Big-O Analysis

Definition: Suppose that f(n) and g(n) are nonnegative functions of n. Then we say that f(n) is O(g(n)) provided that there are constants $C > 0$ and $N > 0$ such that for all $n > N$, $f(n) \leq Cg(n)$.

Big-O expresses an upper bound on the growth rate of a function, for sufficiently large values of n.

By the definition above, demonstrating that a function f is big-O of a function g requires that we find specific constants C and N for which the inequality holds (and show that the inequality does, in fact, hold).

Big-O Example

Consider the following function: $T(n) = \frac{5}{2}n^2 + \frac{5}{2}n + 2$

We might guess that: $T(n) \le 5n^2$ for all $n \ge 2$

We could easily verify the guess by induction:

If n = 2, then T(2) = 17 which is less than 20, so the guess is valid if n = 2.

Assume that for some $n \ge 2$, $T(n) \le 5n^2$. Then:

$$T(n+1) = \frac{5}{2}(n+1)^2 + \frac{5}{2}(n+1) + 2 = \frac{5}{2}n^2 + 5n + \frac{5}{2} + \frac{5}{2}n + \frac{5}{2} + 2$$

$$= \left(\frac{5}{2}n^2 + \frac{5}{2}n + 2\right) + 5n + 5$$

$$\le 5n^2 + 5n + 5 \qquad \text{by the inductive assumption}$$

$$\le 5n^2 + 10n + 5 = 5(n+1)^2$$

Thus, by induction, $T(n) \le 5n^2$ for all $n \ge 2$. So, by definition, $T(n)$ is $O(n^2)$.

Making the Guess

The obvious question is "how do we come up with the guess in the first place"?

Here's one possible analysis (which falls a bit short of being a proof):

$$
\begin{aligned}
T(n) &= \frac{5}{2}n^2 + \frac{5}{2}n + 2 \\
&\leq \frac{5}{2}n^2 + \frac{5}{2}n^2 + 2 - 2 \quad \text{(replace n with n\^2, subtract 2)} \\
&= 5n^2
\end{aligned}
$$

The middle step seems sound since if $n \geq 2$ then $n \leq n^2$, substituting n^2 will thus add at least 5 to the expression, so that subtracting 2 should still result in a larger value.

Big-O Theorems

For all the following theorems, assume that f(n) is a non-negative function of n and that K is an arbitrary positive constant.

Theorem 1:	K is O(1)

Theorem 2:	A polynomial is O(the term containing the highest power of n)

$$f(n) = 7n^4 + 3n^2 + 5n + 1000 \text{ is } O(7n^4)$$

Theorem 3:	K*f(n) is O(f(n)) [i.e., constant coefficients can be dropped]

$$g(n) = 7n^4 \text{ is } O(n^4)$$

Theorem 4:	If f(n) is O(g(n)) and g(n) is O(h(n)) then f(n) is O(h(n)). [transitivity]

$$f(n) = 7n^4 + 3n^2 + 5n + 1000 \text{ is } O(n^4)$$

Big-O Theorems

Theorem 5: Each of the following functions is **strictly** big-O of its successors:

- K [constant]
- $\log_b(n)$ [always log base 2 if no base is shown]
- n
- $n \log_b(n)$
- n^2
- n to higher powers
- 2^n
- 3^n
- larger constants to the n-th power
- n! [n factorial]
- n^n

$$f(n) = 3n\log(n) \text{ is } O(n\log(n)) \text{ and } O(n^2) \text{ and } O(2^n)$$

Big-O Theorems

Theorem 6: In general, f(n) is big-O of the dominant term of f(n), where "dominant" may usually be determined from Theorem 5.

$$f(n) = 7n^2 + 3n\log(n) + 5n + 1000 \text{ is } O(n^2)$$

$$g(n) = 7n^4 + 3^n + 1000000 \text{ is } O(3^n)$$

$$h(n) = 7n(n + \log(n)) \text{ is } O(n^2)$$

Theorem 7: For any base b, $\log_b(n)$ is $O(\log(n))$.

Big-Omega

In addition to big-O, we may seek a lower bound on the growth of a function:

Definition: Suppose that f(n) and g(n) are nonnegative functions of n. Then we say that f(n) is $\Omega(g(n))$ provided that there are constants $C > 0$ and $N > 0$ such that for all $n > N$, $f(n) \geq Cg(n)$.

Big- Ω expresses a lower bound on the growth rate of a function, for sufficiently large values of n.

Analagous theorems can be proved for big-Ω.

Big-Theta

Finally, we may have two functions that grow at essentially the same rate:

Definition: Suppose that f(n) and g(n) are nonnegative functions of n. Then we say that f(n) is $\Theta(g(n))$ provided that f(n) is $O(g(n))$ and also that f(n) is $\Omega(g(n))$.

