

## Asymptotic Notations

### Definitions and Properties

**Definition 1** (Big-O). Let  $f = f(n)$  and  $g = g(n)$  be two positive functions over integers  $n$ . We say  $f = O(g)$ , if there exists positive number  $c > 0$  and integer  $N \geq 0$  such that  $f(n) \leq c \cdot g(n)$  for all  $n \geq N$ .

Similarly, we can define Big-O for multiple-variable functions.

**Definition 2** (Big-O). Let  $f = f(m, n)$  and  $g = g(m, n)$  be two positive functions over integers  $m$  and  $n$ . We say  $f = O(g)$ , if there exists positive number  $c > 0$  and integers  $M \geq 0$  and  $N \geq 0$  such that  $f(m, n) \leq c \cdot g(m, n)$  for all  $m \geq M$  and  $n \geq N$ .

Intuitively, Big-O is analogous to “ $\leq$ ”.

*Example.* Let  $f(m, n) = 4m + 4n + 5$  and  $g(m, n) = m + n$ . We now show that  $f = O(g)$ , using above definition. To show it, we need to find  $c$ ,  $M$ , and  $N$ . What are good choices for them? There are lots of choices; one set of it is:  $c = 7$ ,  $M = 1$ , and  $N = 1$ . Let's verify:  $f(m, n) - c \cdot g(m, n) = 4m + 4n + 5 - 7m - 7n = 5 - 3m - 3n \leq 5 - 3 - 3 = -1 \leq 0$ , where we use that  $m \geq M = 1$  and  $n \geq N = 1$ . This proves that  $f = O(g)$ .

**Definition 3** (Big-Omega). Let  $f = f(n)$  and  $g = g(n)$  be two positive functions over integers  $n$ . We say  $f = \Omega(g)$ , if there exists positive number  $c > 0$  and integer  $N \geq 0$  such that  $f(n) \geq c \cdot g(n)$  for all  $n \geq N$ .

Similarly, we can define Big-Omega for multiple-variable functions.

**Definition 4** (Big-O). Let  $f = f(m, n)$  and  $g = g(m, n)$  be two positive functions over integers  $m$  and  $n$ . We say  $f = \Omega(g)$ , if there exists positive number  $c > 0$  and integers  $M \geq 0$  and  $N \geq 0$  such that  $f(m, n) \geq c \cdot g(m, n)$  for all  $m \geq M$  and  $n \geq N$ .

Intuitively, Big-Omega is analogous to “ $\geq$ ”.

*Example.* Let  $f(m, n) = 4m + 4n + 5$  and  $g(m, n) = m + n$ . We now show that  $f = \Omega(g)$ , using above definition. To show it, we need to find  $c$ ,  $M$ , and  $N$ . We can choose:  $c = 1$ ,  $M = 0$ , and  $N = 0$ . Let's verify:  $f(m, n) - c \cdot g(m, n) = 4m + 4n + 5 - m - n = 5 + 3m + 3n \geq 5 \geq 0$ , where we use that  $m \geq M = 0$  and  $n \geq N = 0$ . This proves that  $f = \Omega(g)$ .

**Claim 1.**  $f = O(g)$  if and only if  $g = \Omega(f)$ .

*Proof.* We have

$$\begin{aligned}
 & f = O(g) \\
 \Leftrightarrow & \exists c > 0, N \geq 0, \text{ s.t. } f(n) \leq c \cdot g(n), \forall n \geq N \\
 \Leftrightarrow & \exists c > 0, N \geq 0, \text{ s.t. } 1/c \cdot f(n) \leq g(n), \forall n \geq N \\
 \Leftrightarrow & \exists c' = 1/c > 0, N \geq 0, \text{ s.t. } g(n) \geq c' \cdot f(n), \forall n \geq N \\
 \Leftrightarrow & g = \Omega(f)
 \end{aligned}$$

□

**Definition 5** (Big-Theta). We say  $f = \Theta(g)$  if and only if  $f = O(g)$  and  $f = \Omega(g)$ .

Intuitively, Big-Theta is analogous to “ $=$ ”.

