CMSC 754: Short Reference Guide

This document contains a short summary of information about algorithm analysis and data structures, which may be useful later in the semester.

Asymptotic Forms: The following gives both the formal "c and n_0" definitions and an equivalent limit definition for the standard asymptotic forms. Assume that f and g are nonnegative functions.

Asymptotic Form	Relationship	Limit Form	Formal Definition
$f(n) \in \Theta(g(n))$	$f(n) \equiv g(n)$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$\exists c_1, c_2, n_0, \forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$.
$f(n) \in O(g(n))$	$f(n) \preceq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$\exists c, n_0, \forall n \geq n_0,\ 0 \leq f(n) \leq c g(n)$.
$f(n) \in \Omega(g(n))$	$f(n) \succeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$	$\exists c, n_0, \forall n \geq n_0,\ 0 \leq c g(n) \leq f(n)$.
$f(n) \in o(g(n))$	$f(n) \prec g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$	$\forall c, \exists n_0, \forall n \geq n_0,\ 0 \leq f(n) \leq c g(n)$.
$f(n) \in \omega(g(n))$	$f(n) \succ g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$\forall c, \exists n_0, \forall n \geq n_0,\ 0 \leq c g(n) \leq f(n)$.

Polylog-Polynomial-Exponential: For any constants a, b, and c, where $b > 0$ and $c > 1$.

$$\log^a n \prec n^b \prec c^n.$$

Common Summations: Let c be any constant, $c \neq 1$, and $n \geq 0$.

Name of Series	Formula	Closed-Form Solution	Asymptotic
Constant Series	$\sum_{i=a}^{b} 1$	$= \max(b - a + 1, 0)$	$\Theta(b - a)$
Arithmetic Series	$\sum_{i=0}^{n} i = 0 + 1 + 2 + \cdots + n$	$= \frac{n(n+1)}{2}$	$\Theta(n^2)$
Geometric Series	$\sum_{i=0}^{n} c^i = 1 + c + c^2 + \cdots + c^n$	$= \frac{c^{n+1} - 1}{c - 1}$	$\begin{cases} \Theta(c^n)\ (c > 1) \\ \Theta(1)\ (c < 1) \end{cases}$
Quadratic Series	$\sum_{i=0}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2$	$= \frac{2n^3 + 3n^2 + n}{6}$	$\Theta(n^3)$
Linear-geom. Series	$\sum_{i=0}^{n-1} ic^i = c + 2c^2 + 3c^3 \cdots + nc^n$	$= \frac{(n-1)c^{(n+1)} - nc^n + c}{(c-1)^2}$	$\Theta(nc^n)$
Harmonic Series	$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$	$\approx \ln n$	$\Theta(\log n)$

Recurrences: Recursive algorithms (especially those based on divide-and-conquer) can often be analyzed using the so-called *Master Theorem*, which states that given constants $a > 0$, $b > 1$, and $d \geq 0$, the function $T(n) = aT(n/b) + O(n^d)$, has the following asymptotic form:

$$T(n) \ = \ \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a. \end{cases}$$

Sorting: The following algorithms sort a set of n keys over a totally ordered domain. Let $[m]$ denote the set $\{0, \ldots, m\}$, and let $[m]^k$ denote the set of ordered k-tuples, where each element is taken from $[m]$. A sorting algorithm is *stable* if it preserves the relative order of equal elements. A sorting algorithm is *in-place* if it uses no additional array storage other than the input array (although $O(\log n)$ additional space is allowed for the recursion stack). The *comparison-based algorithms* (Insertion-, Merge-, Heap-, and QuickSort) operate under the general assumption that there is a *comparator function* $f(x, y)$ that takes two elements x and y and determines whether $x < y$, $x = y$, or $x > y$.

Algorithm	Domain	Time	Space	Stable	In-place
CountingSort	Integers $[m]$	$O(n + m)$	$O(n + m)$	Yes	No
RadixSort	Integers $[m]^k$ or $[m^k]$	$O(k(n + m))$	$O(kn + m)$	Yes	No
InsertionSort	Total order	$O(n^2)$	$O(n)$	Yes	Yes
MergeSort HeapSort QuickSort	Total order	$O(n \log n)$	$O(n)$	Yes No Yes/No*	No Yes No/Yes

*There are two versions of QuickSort, one which is stable but not in-place, and one which is in-place but not stable.

Order statistics: For any k, $1 \leq k \leq n$, the kth smallest element of a set of size n (over a totally ordered domain) can be computed in $O(n)$ time.

Useful Data Structures: All these data structures use $O(n)$ space to store n objects.

Unordered Dictionary: (by randomized hashing) Insert, delete, and find in $O(1)$ expected time each. (Note that you can find an element exactly, but you cannot quickly find its predecessor or successor.)

Ordered Dictionary: (by balanced binary trees or skiplists) Insert, delete, find, predecessor, successor, merge, split in $O(\log n)$ time each. (Merge means combining the contents of two dictionaries, where the elements of one dictionary are all smaller than the elements of the other. Split means splitting a dictionary into two about a given value x, where one dictionary contains all the items less than or equal to x and the other contains the items greater than x.) Given the location of an item x in the data structure, it is possible to locate a given element y in time $O(\log k)$, where k is the number of elements between x and y (inclusive).

Priority Queues: (by binary heaps) Insert, delete, extract-min, union, decrease-key, increase-key in $O(\log n)$ time. Find-min in $O(1)$ time each. Make-heap from n keys in $O(n)$ time.

Priority Queues: (by Fibonacci heaps) Any sequence of n insert, extract-min, union, decrease-key can be done in $O(1)$ amortized time each. (That is, the sequence takes $O(n)$ total time.) Extract-min and delete take $O(\log n)$ amortized time. Make-heap from n keys in $O(n)$ time.

Disjoint Set Union-Find: (by inverted trees with path compression) Union of two disjoint sets and find the set containing an element in $O(\log n)$ time each. A sequence of m operations can be done in $O(\alpha(m, n))$ amortized time. That is, the entire sequence can be done in $O(m \cdot \alpha(m, n))$ time. (α is the *extremely* slow growing inverse-Ackerman function.)

Orientation Testing: For any constant dimension d, given any ordered $(d + 1)$-tuple of points in $\mathbb{R}^d$, it can be determined in $O(1)$ time whether these points are (a) negatively oriented (clockwise), (b) positively oriented (counterclockwise) or (c) affinely dependent (collinear). This test can be applied for many other geometric predicates, such as determining whether two given line segments in the plane intersect, whether a given point lies within a given triangle, and whether a given point lies within the circumcircle of three other given points. (This will be discussed later in the semester.)