# **Algorithms and Data Structures Cheatsheet**

**☑**CODING BUGS **☑**NOTES GALLERY

We summarize the performance characteristics of classic algorithms and data structures for sorting, priority queues, symbol tables, and graph processing.

We also summarize some of the mathematics useful in the analysis of algorithms, including commonly encountered functions; useful formulas and approximations; properties of logarithms; asymptotic notations; and solutions to divide-and-conquer recurrences.

#### Sorting.

The table below summarizes the number of compares for a variety of sorting algorithms, as implemented in this textbook. It includes leading constants but ignores lower-order terms.

ALGORITHM	CODE	IN PLACE	STABLE	BEST	AVERAGE	WORST	REMARKS
selection sort	Selection.java	✓		$\frac{1}{2}n^2$	$\frac{1}{2} n^2$	$\frac{1}{2}$ $n^2$	<i>n</i> exchanges; quadratic in best case
insertion sort	Insertion.java	✓	<b>√</b>	n	$\frac{1}{4}n^2$	$\frac{1}{2}n^2$	use for small or partially-sorted arrays
bubble sort	Bubble.java	✓	✓	n	$\frac{1}{2} n^2$	$\frac{1}{2}$ $n^2$	rarely useful; use insertion sort instead
shellsort	Shell.java	✓		$n \log_3 n$	unknown	$c n^{3/2}$	tight code; subquadratic
mergesort	Merge.java		✓	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	<i>n</i> log <i>n</i> guarantee; stable
quicksort	Quick.java	✓		$n \lg n$	$2 n \ln n$	$\frac{1}{2}$ $n^2$	<i>n</i> log <i>n</i> probabilistic guarantee; fastest in practice
heapsort	<u>Heap.java</u>	✓		$n^{\dagger}$	2 <i>n</i> lg <i>n</i>	$2 n \lg n$	n log n guarantee; in place

 $<sup>^{\</sup>dagger}$  n lg n if all keys are distinct

### Priority queues.

The table below summarizes the order of growth of the running time of operations for a variety of priority queues, as implemented in this textbook. It ignores leading constants and lower-order terms. Except as noted, all running times are worst-case running times.

DATA STRUCTURE	CODE	INSERT	<b>DEL-MIN</b>	MIN	<b>DEC-KEY</b>	<b>DELETE</b>	MERGE
array	BruteIndexMinPQ.java	1	n	n	1	1	n
binary heap	IndexMinPQ.java	$\log n$	$\log n$	1	$\log n$	$\log n$	n
d-way heap	IndexMultiwayMinPQ.java	$\log_d n$	$d \log_d n$	1	$\log_d n$	$d \log_d n$	n
binomial heap	IndexBinomialMinPQ.java	1	$\log n$	1	$\log n$	$\log n$	$\log n$
Fibonacci heap	IndexFibonacciMinPQ.java	1	$\log n^{\dagger}$	1	1 <sup>†</sup>	$\log n^{\dagger}$	1
						+	

<sup>†</sup> amortized guarantee

#### Symbol tables.

The table below summarizes the order of growth of the running time of operations for a variety of symbol tables, as implemented in this textbook. It ignores leading constants and lower-order terms.

			worst case		;	average case	,
DATA STRUCTURE	CODE	SEARCH	INSERT	DELETE	SEARCH	INSERT	DELETE
sequential search (in an unordered list)	SequentialSearchST, java	n	n	n	n	n	n
<b>binary search</b> (in a sorted array)	BinarySearchST.java	$\log n$	n	n	$\log n$	n	n
binary search tree (unbalanced)	BST.java	n	n	n	log n	$\log n$	sqrt(n)
red-black BST (left-leaning)	RedBlackBST.java	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$
AVL	AVLTreeST.java	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$

hash table (separate-chaining)	<u>SeparateChainingHashST.java</u>	n	n	n	1 <sup>†</sup>	1 †	1 †
hash table (linear-probing)	LinearProbingHashST.java	n	n	n	1 †	1 †	1 <sup>†</sup>
					† uniform hasł	ning assump	otion

# Graph processing.

The table below summarizes the order of growth of the worst-case running time and memory usage (beyond the memory for the graph itself) for a variety of graph-processing problems, as implemented in this textbook. It ignores leading constants and lower-order terms. All running times are worst-case running times.

