$

$\Theta(N!)$

Orders of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

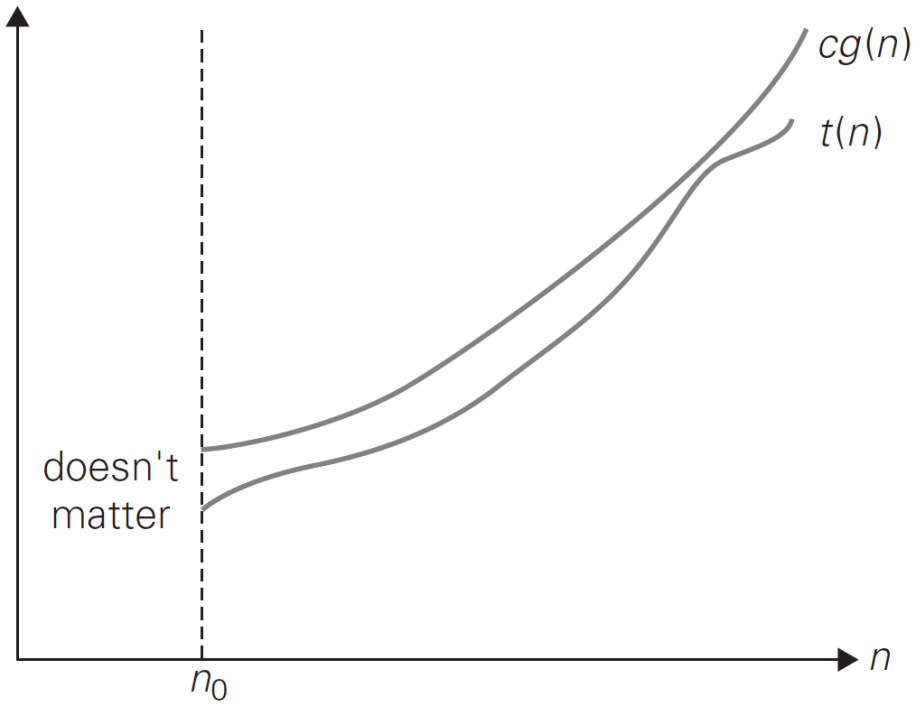
Orders of Growth

n	log n	n	n log n	n^2	2^n	n!
10	0.003ns	0.01ns	0.033ns	0.1ns	1ns	3.65ms
20	0.004ns	0.02ns	0.086ns	0.4ns	1ms	77years
30	0.005ns	0.03ns	0.147ns	0.9ns	1sec	$8.4x10^{15}$yrs
40	0.005ns	0.04ns	0.213ns	1.6ns	18.3min	--
50	0.006ns	0.05ns	0.282ns	2.5ns	13days	--
100	0.07	0.1ns	0.644ns	0.10ns	$4x10^{13}$yrs	--
1,000	0.010ns	1.00ns	9.966ns	1ms	--	--
10,000	0.013ns	10ns	130ns	100ms	--	--
100,000	0.017ns	0.10ms	1.67ms	10sec	--	--
1'000,000	0.020ns	1ms	19.93ms	16.7min	--	--
10'000,000	0.023ns	0.01sec	0.23ms	1.16days	--	--
100'000,000	0.027ns	0.10sec	2.66sec	115.7days	--	--
1,000'000,000	0.030ns	1sec	29.90sec	31.7 years	--	--

O-notation

A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n. If there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \leq cg(n) \quad \text{for all } n \geq n_0$$



O-notation: Example 1

$$t(n) = 100n + 5$$

$$100n + 5 \leq 100n + 5n \qquad (\text{for } n \geq 1)$$

$$100n + 5 \leq 105n \qquad (\text{for } n \geq 1)$$

$$c = 105$$

$$n_0 = 1$$

$$t(n) \in O(n)$$

O-notation: Example 2

$$t(n) = n^2 + 2n + 1$$

$$n^2 + 2n + 1 \leq n^2 + 2n^2 + 1n^2 \qquad (\text{for } n \geq 1)$$

$$n^2 + 2n + 1 \leq 4n^2 \qquad (\text{for } n \geq 1)$$

$$c = 4$$

$$n_0 = 1$$

$$t(n) \in O(n^2)$$

O-notation: Example 3

$$t(n) = 5n^4 + 3n^3 + 2n^2 + 4n + 1$$

$$5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq 5n^4 + 3n^4 + 2n^4 + 4n^4 + 1n^4 \qquad (\text{for } n \geq 1)$$

