

# Review of Big-O Notation

Last Updated: August 24th, 2024

## 1 Big-O Notation

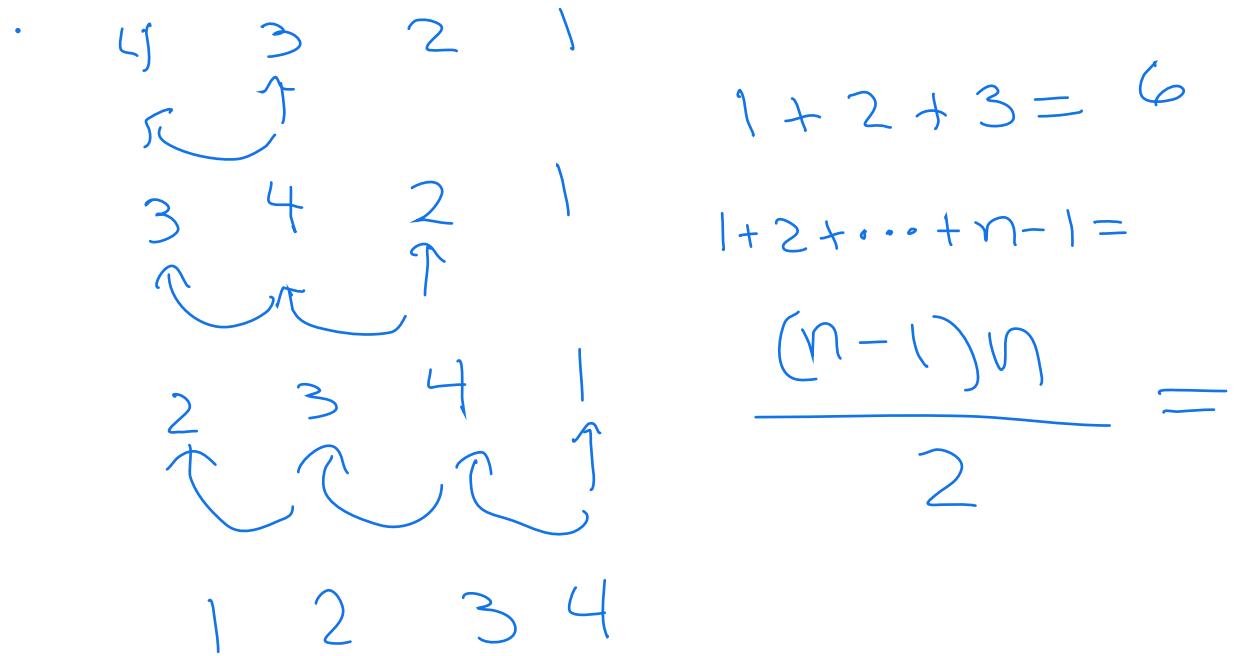
Big-O notation is useful for making statements about the growth of a function  $f(n)$ ,  $n$  a natural number, whose values may seem difficult or impossible to compute. The statements we care most about are those that state upper and/or lower bounds on  $f$ 's growth. Although we may not know the exact bounds (since we may not know the exact values of  $f$ ), we may be able to determine meaningful ones in case we know the following two things:

1. the rule, call it  $g(n)$ , for the fastest growing term (ignoring constants) of the bounding function, and
2. that there exists a constant  $c > 0$  such that,  $cg(n)$  provides a bound for  $f$ , for sufficiently large  $n$ .

**Example 1.1.** Carol has programmed the **Insertion Sort** algorithm to run on her laptop. What upper bound can she provide on the elapsed time  $t(n)$  that will occur on her laptop clock after **Insertion Sort** has sorted an integer array of size  $n$ ? She knows that the worst case occurs when the input array is sorted in reverse order, and in this case a total of

$$\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

comparisons and swaps must be performed in order to sort such an array. In this case, the rule for the fastest growing term of the upper-bounding function is  $g(n) = n^2$ . Also, she knows that each comparison and swap requires at most two machine instructions, and that each machine instruction requires at most  $10^{-8}$  seconds to execute. Therefore, there is a  $c > 0$  such that  $cg(n)$  is an upper bound for the elapsed time.  $\square$



Let  $f(n)$  and  $g(n)$  be functions from the set of nonnegative integers to the set of nonnegative real numbers. Then

**Big-O**  $f(n) = O(g(n))$  iff there exist constants  $c > 0$  and  $k \geq 1$  such that  $f(n) \leq cg(n)$  for every  $n \geq k$ .