*Example.* Let  $f(m, n) = 4m + 4n + 5$  and  $g(m, n) = m + n$ . We have  $f = \Theta(g)$  as we proved that  $f = O(g)$

and that  $f = \Omega(g)$ .

Below we give an equivalent description of Big-Theta.

**Fact 1.** Let  $f$  and  $g$  be two positive functions. Then  $f = \Theta(g)$  if and only if  $\lim_{n \rightarrow \infty} f(n)/g(n) = c > 0$ .

*Example.* Let  $f(m, n) = 4m + 4n + 5$  and  $g(m, n) = m + n$ . We now show  $f = \Theta(g)$  using above fact.  $\lim_{m \rightarrow \infty, n \rightarrow \infty} f/g = \lim_{m \rightarrow \infty, n \rightarrow \infty} (4m + 4n + 5)/(m + n) = 4 > 0$ . Hence,  $f = \Theta(g)$ .

**Definition 6** (small-o). Let  $f = f(n)$  and  $g = g(n)$  be two positive functions. We say  $f = o(g)$  if and only if  $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$ .

Intuitively, small-o is analogous to “ $<$ ”.

*Example.* Let  $f(n) = n$  and  $g(n) = n^2$ . We now show  $f = o(g)$  using above definition.  $\lim_{n \rightarrow \infty} f/g = \lim_{n \rightarrow \infty} n/n^2 = \lim_{n \rightarrow \infty} 1/n = 0$ . Hence,  $f = o(g)$ .

## Commonly-Used Functions in Algorithm Analysis

In theoretical computer science, we often see following categories of functions.

1. logarithmic functions:  $\log \log n, \log n, (\log n)^2$ ;
2. polynomial functions:  $\sqrt{n} = n^{0.5}, n, n \log n, n^{1.001}$ ;
3. exponential functions:  $2^n, n2^n, 3^n$ ;
4. factorial functions:  $n!$ ;

In above lists, any logarithmic function is small-o of any polynomial function: for example,  $(\log n)^2 = o(n^{0.01})$ ; any polynomial function is small-o of any exponential function: for example,  $n^2 = o(2^n)$ ; any exponential function is small-o of any factorial function: for example,  $n2^n = o(n!)$ . Within each category, a function to the left is small-o of a function to the right, for example  $n \log n = o(n^{1.001})$ .

## Merge-Sort

We now start introducing the first algorithm-design technique: divide-and-conquer. A typical divide-and-conquer algorithm follows the framework below.

1. partition the original problem into smaller problems;
2. recursively solve all subproblems;
3. combine the solutions of the subproblems to obtain the solution of the original problem.

We use sorting as the first problem to demonstrate designing divide-and-conquer algorithms. Recall that the *sorting* problem is to find the sorted array (say, in increasing order)  $S'$  of a given array  $S$ . We now design a divide-and-conquer algorithm for it. For any recursive algorithm, we always need to clearly define the recursion. In this case, we define function merge-sort ( $S$ ) returns the sorted array (in ascending order) of  $S$ .

The idea is to sort the first half and second half of  $S$ , by recursively call the merge-sort function. How to obtain the sorted array of  $S$  then with the two sorted half-sized arrays? We have introduced such an algorithm to merge two sorted arrays into a single sorted array. That's exactly the algorithm we need here in the combining step.

```
Algorithm merge-sort ( $S[1 \cdots n]$ )  
    if  $n \leq 1$ : return  $S$ ;  
     $S'_1 = \text{merge-sort}(S[1 \cdots n/2])$ ;  
     $S'_2 = \text{merge-sort}(S[n/2 + 1 \cdots n])$ ;  
    return merge-two-sorted-arrays ( $S'_1, S'_2$ );  
end algorithm;
```