$$5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq 15n^4 \qquad (\text{for } n \geq 1)$$

$$c = 15$$

$$n_0 = 1$$

$$t(n) \in O(n^4)$$

O-notation: Example 4

$$t(n) = 20n^3 + 10n\log(n) + 5$$

$$20n^3 + 10n\log(n) + 5 \leq 20n^3 + 10n^3 + 5n^3 \qquad (\text{for } n \geq 1)$$

$$20n^3 + 10n\log(n) + 5 \leq 35n^3 \qquad (\text{for } n \geq 1)$$

$$c = 35$$

$$n_0 = 1$$

$$t(n) \in O(n^3)$$

O-notation: Example 5

$$t(n) = 2n + 100\log(n)$$

$$2n + 100\log(n) \leq 2n + 100n \qquad (\text{for } n \geq 1)$$

$$2n + 100\log(n) \leq 102n \qquad (\text{for } n \geq 1)$$

$$c = 102$$

$$n_0 = 1$$

$$t(n) \in O(n)$$

O-notation: Example 6

$$t(n) = 3\log(n) + 2$$

$$3\log(n) + 2 \leq 3\log(n) + 2\log(n) \qquad (\text{for } n \geq 2)$$

$$3\log(n) + 2 \leq 5\log(n) \qquad (\text{for } n \geq 2)$$

$$c = 5$$

$$n_0 = 2$$

$$t(n) \in O(\log(n))$$

O-notation: Example 7

$$t(n) = 2^{n+2}$$

$$2^{n+2} = 2^2 \times 2^n = 4 \times 2^n$$

$$2^{n+2} \leq 4 \times 2^n \qquad (\text{for } n \geq 1)$$

$$c = 4$$

$$n_0 = 1$$

$$t(n) \in O(2^n)$$

O-notation: Example 8

$$t(n) = 10$$

$$10 \leq 10n^0 \qquad (\text{for } n \geq 1)$$

$$10 \leq 10 \times 1 \qquad (\text{for } n \geq 1)$$

$$c = 10$$

$$n_0 = 1$$

$$t(n) \in O(1)$$

O-notation: Example 9

$$t(n) = 1og_2(n^2)$$

$$1og_2(n^2) = 21og_2(n)$$

$$1og_2(n^2) \leq 21og_2(n) \qquad (\text{for } n \geq 1)$$

$$c = 2$$

$$n_0 = 1$$

$$t(n) \in O(1og_2(n))$$

O-notation: Example 10

