Algorithms and Data Structures Cheatsheet

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	✓		$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges; quadratic in best case
insertion sort	Insertion.java	✓	✓	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		✓	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable
quicksort	Quick.java	✓		$n \lg n$	$2 n \ln n$	$\frac{1}{2} n^2$	$n \log n$ probabilistic guarantee; fastest in practice
heapsort	Heap.java	✓		n †	$2 n \lg n$	$2 n \lg n$	$n \log n$ guarantee; in place

† $n \lg n$ 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	DEL-MIN	MIN	DEC-KEY	DELETE	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	$\log n$ †	1	1 †	$\log n$ †	1

† 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.

DATA STRUCTURE	CODE	worst case SEARCH	worst case INSERT	worst case DELETE	average case SEARCH	average case INSERT	average case DELETE
sequential search (in an unordered list)	SequentialSearchST.java	n	n	n	n	n	n
binary search (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)	SeparateChainingHashST.java	n	n	n	1 †	1 †	1 †
hash table (linear-probing)	LinearProbingHashST.java	n	n	n	1 †	1 †	1 †

† uniform hashing assumption

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	DepthFirstDirectedPaths.java	$E + V$	V
shortest directed path (fewest edges)	BFS	BreadthFirstDirectedPaths.java	$E + V$	V
directed cycle	DFS	DirectedCycle.java	$E + V$	V
topological sort	DFS	Topological.java	$E + V$	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	DirectedEulerianCycle.java	$E + V$	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	Dijkstra	DijkstraSP.java	$E \log V$	V

weights)				
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]()	$V^{1/2}(E+V)$	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.

FUNCTION	NOTATION	DEFINITION
floor	$\lfloor x \rfloor$	greatest integer $\leq x$
ceiling	$\lceil x \rceil$	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	$n!$	$1 \times 2 \times 3 \times \ldots \times n$
binomial coefficient	$\binom{n}{k}$	$\frac{n!}{k!\,(n-k)!}$

Useful formulas and approximations.

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

- *Harmonic sum:* $1 + 1/2 + 1/3 + \ldots + 1/n \sim \ln n$
- *Triangular sum:* $1 + 2 + 3 + \ldots + n = n(n+1)/2 \sim n^2/2$
- *Sum of squares:* $1^2 + 2^2 + 3^2 + \ldots + n^2 \sim n^3/3$
- *Geometric sum:* If $r \neq 1$, then $1 + r + r^2 + r^3 + \ldots + r^n = (r^{n+1} - 1)/(r-1)$
  - $r = 1/2$: $1 + 1/2 + 1/4 + 1/8 + \ldots + 1/2^n \sim 2$
  - $r = 2$: $1 + 2 + 4 + 8 + \ldots + n/2 + n = 2n - 1 \sim 2n$, 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:* $(1 + 1/n)^n \sim e$; $(1 - 1/n)^n \sim 1/e$
- *Binomial coefficients:* $\binom{n}{k} \sim n^k / k!$ when k is a small constant

- *Approximate sum by integral:* If $f(x)$ f(x) is a monotonically increasing function, then
$\int_0^n f(x)\,dx \le \sum_{i=1}^{n} f(i) \le \int_1^{n+1} f(x)\,dx$ ∫0nf(x)dx≤∑i=1nf(i)≤∫1n+1f(x)dx

Properties of logarithms.

- *Definition:* $\log_b a = c$ logba=c means $b^c = a$ bc=a. We refer to bb as the *base* of the logarithm.
- *Special cases:* $\log_b b = 1,\ \log_b 1 = 0$ logbb=1,logb1=0
- *Inverse of exponential:* $b^{\log_b x} = x$ blogbx=x
- *Product:* $\log_b(x \times y) = \log_b x + \log_b y$ logb(x×y)=logbx+logby
- *Division:* $\log_b(x \div y) = \log_b x - \log_b y$ logb(x÷y)=logbx−logby
- *Finite product:*
$\log_b(x_1 \times x_2 \times \ldots \times x_n) = \log_b x_1 + \log_b x_2 + \ldots + \log_b x_n$ logb(x1×x2×…×xn)=logbx1+logbx2+…+logbxn
- *Changing bases:* $\log_b x = \log_c x \ / \ \log_c b$ logbx=logcx/logcb
- *Rearranging exponents:* $x^{\log_b y} = y^{\log_b x}$ xlogby=ylogbx
- *Exponentiation:* $\log_b(x^y) = y \log_b x$ logb(xy)=ylogbx

Asymptotic notations: definitions.