PROBLEM	ALGORITHM	CODE	TIME	SPACE
path	DFS	DepthFirstPaths.java	E + V	V
shortest path (fewest edges)	BFS	BreadthFirstPaths.java	E + V	V
cycle	DFS	Cycle.java	E + V	V
directed path	DFS	<u>DepthFirstDirectedPaths.java</u>	E + V	V
shortest directed path (fewest edges)	BFS	BreadthFirstDirectedPaths.java	E + V	V
directed cycle	DFS	<u>DirectedCycle.java</u>	E + V	V
topological sort	DFS	<u>Topological.java</u>	E + V	V
bipartiteness / odd cycle	DFS	Bipartite.java	E + V	V
connected components	DFS	CC.java	E + V	V
strong components	Kosaraju–Sharir	KosarajuSharirSCC.java	E + V	V
strong components	Tarjan	TarjanSCC.java	E + V	V
strong components	Gabow	GabowSCC.java	E + V	V
Eulerian cycle	DFS	EulerianCycle.java	E + V	E + V
directed Eulerian cycle	DFS	<u>DirectedEulerianCycle.java</u>	E + V	V
transitive closure	DFS	TransitiveClosure.java	V(E+V)	$V^2$
minimum spanning tree	Kruskal	KruskalMST, java	$E \log E$	E + V
minimum spanning tree	Prim	PrimMST.java	$E \log V$	V
minimum spanning tree	Boruvka	BoruvkaMST.java	$E \log V$	V
shortest paths (nonnegative weights)	Dijkstra	<u>DijkstraSP.java</u>	$E \log V$	V
shortest paths (no negative cycles)	Bellman-Ford	BellmanFordSP.java	V(V+E)	V
shortest paths (no cycles)	topological sort	AcyclicSP.java	V + E	V
all-pairs shortest paths	Floyd–Warshall	FloydWarshall.java	$V^3$	$V^2$
maxflow-mincut	Ford-Fulkerson	FordFulkerson.java	E V (E + V)	V
bipartite matching	Hopcroft-Karp	HopcroftKarp.java	$V^{\frac{1}{2}}\left(E+V\right)$	V
assignment problem	successive shortest paths	AssignmentProblem.java	$n^3 \log n$	$n^2$

# Commonly encountered functions.

Here are some functions that are commonly encountered when analyzing algorithms.

<b>FUNCTION</b>	<b>NOTATION</b>	DEFINITION
floor	$\lfloor x \rfloor$	greatest integer $\leq x$
ceiling	$\lceil x  ceil$	smallest integer $\geq x$
binary logarithm	$\lg x$ or $\log_2 x$	$y$ such that $2^{y}=x$
natural logarithm	$\ln x$ or $\log_e x$	$y$ such that $e^{y}=x$
common logarithm	$\log_{10} x$	$y$ such that $10^{\:y}=x$
iterated binary logarithm	$\lg^* x$	$0$ if $x \leq 1; \; 1 + \lg^*(\lg x)$ otherwise
harmonic number	$H_n$	$1+1/2+1/3+\ldots+1/n$
factorial	n!	$1\times 2\times 3\times \ldots \times n$

#### Useful formulas and approximations.

Here are some useful formulas for approximations that are widely used in the analysis of algorithms.

- ullet Harmonic sum:  $1+1/2+1/3+\ldots+1/n\sim \ln n$
- ullet Triangular sum:  $1+2+3+\ldots+n=n\left(n+1
  ight)/\left(2-n^2\right)/\left(2-n^2\right)$
- ullet Sum of squares:  $1^2+2^2+3^2+\ldots+n^2\sim n^3$  / 3
- ullet Geometric sum: If r 
  eq 1 , then  $1+r+r^2+r^3+\ldots+r^n=(r^{n+1}-1)\ /\ (r-1)$

$$\circ \ r = 1/2 \colon \ 1 + 1/2 + 1/4 + 1/8 + \ldots + 1/2^n \sim 2$$

$$\circ \ r=2$$
:  $\ 1+2+4+8+\ldots+n/2+n=2n-1\sim 2n$  , when  $n$  is a power of 2