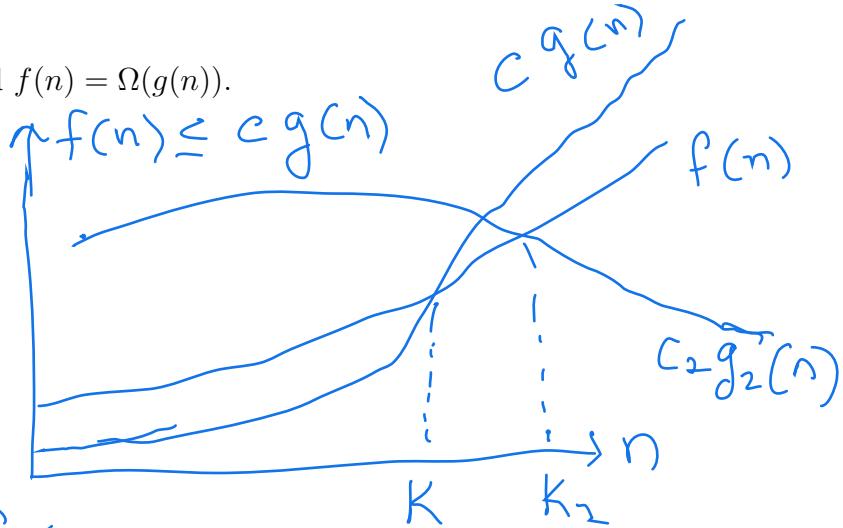
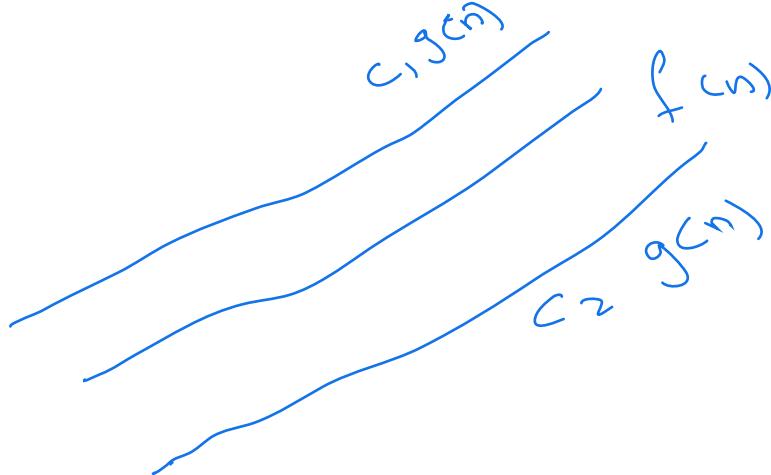
**Big- $\Omega$**   $f(n) = \Omega(g(n))$  iff there exist constants  $c > 0$  and  $k \geq 1$  such that  $f(n) \geq cg(n)$  for every  $n \geq k$ .

**Big- $\Theta$**   $f(n) = \Theta(g(n))$  iff  $f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$ .

**little-o**  $f(n) = o(g(n))$  iff  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ .

**little- $\omega$**   $f(n) = \omega(g(n))$  iff  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ .

$$f(n) \geq c_2 g_2(n)$$



**Example 1.2.** From the above discussion we have  $f(n) = 3.5n^2 + 4n + 36 = \Theta(n^2)$  since

$$n^2 \leq 3.5n^2 \leq f(n) = 3.5n^2 + 4n + 36 \leq 3.5n^2 + 4n^2 + 36n^2 = 43.5n^2,$$

is true for all  $n \geq 1$ . And so  $f(n) = \Theta(n^2)$ , where  $c_1 = 3.5$  and  $c_2 = 43.5$  are the respective lower and upper-bound constants.  $\square$

$$\rightarrow f(n) = \Theta(n^2)$$

$$f(n) = O(n^2)$$

$$\text{via } c_2 = 43.5$$

**Definition 1.3.** The following table shows the most common kinds of rules for  $g(n)$  that are used within big-O notation.