Alernative: Suppose that f(n) and g(n) are nonnegative functions of n. Then we say that f(n) is $\Theta(g(n))$ provided that there exist constants $C1 > 0$, $C2 > 0$ and $N > 0$ such that, for all $n > N$, $C_1 g(n) \leq f(n) \leq C_2 g(n)$.

If f is $\Theta(g)$ then, from some point on, f is bounded below by one multiple of g and bounded above by another multiple of g (and vice versa).

So, in a very basic sense f and g grow at the same rate.

Order and Limits

The task of determining the order of a function is simplified considerably by the following result:

Theorem 8: f(n) is $\Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ where $0 < c < \infty$

Recall Theorem 7… we may easily prove it (and a bit more) by applying Theorem 8:

$$\lim_{n\to\infty} \frac{\log_b(n)}{\log(n)} = \lim_{n\to\infty} \frac{\frac{1}{n\ln(b)}}{\frac{1}{n\ln(2)}} = \lim_{n\to\infty} \frac{\ln(2)}{\ln(b)} = \frac{\ln(2)}{\ln(b)}$$

The last term is finite and positive, so $\log_b(n)$ is $\Theta(\log(n))$ by Theorem 8.

Corollary: if the limit above is 0 then f(n) is strictly O(g(n)), and

if the limit is ∞ then f(n) is strictly $\Omega(g(n))$.

Order and Limits

The converse of Theorem 8 is false. However, it is possible to prove:

Theorem 9: If f(n) is Θ(g(n)) then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{where} \quad 0 < c < \infty$$

provided that the limit exists.

A similar extension of the preceding corollary also follows.

More Theorems

Many of the big-O theorems may be strengthened to statements about big-Θ:

Theorem 10: If K > 0 is a constant, then K is Θ(1).

Theorem 11: A polynomial is Θ(the highest power of n).

proof: Suppose a polynomial of degree k. Then we have:

$$\lim_{n\to\infty} \frac{a_0 + a_1 n + \cdots + a_k n^k}{n^k} = \lim_{n\to\infty} \left(\frac{a_0}{n^k} + \frac{a_1}{n^{k-1}} + \cdots + \frac{a_{k-1}}{n} + a_k \right) = a_k$$

Now $a_k > 0$ since we assume the function is nonnegative. So by Theorem 8, the polynomial is $\Theta(n^k)$.

QED

More Theorems

Theorems 3, 6 and 7 can be similarly extended.

Theorem 12:	K*f(n) is Θ(f(n)) [i.e., constant coefficients can be dropped]

Theorem 13:	In general, f(n) is big-Θ of the dominant term of f(n), where "dominant" may usually be determined from Theorem 5.

Theorem 14:	For any base b, $\log_b(n)$ is Θ(log(n)).

Strict Comparisons

For convenience, we will say that:

- f is *strictly* O(g) if and only if f is O(g) but f is not Θ(g)
- f is *strictly* Ω(g) if and only if f is Ω(g) but f is not Θ(g)

For example, n log n is strictly $O(n^2)$ by Theorem 8 and its corollary, because:

$$\lim_{n\to\infty} \frac{n \log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1/n}{1} = 0$$

Big-Θ Is an Equivalence Relation

Theorem 15: If f(n) is Θ(f(n)). [reflexivity]

Theorem 16: If f(n) is Θ(g(n)) then g(n) is Θ(f(n)). [symmetry]

Theorem 17: If f(n) is Θ(g(n)) and g(n) is Θ(h(n)) then f(n) is Θ(h(n)). [transitivity]

By Theorems 15–17, Θ is an equivalence relation on the set of positive-valued functions.

The equivalence classes represent fundamentally different growth rates.

Algorithms whose complexity functions belong to the same class are essentially equivalent in performance (up to constant multiples, which are not unimportant in practice).

Applications to Algorithm Analysis

Ex 1: An algorithm with complexity function

$T(n) = \frac{3}{2}n^2 + \frac{5}{2}n - 3$ is $\Theta(n^2)$ by Theorem 10.

Ex 2: An algorithm with complexity function

$T(n) = 3n\log n + 4\log n + 2$ is O(n log(n)) by Theorem 5.

Furthermore, the algorithm is also $\Theta(n \log(n))$ by Theorem 8 since:

$$\lim_{n\to\infty} \frac{T(n)}{n\log n} = \lim_{n\to\infty}\left(3 + \frac{4}{n} + \frac{2}{n\log n}\right) = 3$$

For most common complexity functions, it's this easy to determine the big-O and/or big-Θ complexity using the given theorems.