Show that $t(n) = 7n^2 \in O(n^3)$

$7n^2 \leq 7n^3 \qquad (\text{for } n \geq 1)$

$c = 7$

$n_0 = 1$

$t(n) \in O(n^3)$

O-notation: Example 11

Show that $t(n) = n! \in O(n^n)$

$n! = 1.2.3...n$

$n^n = n.n.n...n$

$1.2.3...n \leq n.n.n...n \qquad (\text{for } n \geq 1)$

$n! \leq n^n \qquad (\text{for } n \geq 1)$

$c = 1$

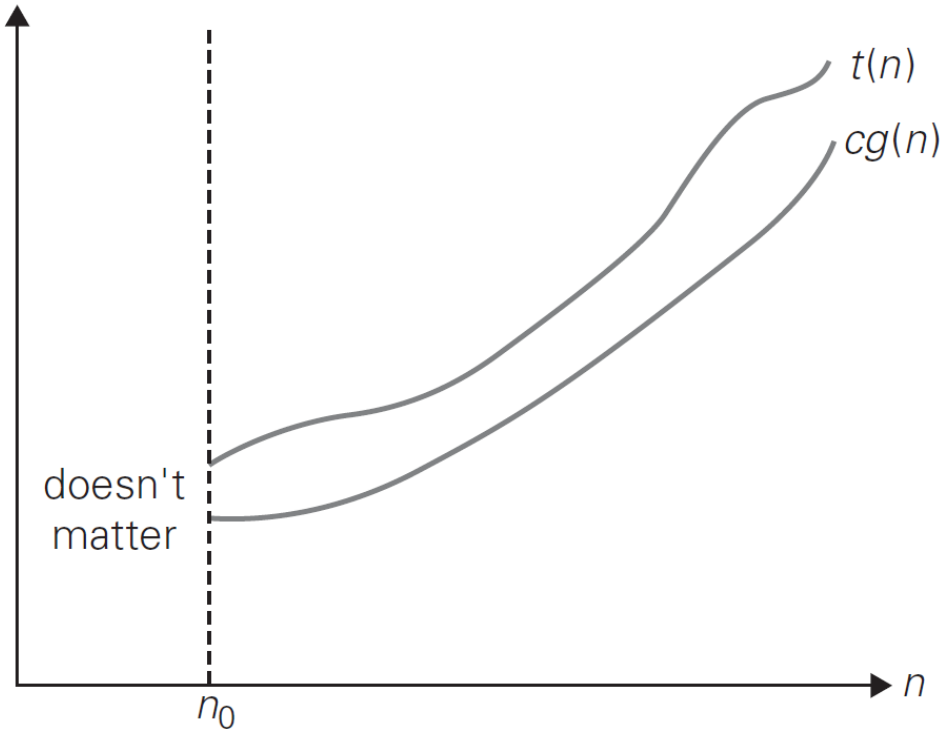
$n_0 = 1$

$t(n) \in O(n^n)$

Ω-notation

A function $t(n)$ is said to be in $\Omega(g(n))$ if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n. If there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \geq cg(n) \qquad \text{for all } n \geq n_0$$



Ω-notation: Example 1

Show that $t(n) = 8n^3 + 5n^2 + 7 \in \Omega(n^3)$

$8n^3 + 5n^2 + 7 \geq n^3 \qquad (\text{for } n \geq 1)$

$c = 1$

$n_0 = 1$

$t(n) \in \Omega(n^3)$

Ω-notation: Example 2

Show that $t(n) = n^3 + 4n^2 \in \Omega(n^2)$

$n^3 + 4n^2 \geq n^2 \qquad (\text{for } n \geq 1)$

$c = 1$

$n_0 = 1$

$t(n) \in \Omega(n^2)$

Ω-notation: Example 3

Show that $t(n) = 3n\log(n) - 2n \in \Omega(n\log(n))$

$3n\log(n) - 2n \geq n\log(n) \qquad (\text{for } n \geq 2)$

$c = 1$

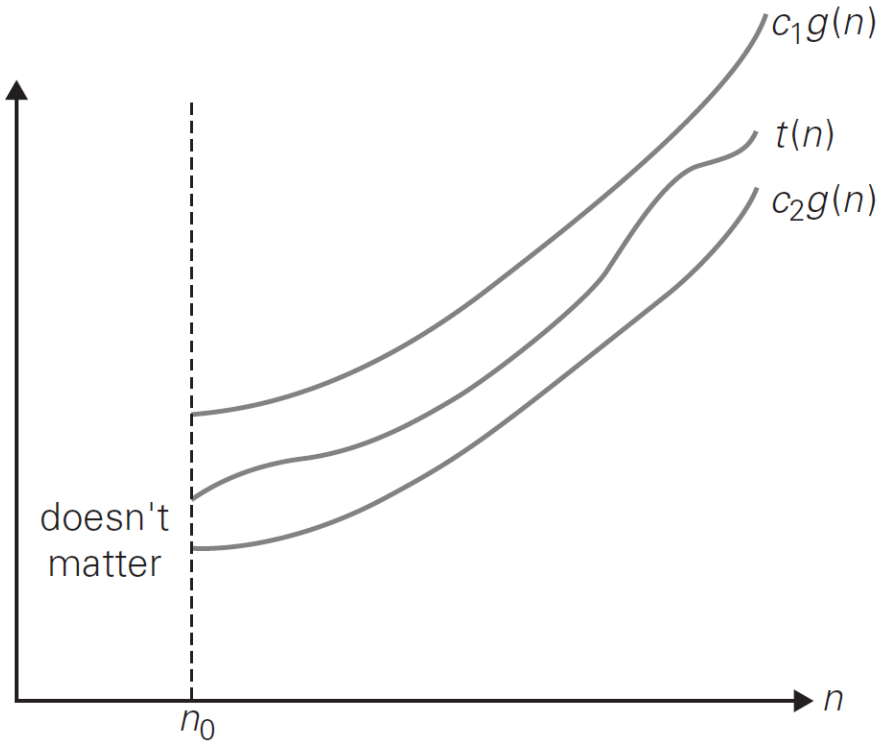
$n_0 = 2$

$t(n) \in \Omega(n\log(n))$

Θ-notation

A function $t(n)$ is said to be in $\Theta(g(n))$ if $t(n)$ is bounded both above and below by some constant multiples of $g(n)$ for all large n. If there exist some positive constants c_1 and c_2 and some nonnegative integer n_0 such that

