Basics of Algorithm Analysis

- **We measure running time as a function of n, the size of the input (in bytes assuming a reasonable encoding).**
- **We work in the RAM model of computation. All "reasonable" operations take "1" unit of time. (e.g. +, *, -, /, array access, pointer following, writing a value, one byte of I/O...)**

What is the running time of an algorithm

- **Best case (seldom used)**
- **Average case (used if we understand the average)**
- **Worst case (used most often)**

We measure as a function of n, and ignore low order terms.

- $5n^3 + n - 6$ **becomes** n^3
- $8n \log n - 60n$ **becomes** $n \log n$
- $2^n + 3n^4$ **becomes** 2^n

Asymptotic notation

big-O

$$O(g(n)) = \{f(n) : \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}\ .$$

Alternatively, we say

$$f(n) = O(g(n)) \text{ if there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

Informally, $f(n) = O(g(n))$ **means that** $f(n)$ **is asymptotically less than or equal to** $g(n)$**.**

big-Ω

$$\Omega(g(n)) = \{f(n) : \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}\ .$$

Alternatively, we say

$$f(n) = \Omega(g(n)) \text{ if there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}\ .$$

Informally, $f(n) = \Omega(g(n)$ **means that** $f(n)$ **is asymptotically greater than or equal to** $g(n)$**.**

big-Θ

$f(n) = \Theta(g(n))$ **if and only if** $f(n) = O(g(n))$ **and** $f(n) = \Omega(g(n))$**.**

Informally, $f(n) = \Theta(g(n)$ **means that** $f(n)$ **is asymptotically equal to** $g(n)$**.**

INFORMAL summary

- $f(n) = O(g(n))$ **roughly means** $f(n) \leq g(n)$
- $f(n) = \Omega(g(n))$ **roughly means** $f(n) \geq g(n)$
- $f(n) = \Theta(g(n))$ **roughly means** $f(n) = g(n)$
- $f(n) = o(g(n))$ **roughly means** $f(n) < g(n)$
- $f(n) = w(g(n))$ **roughly means** $f(n) > g(n)$

We use these to classify algorithms into classes, e.g. n**,** n^2**,** $n \log n$**,** 2^n**.**

See chart for justification

3 useful formulas

Arithmetic series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Geometric series

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a} \quad \textbf{for } 0 < a < 1$$

Harmonic series

$$\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1) = \Theta(\ln n)$$

Algorithmic Correctness

- **Very important, but we won't typically prove correctness from first principles.**
- **We will use loop invariants**
- **We will use other problem specific methods**

MergeSort

$Merge - Sort(A, p, r)$

if $p < r$
    $q = \lfloor (p+r)/2 \rfloor$
    $\textsc{Merge-Sort}(A, p, q)$
    $\textsc{Merge-Sort}(A, q+1, r)$
    $\textsc{Merge}(A, p, q, r)$

Let $T(n)$ be the running time of MergeSort on n items. Merge takes $O(n)$ time.

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n = 1 \ , \\ 2T(n/2) + \Theta(n) & \textbf{if } n > 1 \ . \end{cases} \tag{1}$$

3 Recurrence Trees

1. $T(n) = 2T(n/2) + n$

2. $T(n) = 2T(n/2) + 1$

3. $T(n) = 2T(n/2) + n^2$

Master Theorem

Master Theorem for Recurrences **Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence**

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows.

1. **If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.**
2. **If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.**
3. **If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.**