Function	Type of Growth
1	constant growth
$\log n$	logarithmic growth
$\log^k n$ , for some integer $k \geq 1$	polylogarithmic growth
$n^k$ for some positive $k < 1$	sublinear growth
$n$	linear growth
$n \log n$	log-linear growth
$n \log^k n$ , for some integer $k \geq 1$	polylog-linear growth
$n^j \log^k n$ , for some integers $j, k \geq 1$	polylog-polynomial growth
$n^2$	quadratic growth
$n^3$	cubic growth
$n^k$ for some integer $k \geq 1$	polynomial growth
$2^{\log^c n}$ , for some $c > 1$	quasi-polynomial growth
$\omega(n^k)$ , for all integers $k \geq 1$	superpolynomial growth
$a^n$ for some $a > 1$	exponential growth

$$2^{\log^2 n} = 2^{\log n \log n}$$

$$\begin{aligned}
 2^{\log_2 n} &= n \\
 \log_2 n &= ? \Rightarrow \\
 ? &= 2 = n \\
 \underline{2^{\log n}} &= ? \\
 2^{\log n} &= 2 = n \\
 (2^{\log n})^{\log n} &= n^{\log n}
 \end{aligned}$$

**Example 1.4.** Returning to Example 1.1, using big-O notation Carol can say that, when running **Insertion Sort** on her laptop with an input of size  $n$ , the elapsed time equals  $O(n^2)$  seconds. Also, since **Insertion Sort** requires at least  $n$  comparisons for any input, she may also say that its running time equals  $\Omega(n)$  seconds.  $\square$

**Theorem 1.5.** The following are all true statements.

1.  $1 = o(\log n)$
2.  $\log n = o(n^\epsilon)$  for any  $\epsilon > 0$
3.  $\log^k n = o(n^\epsilon)$  for any  $k > 0$  and  $\epsilon > 0$
4.  $n^a = o(n^b)$  if  $a < b$ , and  $n^a = \Theta(n^b)$  if  $a = b$ .
5.  $n^k = o(2^{\log^c n})$ , for all  $k > 0$  and  $c > 1$ .
6.  $2^{\log^c n} = o(a^n)$  for all  $a, c > 1$ .
7. For nonnegative functions  $f(n)$  and  $g(n)$ ,

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

8. If  $f(n) = \Theta(h(n))$  and  $g(n) = \Theta(k(n))$ , then  $(fg)(n) = \Theta((hk)(n))$ .
9. If  $f(n) = o(g(n))$  then  $f(n) = O(g(n))$ .
10. If  $f(n) = \omega(g(n))$  then  $f(n) = \Omega(g(n))$ .

**Example 1.6.** For each of the following, state whether  $f(n) = O(g(n))$ ,  $f(n) = \Omega(g(n))$ , or both, i.e.  $f(n) = \Theta(g(n))$ .

1.  $f(n) = 3n + 5$ ,  $g(n) = 10n + 6 \log n$ .
2.  $f(n) = \sqrt{n} \cdot \log^2 n$ ,  $g(n) = \sqrt[3]{n} \log^3 n$ .
3.  $f(n) = 10 \log n$ ,  $g(n) = 50 \log n^2$ .
4.  $f(n) = n^2 / \log n$ ,  $g(n) = n \log^2 n$ .
5.  $f(n) = n2^n$ ,  $g(n) = 3^n$ .