$$c_1\, g(n) \le t(n) \le c_2\, g(n) \qquad \text{for all } n \ge n_0$$



Θ-notation: Example 1

$$t(n) = 4n + 3$$

$$n \leq 4n + 3 \leq 4n + 3n \qquad (\text{for } n \geq 1)$$

$$n \leq 4n + 3 \leq 7n \qquad (\text{for } n \geq 1)$$

$$c_1 = 1$$

$$c_2 = 7$$

$$n_0 = 1$$

$$t(n) \in \Theta(n)$$

Θ-notation: Example 2

$$t(n) = 3n^2 + 8n\log(n)$$

$$n^2 \le 3n^2 + 8n\log(n) \le 3n^2 + 8n^2 \qquad (\text{for } n \ge 1)$$

$$n^2 \le 3n^2 + 8n\log(n) \le 11n^2 \qquad (\text{for } n \ge 1)$$

$$c_1 = 1$$

$$c_2 = 11$$

$$n_0 = 1$$

$$t(n) \in \Theta(n^2)$$

Θ-notation: Example 3

$$t(n) = 3n\log(n) + 4n + 5\log(n)$$

$$3n\log(n) \leq 3n\log(n) + 4n + 5\log(n) \leq 3n\log(n) + 4n\log(n) + 5n\log(n)$$
$$3n\log(n) \leq 3n\log(n) + 4n + 5\log(n) \leq 12n\log(n)$$

$$c_1 = 3$$
$$c_2 = 12$$
$$n_0 = 2$$
$$t(n) \in \Theta(n\log(n))$$

Θ-notation: Example 4

$$t(n) = 2^{n+1} + 3^{n-1}$$

$$2^{n+1} + 3^{n-1} = 2^1 \times 2^n + 3^{-1} \times 3^n$$

$$3^{-1} \times 3^n \leq 2^{n+1} + 3^{n-1} \leq 3 \times 3^n \qquad (\text{for } n \geq 1)$$

$$c_1 = 3^{-1}$$

$$c_2 = 3$$

$$n_0 = 1$$

$$t(n) \in \Theta(3^n)$$

Θ-notation: Example 5

$$t(n) = (n^2 + 1)^{10}$$

$$(n^2 + 0)^{10} \leq (n^2 + 1)^{10} \leq (n^2 + n^2)^{10} \qquad (\text{for } n \geq 1)$$

$$(n^2)^{10} \leq (n^2 + 1)^{10} \leq (2n^2)^{10} \qquad (\text{for } n \geq 1)$$

