Quick Reference Guide

1. Asymptotic forms:

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

2. Polylogs, polynomials and exponentials - a and b are constants:

   (a) log bases do not matter $\quad log_a n \approx log_b n, a, b > 1$

   (b) exponential bases do matter $\quad a^n \prec b^n, 1 < a < b$

   (c) polynomials grow faster than polylogs $\quad log^a n \prec n^b, a, b > 0$

   (d) exponentails grow faster than polynomials $\quad n^a \prec b^n, a > 0, b > 1$

   Note that the convention is to write the a-th power of $logn$ as $log^a n$, instead of $(logn)^a$. And logs that don't have specified bases are assumed to be base 2, i.e. $logn \equiv log_2 n$.

3. Summations: c is a constant, $c \neq 1$, and $n \geq 0$

Series	Formula		Closed-form Solutions	Asymptotic
Constant series	$\sum_{i=a}^{b} 1$	# of elements btw a and b	$max(b-a+1, 0)$	$\Theta(b-a)$
Arithmetic series	$\sum_{i=0}^{n} i$	$0+1+2+...+n$	$\frac{n(n+1)}{2}$	$\Theta(n^2)$
Quadratic series	$\sum_{i=0}^{n} i^2$	$0+1+4+...+n^2$	$\frac{2n^3+3n^2+n}{6}$	$\Theta(n^3)$
Geometric series	$\sum_{i=0}^{n} c^i$	$1+c+c^2+...+c^n$	$\frac{c^{n+1}-1}{c-1}$	$\begin{cases} \Theta(c^n) & c > 1 \\ \Theta(1) & c < 1 \end{cases}$
Polynomial series	$\sum_{i=0}^{n} i^c$	$0+1+2^c+...+n^c$	-	$\Theta(n^{c+1})$
Linear-geom. series	$\sum_{i=0}^{n-1} ic^i$	$0+c+2c^2+...+(n-1)c^{(n-1)}$	$\frac{(n-1)c^{n+1}-nc^n+c}{(c-1)^2}$	$\Theta(nc^n)$
Harmonic series	$\sum_{i=1}^{n} \frac{1}{i}$	$1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n}$	$\approx ln\, n$	$\Theta(logn)$

4. Sorting: The following algorithms sort a set of n keys. We say that a sorting algorithm is *stable* if it preserves the relative order of elements with equal keys, and a sorting algorithm is *in-place* if it uses no additional storage other than the input.

Algorithm	Time	Space	Stable	In-place
InsertionSort /SelectionSort /BubbleSort	$O(n^2)$	$O(n)$	yes	yes
MergeSort	$O(nlogn)$	$O(n)$	yes	no
HeapSort	$O(nlogn)$	$O(n)$	no	yes
QuickSort (Hoare's or Lomuto's)	$O(nlogn)$	$O(n)$	no	yes
Stable QuickSort	$O(nlogn)$	$O(n)$	yes	no

5. Data Structures

Basic data structures and operational times

	insert	delete	nth	find	findMin	deleteMin	changeKey
Unordered Array	$O(1)^+$	$O(1)^*$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
Ordered Array	$O(n)$	$O(n)$	$O(1)$	$O(logn)$	$O(1)$	$O(1)^*$	$O(n)$
Unordered LL	$O(1)$	$O(1)^\dagger$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)^\dagger$
Ordered LL	$O(n)$	$O(1)^\dagger$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Balanced BST	$O(logn)$	$O(logn)$	$O(logn)$	$O(logn)$	$O(logn)$	$O(logn)$	$O(logn)$
Binary Heap	$O(logn)$	$O(logn)$	$O(n)$	$O(n)$	$O(1)$	$O(logn)$	$O(logn)$
Hash Table	$O(1)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$

Notes:

$+$ insert at the end of array

***** delete without shifting (mark with -1 or NULL)

† when given reference/pointer to element, i.e. no additional find

High-level data structures

(a) Unordered Dictionary (randomized hashing): insert, delete and find in $O(1)$ expected. You can quickly find an element exactly, but you cannot quickly find its predecessor or successor. Any ordered access will cost $O(n)$.

(b) Ordered Dictionary (balanced BST): insert, delete, find, predecessor, successor, min/max in $O(logn)$. Given the location of an element x in the data structure, it is possible to locate a given element y in time $O(logk)$, where k is the number of elements between x and y (inclusive).

(c) Priority Queue (binary heap): insert, delete, deleteMin, union, changeKey in $O(logn)$. findMin in $O(1)$. Heap construction of n keys cost $O(n)$.

(d) Union-Find (inverted trees with path compression): union and find costs $O(logn)$ amortized.