Complexity of Linear Storage

For a contiguous list of N elements, assuming each is equally likely to be the target of a search:

- average search cost is $\Theta(N)$ if list is randomly ordered
- average search cost is $\Theta(\log N)$ is list is sorted
- average random insertion cost is $\Theta(N)$
- insertion at tail is $\Theta(1)$

For a linked list of N elements, assuming each is equally likely to be the target of a search:

- average search cost is $\Theta(N)$, regardless of list ordering
- average random insertion cost is $\Theta(1)$, excluding search time

Most Common Complexity Classes

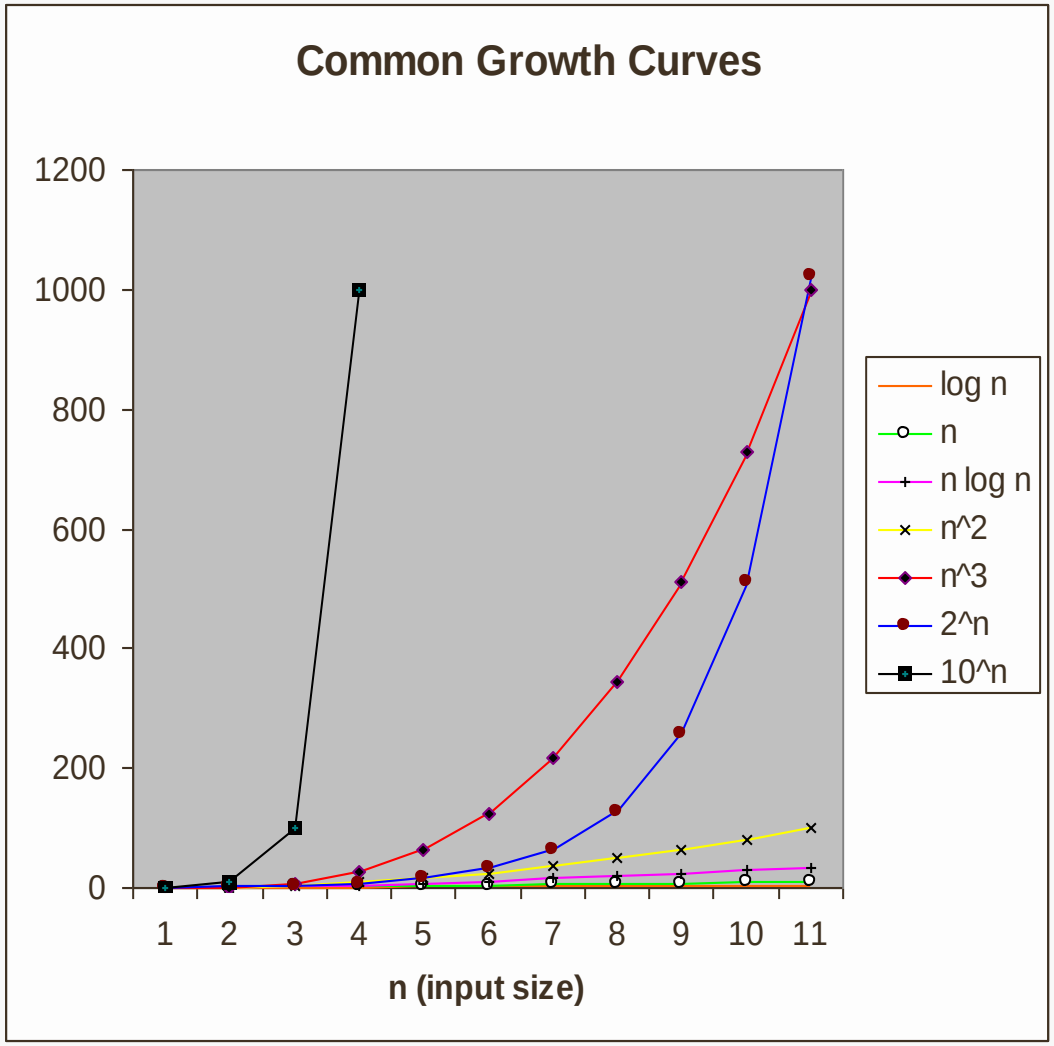
Theorem 5 lists a collection of representatives of distinct big-Θ equivalence classes:

K	[constant]
$\log_b(n)$	[always log base 2 if no base is shown]
n	
$n \log_b(n)$	
n^2	
n to higher powers	
2^n	
3^n	
larger constants to the n-th power	
n! [n factorial]	
n^n	

Most common algorithms fall into one of these classes.