$$n^{20} \leq (n^2 + 1)^{10} \leq 1024n^{20} \qquad (\text{for } n \geq 1)$$

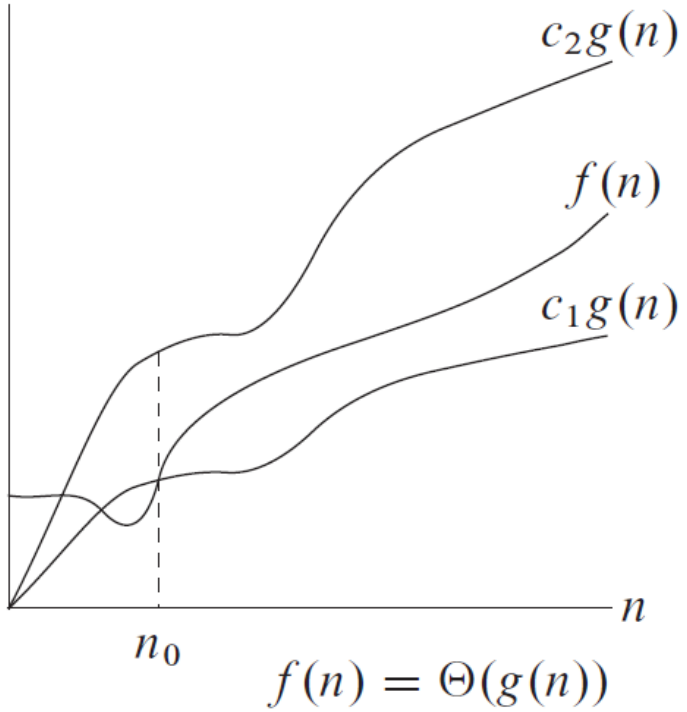
$$c_1 = 1$$

$$c_2 = 1024$$

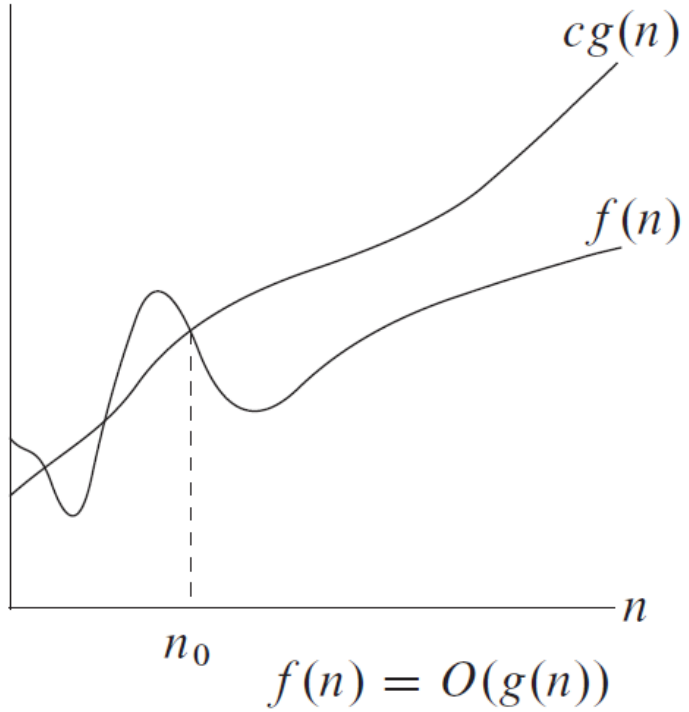
$$n_0 = 1$$

$$t(n) \in \Theta(n^{20})$$

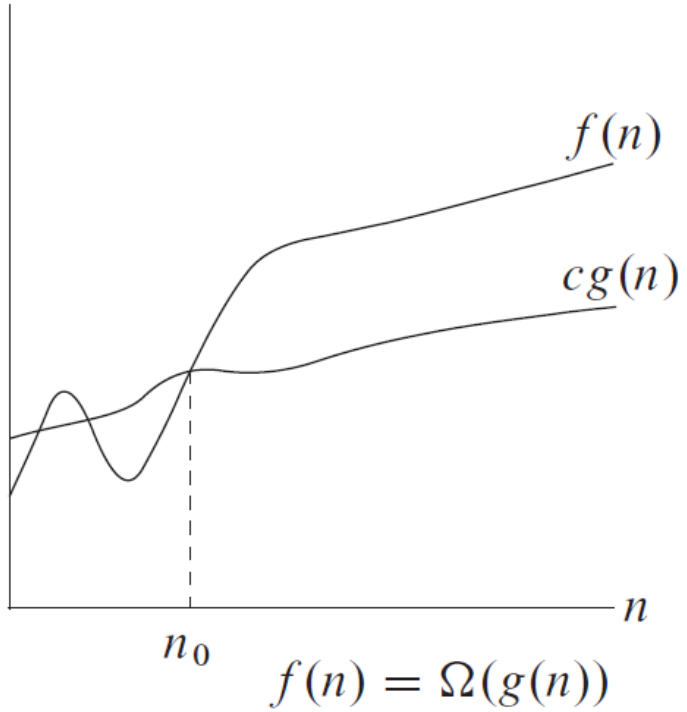
Asymptotic Notations: Summary



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



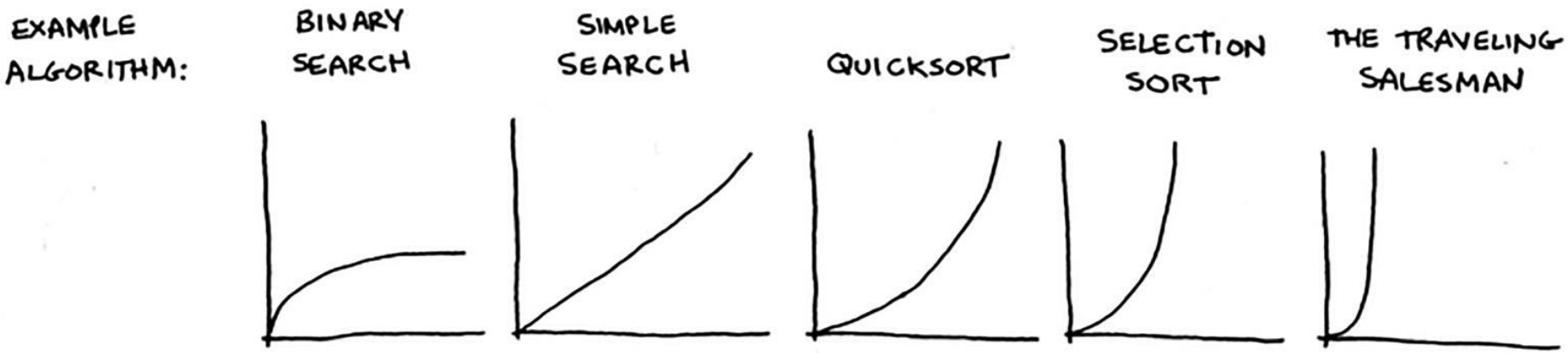
$f(n) = O(g(n))$



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

The Most Common Big O Run Times

Estimates based on a slow computer that performs 10 operations per second.



EXAMPLE ALGORITHM:	BINARY SEARCH	SIMPLE SEARCH	QUICKSORT	SELECTION SORT	THE TRAVELING SALESMAN
ARRAY SIZE	O(log n)	O(n)	O(n log n)	O(n²)	O(n!)
10	0.3 sec	1 sec	3.3 sec	10 sec	4.2 days
100	0.6 sec	10 sec	664 sec	16.6 min	2.9×10^{149} years
1000	1 sec	100 sec	996 sec	27.7 hours	1.27×10^{2559} years

Exponential Growth Functions

- The exponential function 2^n and the factorial function $n!$
- Both these functions grow so fast that their values become large even for rather small values of n.
- There is a difference between the orders of growth of the functions 2^n and $n!$.
- Yet both are often referred to as "exponential-growth functions".
- Algorithms that require an exponential number of operations are practical for solving only problems of very small sizes.

Worst-Case, Best-Case, and Average-Case Efficiencies

- Time efficiency is measured by counting the number of times the algorithm's basic operation is executed.
- Space efficiency is measured by counting the number of extra memory units consumed by the algorithm.
- The efficiencies of some algorithms may differ significantly for inputs of the same size.
- For such algorithms, we need to distinguish between the worst-case, average-case, and best-case efficiencies.

Sequential Search

ALGORITHM *SequentialSearch* (A[0..$n-1$], K)

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element in A that matches K