- Stirling's approximation:  $\lg(n!) = \lg 1 + \lg 2 + \lg 3 + \ldots + \lg n \sim n \lg n$
- Exponential:  $(1+1/n)^n \sim e;$   $(1-1/n)^n \sim 1/e$
- ullet Binomial coefficients:  $inom{n}{k} \sim n^k \, / \, k!$  when k is a small constant
- Approximate sum by integral: If f(x) is a monotonically increasing function, then  $\int_0^n f(x) \ dx \le \sum_{i=1}^n \ f(i) \le \int_1^{n+1} f(x) \ dx$

#### Properties of logarithms.

- Definition:  $\log_b a = c$  means  $b^c = a$ . We refer to b as the base of the logarithm.
- Special cases:  $\log_b b = 1, \, \log_b 1 = 0$
- Inverse of exponential:  $b^{\log_b x} = x$
- Product:  $\log_b(x \times y) = \log_b x + \log_b y$
- Division:  $\log_b(x \div y) = \log_b x \log_b y$
- Finite product:  $\log_b(x_1 imes x_2 imes \ldots imes x_n) = \log_b x_1 + \log_b x_2 + \ldots + \log_b x_n$
- ullet Changing bases:  $\log_b x = \log_c x \ / \ \log_c b$
- Rearranging exponents:  $x^{\log_b y} = y^{\log_b x}$
- Exponentiation:  $\log_b(x^y) = y \log_b x$

#### Asymptotic notations: definitions.

NAME	NOTATION	DESCRIPTION	DEFINITION
Tilde	$f(n) \sim g(n)$	f(n) is equal to $g(n)$ asymptotically (including constant factors)	$\lim_{n\to\infty}\frac{f(n)}{g(n)}=1$
Big Oh	f(n) is $O(g(n))$	f(n) is bounded above by $g(n)$ asymptotically (ignoring constant factors)	there exist constants $c>0$ and $n_0\geq 0$ such that $0\leq f(n)\leq c\cdot g(n)$ for all $n\geq n_0$
Big Omega	$f(n)$ is $\Omega(g(n))$	f(n) is bounded below by $g(n)$ asymptotically (ignoring constant factors)	g(n) is $O(f(n))$
Big Theta	$f(n)$ is $\Theta(g(n))$	f(n) is bounded above and below by $g(n)$ asymptotically (ignoring constant factors)	$f(n)$ is both $O(g(n))$ and $\Omega(g(n))$
Little oh	f(n) is $o(g(n))$	f(n) is dominated by $g(n)$ asymptotically (ignoring constant factors)	$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$
Little omega	$f(n)$ is $\omega(g(n))$	f(n) dominates $g(n)$ asymptotically (ignoring constant factors)	g(n) is $o(f(n))$

(ignoring constant factors)

#### Common orders of growth.

NAME	NOTATION	EXAMPLE	CODE FRAGMENT
Constant	O(1)	array access arithmetic operation function call	op();
Logarithmic	$O(\log n)$	binary search in a sorted array insert in a binary heap search in a red-black tree	<pre>for (int i = 1; i &lt;= n; i = 2*i)     op();</pre>
Linear	O(n)	sequential search grade-school addition BFPRT median finding	<pre>for (int i = 0; i &lt; n; i++)   op();</pre>
Linearithmic	$O(n \log n)$	mergesort heapsort fast Fourier transform	<pre>for (int i = 1; i &lt;= n; i++)   for (int j = i; j &lt;= n; j = 2*j)     op();</pre>
Quadratic	$O(n^2)$	enumerate all pairs insertion sort grade-school multiplication	<pre>for (int i = 0; i &lt; n; i++)   for (int j = i+1; j &lt; n; j++)     op();</pre>
Cubic	$O(n^3)$	enumerate all triples Floyd–Warshall grade-school matrix multiplication	<pre>for (int i = 0; i &lt; n; i++)   for (int j = i+1; j &lt; n; j++)     for (int k = j+1; k &lt; n; k++)         op();</pre>
Polynomial	$O(n^c)$	ellipsoid algorithm for LP AKS primality algorithm Edmond's matching algorithm	
Exponential	$2^{O(n^c)}$	enumerating all subsets enumerating all permutations backtracking search	