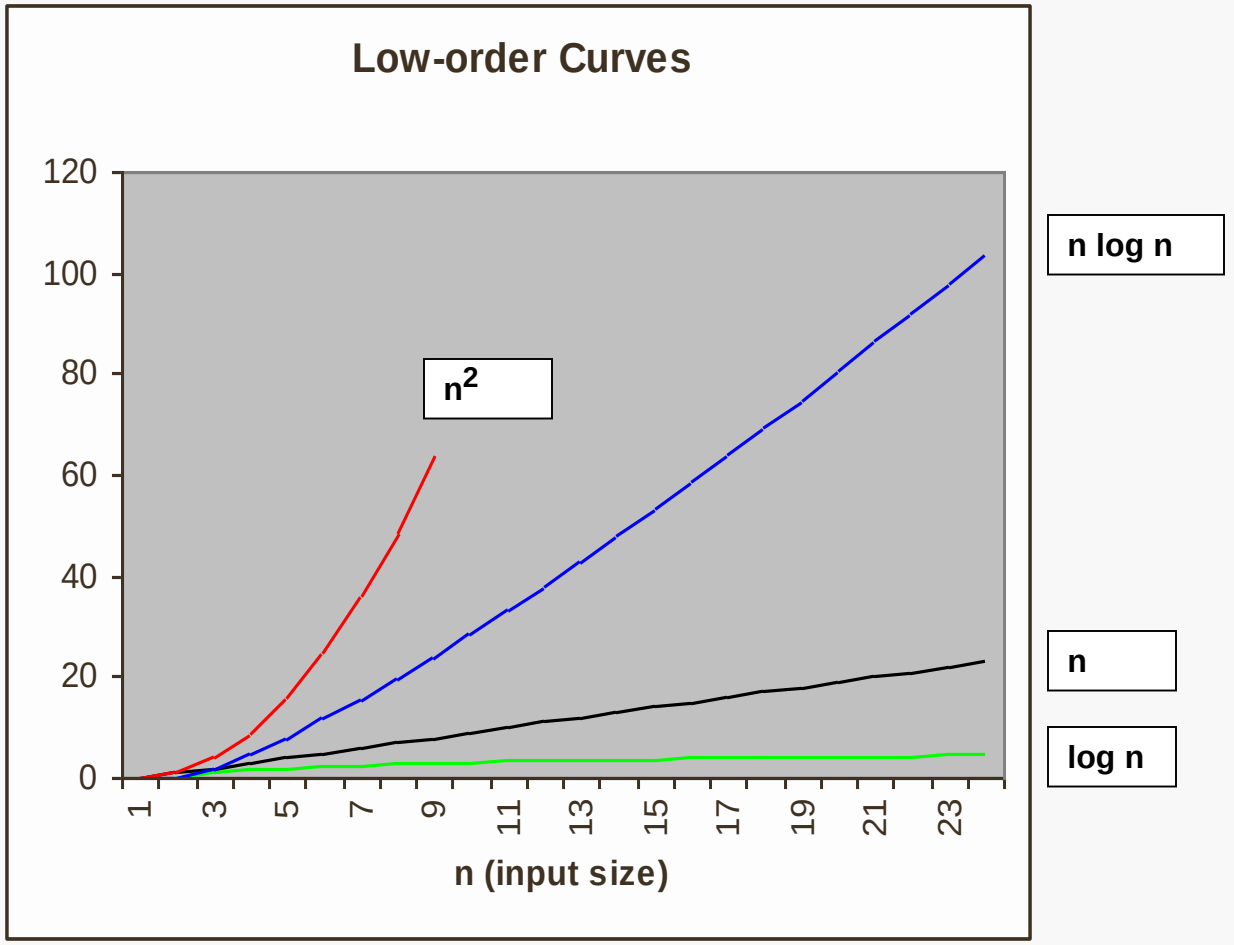
Knowing this list provides some knowledge of how to compare and choose the right algorithm. The following charts provide some visual indication of how significant the differences are…

Graphical Comparison



Lower-order Classes

For significantly large values of n, only these classes are truly practical, and whether n^2 is practical is debated.



Proof of Theorem 8

Theorem 8: f(n) is Θ(g(n)) if

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = c \text{ where } 0 < c < \infty$$

Suppose that f and g are non-negative functions of n, and that

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = c \text{ where } 0 < c < \infty$$

Then, from the definition of the limit, for every ε > 0 there exists an N > 0 such that whenever n > N:

$$\left|\frac{f(n)}{g(n)} - c\right| < \varepsilon \text{ from which we have } (c-\varepsilon)g(n) \le f(n) \le (c+\varepsilon)g(n)$$

Let ε = c/2, then we have that:

$$f(n) \le \frac{3c}{2}g(n) \text{ whence } f \text{ is } O(g)$$

$$\frac{c}{2}g(n) \le f(n) \text{ whence } f \text{ is } \Omega(g)$$

Therefore, by definition, f is Θ(g).