

CS447 Computer Algorithms. Supplementary Notes A. Fall 2020.

Reference: Some of the material below can be found in “Introduction to Algorithms” by Cormen, Leiserson, Rivest, and Stein, The MIT Press.

Asymptotics We review some asymptotic notation useful for analyzing algorithms. The purpose is to generalize the comparisons such as $f \leq g$, $f \geq g$, $f = g$, $f < g$, and $f > g$ for cases when f and g are functions over the natural numbers.

Here are the common notations (or sets of functions).

1. **Big Oh:**

$$O(g(n)) = \{f(n) : (\exists c, n_0)(\forall n \geq n_0)[0 \leq f(n) \leq cg(n)]\}$$

2. **Big Omega:**

$$\Omega(g(n)) = \{f(n) : (\exists c, n_0)(\forall n \geq n_0)[0 \leq cg(n) \leq f(n)]\}$$

3. **Big Theta:**

$$\Theta(g(n)) = \{f(n) : f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))\}$$

4. **little oh:**

$$o(g(n)) = \left\{f(n) : \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0\right\}$$

5. **little omega:**

$$\omega(g(n)) = \left\{f(n) : \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty\right\}$$

What it states	What it “means”
$f(n) = O(g(n))$	$f \leq g$
$f(n) = \Omega(g(n))$	$f \geq g$
$f(n) = \Theta(g(n))$	$f = g$
$f(n) = o(g(n))$	$f < g$
$f(n) = \omega(g(n))$	$f > g$

Summations We review three important types of summations relevant to this course.

1. **Arithmetic sum:**

$$\sum_{k=1}^n k = \binom{n}{2} = \frac{n(n-1)}{2}.$$

The above formula can be proved using induction (as is done in MA211 Foundations).

2. **Geometric sum:** suppose $\alpha \neq 1$ is a positive real number. Then

$$\sum_{k=0}^{n-1} \alpha^k = \frac{\alpha^n - 1}{\alpha - 1}$$

The above formula can be proved using induction and apparently was known to Gauss when he was in elementary school.

3. **Harmonic sum:**

$$\sum_{k=1}^n \frac{1}{k} = \ln(n) + O(1).$$

This formula can be proved using integrals (from MA131 Calculus I).

Recurrences The analyses of recursive algorithms naturally lead to recurrences. Some of the basic ones arise from analyzing searching and sorting algorithms. Here is a well-known cookbook method that can handle *most* (but not all) of the common recurrences.

(Master Theorem) Suppose $T(n)$ is given by the recurrence

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is a function defined over the natural numbers.

1. **(Low case)** If $f(n) = O(n^{\log_b(a)-\epsilon})$ for some $\epsilon > 0$, then

$$T(n) = \Theta(n^{\log_b(a)}).$$

2. **(Middle case)** If $f(n) = \Theta(n^{\log_b(a)})$, then

$$T(n) = \Theta(n^{\log_b(a)} \log n).$$

3. **(High case)** If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some $\epsilon > 0$ and there is a constant $c < 1$ so that $af(n/b) < cf(n)$ for large enough n , then

$$T(n) = \Theta(f(n)).$$