#### Asymptotic notations: properties.

- Reflexivity: f(n) is O(f(n)).
- Constants: If f(n) is O(g(n)) and c>0, then  $c\cdot f(n)$  is O(g(n)).
- Products: If  $f_1(n)$  is  $O(g_1(n))$  and  $f_2(n)$  is  $O(g_2(n))$ , then  $f_1(n) \cdot f_2(n)$  is  $O(g_1(n) \cdot g_2(n))$ .
- ullet Sums: If  $f_1(n)$  is  $O(g_1(n))$  and  $f_2(n)$  is  $O(g_2(n))$ , then  $f_1(n)+f_2(n)$  is  $O(\max\{g_1(n),g_2(n)\})$ .
- Transitivity: If f(n) is O(g(n)) and g(n) is O(h(n)), then f(n) is O(h(n)).
- ullet Polynomials: Let  $f(n)=a_0+a_1n+\ldots+a_dn^d$  with  $a_d>0$  . Then, f(n) is  $\Theta(n^d)$  .
- Logarithms and polynomials:  $\log_b n$  is  $O(n^d)$  for every b>0 and every d>0.
- ullet Exponentials and polynomials:  $\, \, n^d \,$  is  $\, O(r^n) \,$  for every  $r>0 \,$  and every  $d>0 \,$  .
- Factorials: n! is  $2^{\Theta(n \log n)}$ .
- Limits: If  $\lim_{n o \infty} rac{f(n)}{g(n)} = c$  for some constant  $0 < c < \infty$  , then f(n) is  $\Theta(g(n))$  .
- Limits: If  $\lim_{n o\infty}rac{f(n)}{g(n)}=0$ , then f(n) is O(g(n)) but not  $\Theta(g(n))$ .
- Limits: If  $\lim_{n o\infty}rac{f(n)}{g(n)}=\infty$  , then f(n) is  $\Omega(g(n))$  but not O(g(n)) .

Here are some examples.

FUNCTION 
$$o(n^2)$$
  $O(n^2)$   $\Theta(n^2)$   $\Omega(n^2)$   $\omega(n^2)$   $\sim 2n^2$   $\sim 4n^2$   $\log_2 n$  , , ,  $\checkmark$  ,  $10n+45$  ,  $\checkmark$  ,  $\checkmark$  ,  $\checkmark$  ,  $\checkmark$ 

## Divide-and-conquer recurrences.

For each of the following recurrences we assume T(1)=0 and that  $n \mathbin{/} 2$  means either  $\lfloor n \mathbin{/} 2 \rfloor$  or  $\lceil n \mathbin{/} 2 \rceil$  .

T(n)	<b>EXAMPLE</b>
$\sim \lg n$	binary search
$\sim n \lg n$	mergesort
$\sim rac{1}{2} n^2$	insertion sort
$\sim n$	tree traversal
$\sim 2^n$	towers of Hanoi
$\Theta(n^{\log_2 3}) = \Theta(n^{1.58})$	Karatsuba multiplication
$\Theta(n^{\log_2 7}) = \Theta(n^{2.81})$	Strassen multiplication
$\Theta(n\log^2 n)$	closest pair
	$egin{align} \sim \lg n \ &\sim n\lg n \ &\sim rac{1}{2}n^2 \ &\sim n \ &\sim 2^n \ &\Theta(n^{\log_2 3}) = \Theta(n^{1.58}) \ &\Theta(n^{\log_2 7}) = \Theta(n^{2.81}) \ \end{array}$

#### Master theorem.

Let  $a \geq 1$  ,  $b \geq 2$  , and c > 0 and suppose that T(n) is a function on the non-negative integers that satisfies the divide-and-conquer recurrence

$$T(n) = a \; T(n \, / \, b) + \Theta(n^c)$$

with T(0)=0 and  $T(1)=\Theta(1)$  , where  $n \mathbin{/} b$  means either  $\lfloor n \mathbin{/} b \rfloor$  or either  $\lceil n \mathbin{/} b \rceil$  .

- If  $c < \log_b a$  , then  $T(n) = \Theta(n^{\log_b a})$
- ullet If  $c=\log_b a$ , then  $T(n)=\Theta(n^c\log n)$ ullet If  $c>\log_b a$ , then  $T(n)=\Theta(n^c)$

#### **☑**CODING BUGS **☑**NOTES GALLERY