// or −1 if there are no matching elements
for i ← 0 to n − 1 do
        if A[i] = K
                return i
return −1
```

Sequential Search: Worst-Case

- Clearly, the running time of this algorithm can be quite different for the same array size n.
- In the worst case, when there are no matching elements or the matching element happens to be the last one on the array, the algorithm makes the largest number of key comparisons among all possible inputs of size

$$T_{worst}(n) = n$$

Sequential Search: Best-Case

- The best-case efficiency of an algorithm is its efficiency for the best-case input of size n, which is an input of size n for which the algorithm runs the fastest among all possible inputs of that size.
- The best-case inputs for sequential search are arrays of size n with their first element equal to a search key.

$$T_{best}(n) = 1$$

Sequential Search: Average-Case

- It is clear from our discussion that neither the worst-case analysis nor its best-case counterpart yields the necessary information about an algorithm's behavior on a "typical" or "random" input.
- This is the information that the average-case efficiency seeks to provide.
- To analyze the algorithm's average-case efficiency, we must make some assumptions about possible inputs of size n.

Sequential Search: Average-Case

- The probability of a successful search is equal to p $(0 \le p \le 1)$.
- In the case of an unsuccessful search, the number of comparisons will be n with the probability of such a search being $(1 - p)$.

$$C_{avg}(n) = [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}] + n \cdot (1 - p)$$

$$= \frac{p}{n}[1 + 2 + \cdots + i + \cdots + n] + n(1 - p)$$

$$= \frac{p}{n}\frac{n(n+1)}{2} + n(1 - p) = \frac{p(n+1)}{2} + n(1 - p).$$

- If $p = 1$ (successful search), the average number of key comparisons made by sequential search is $(n + 1)/2$.

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)