## 2 Advanced Results

**Log Ratio Test.** Suppose  $f$  and  $g$  are continuous functions over the interval  $[1, \infty)$ , and

$$\lim_{n \rightarrow \infty} \log\left(\frac{f(n)}{g(n)}\right) = \lim_{n \rightarrow \infty} \log(f(n)) - \log(g(n)) = L.$$

Then

1. If  $L = \infty$  then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$$

2. If  $L = -\infty$  then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$$

3. If  $L \in (-\infty, \infty)$  is a constant then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 2^L.$$

**Integral Theorem.** Let  $f(x) > 0$  be an increasing or decreasing Riemann-integrable function over the interval  $[1, \infty)$ . Then

$$\sum_{i=1}^n f(i) = \Theta\left(\int_1^n f(x)dx\right),$$

if  $f$  is decreasing. Moreover, the same is true if  $f$  is increasing, provided  $f(n) = O(\int_1^n f(x)dx)$ .

**Proof of Integral Theorem.** We prove the case when  $f$  is decreasing. The case when  $f$  is increasing is left as an exercise. The quantity  $\int_1^n f(x)dx$  represents the area under the curve of  $f(x)$  from 1 to  $n$ . Moreover, for  $i = 1, \dots, n-1$ , the rectangle  $R_i$  whose base is positioned from  $x = i$  to  $x = i+1$ , and whose height is  $f(i+1)$  lies under the graph. Therefore,

$$\sum_{i=1}^{n-1} \text{Area}(R_i) = \sum_{i=2}^n f(i) \leq \int_1^n f(x)dx.$$

Adding  $f(1)$  to both sides of the last inequality gives

$$\sum_{i=1}^n f(i) \leq \int_1^n f(x)dx + f(1).$$

Now, choosing  $C > 0$  so that  $f(1) = C \int_1^n f(x)dx$  gives

$$\sum_{i=1}^n f(i) \leq (1 + C) \int_1^n f(x)dx,$$

which proves  $\sum_{i=1}^n f(i) = O(\int_1^n f(x)dx)$ .

Now, for  $i = 1, \dots, n-1$ , consider the rectangle  $R'_i$  whose base is positioned from  $x = i$  to  $x = i+1$ , and whose height is  $f(i)$ . This rectangle covers all the area under the graph of  $f$  from  $x = i$  to  $x = i+1$ . Therefore,

$$\sum_{i=1}^{n-1} \text{Area}(R'_i) = \sum_{i=1}^{n-1} f(i) \geq \int_1^n f(x)dx.$$

Now adding  $f(n)$  to the left side of the last inequality gives

$$\sum_{i=1}^n f(i) \geq \int_1^n f(x)dx,$$

which proves  $\sum_{i=1}^n f(i) = \Omega(\int_1^n f(x)dx)$ .

Therefore,

$$\sum_{i=1}^n f(i) = \Theta(\int_1^n f(x)dx).$$

### 3 Big-O Notation within the Context of Algorithms

Given a problem  $L$ , and an algorithm  $\mathcal{A}$  that solves  $L$ , big-O notation finds its use in the study of algorithms as a means for describing bounds on the number of steps and the amount of memory required by  $\mathcal{A}$  as a function of the **size parameters** of  $L$ , i.e. the parameters used to indicate the number of bits required to represent an instance of  $L$ .

**Example 3.1.** The following are examples of how big-O notation arises in the study of data structures and algorithms.

1. Inserting an item into a balanced tree of size  $n$  requires  $O(\log n)$  insertions.
2. It has been proven that, sorting  $n$  numbers using pairwise comparisons requires  $\Omega(n \log n)$  comparisons.
3. The Fast Fourier Transform algorithm has a running time of  $\Theta(m \log m)$ , where  $m$  is the degree of the input polynomial.
4. Two  $b$ -bit integers may be recursively added/subtracted in  $O(b)$  steps and recursively multiplied/divided in  $O(b^2)$ .

It is also common to see big-O notation used within an arithmetic expression, such as  $n^2 + o(n)$ , and  $n^{O(1)}$ .