NAME	NOTATION	DESCRIPTION	DEFINITION
Tilde	$f(n) \sim g(n)$ f(n)~g(n)	$f(n)$ f(n) is equal to $g(n)$ g(n) asymptotically (including constant factors)	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ limn→∞f(n)g(n)=1
Big Oh	$f(n)$ f(n) is $O(g(n))$ O(g(n))	$f(n)$ f(n) is bounded above by $g(n)$ g(n) asymptotically (ignoring constant factors)	there exist constants $c > 0$ c>0 and $n_0 \ge 0$ n0≥0 such that $0 \le f(n) \le c \cdot g(n)$ 0≤f(n)≤c·g(n) for all $n \ge n_0$ n≥n0
Big Omega	$f(n)$ f(n) is $\Omega(g(n))$ Ω(g(n))	$f(n)$ f(n) is bounded below by $g(n)$ g(n) asymptotically (ignoring constant factors)	$g(n)$ g(n) is $O(f(n))$ O(f(n))
Big Theta	$f(n)$ f(n) is $\Theta(g(n))$ Θ(g(n))	$f(n)$ f(n) is bounded above and below by $g(n)$ g(n) asymptotically (ignoring constant factors)	$f(n)$ f(n) is both $O(g(n))$ O(g(n)) and $\Omega(g(n))$ Ω(g(n))
Little oh	$f(n)$ f(n) is $o(g(n))$ o(g(n))	$f(n)$ f(n) is dominated by $g(n)$ g(n) asymptotically (ignoring constant factors)	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ limn→∞f(n)g(n)=0
Little omega	$f(n)$ f(n) is $\omega(g(n))$ ω(g(n))	$f(n)$ f(n) dominates $g(n)$ g(n) asymptotically (ignoring constant factors)	$g(n)$ g(n) is $o(f(n))$ o(f(n))

Common orders of growth.

NAME	NOTATION	EXAMPLE	CODE FRAGMENT
Constant	$O(1)$ O(1)	array access arithmetic operation	`op();`

		function call	
Logarithmic	$O(\log n)$ O(logn)	binary search in a sorted array insert in a binary heap search in a red–black tree	`for (int i = 1; i <= n; i = 2*i)` `   op();`
Linear	$O(n)$ O(n)	sequential search grade-school addition BFPRT median finding	`for (int i = 0; i < n; i++)` `   op();`

Linearithmic	$O(n \log n)$ O(nlogn)	mergesort
		heapsort
		fast Fourier transform
			`for (int i = 1; i <= n; i++)`
			`   for (int j = i; j <= n; j = 2*j)`
			`      op();`

Quadratic	$O(n^2)$ O(n2)	enumerate all pairs
		insertion sort
		grade-school multiplication
			`for (int i = 0; i < n; i++)`
			`   for (int j = i+1; j < n; j++)`
			`      op();`

Cubic	$O(n^3)$ O(n3)	enumerate all triples
		Floyd–Warshall
		grade-school matrix multiplication
			`for (int i = 0; i < n; i++)`
			`   for (int j = i+1; j < n; j++)`
			`      for (int k = j+1; k < n; k++)`
			`         op();`

Polynomial	$O(n^c)$ O(nc)	ellipsoid algorithm for LP AKS primality algorithm Edmond's matching algorithm
Exponential	$2^{O(n^c)}$ 2O(nc)	enumerating all subsets enumerating all permutations backtracking search

Asymptotic notations: properties.

- *Reflexivity:* $f(n)$ f(n) is $O(f(n))$ O(f(n)).
- *Constants:* If $f(n)$ f(n) is $O(g(n))$ O(g(n)) and $c > 0$ c>0, then $c \cdot f(n)$ c·f(n) is $O(g(n)))$ O(g(n))).
- *Products:* If $f_1(n)$ f1(n) is $O(g_1(n))$ O(g1(n)) and $f_2(n)$ f2(n) is $O(g_2(n)))$ O(g2(n))), then $f_1(n) \cdot f_2(n)$ f1(n)·f2(n) is $O(g_1(n) \cdot g_2(n)))$ O(g1(n)·g2(n))).
- *Sums:* If $f_1(n)$ f1(n) is $O(g_1(n))$ O(g1(n)) and $f_2(n)$ f2(n) is $O(g_2(n)))$ O(g2(n))), then $f_1(n) + f_2(n)$ f1(n)+f2(n) is $O(\max\{g_1(n), g_2(n)\})$ O(max{g1(n),g2(n)}).
- *Transitivity:* If $f(n)$ f(n) is $O(g(n))$ O(g(n)) and $g(n)$ g(n) is $O(h(n))$ O(h(n)), then $f(n)$ f(n) is $O(h(n))$ O(h(n)).
- *Polynomials:* Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ f(n)=a0+a1n+…+adnd with $a_d > 0$ ad>0. Then, $f(n)$ f(n) is $\Theta(n^d)$ Θ(nd).
- *Logarithms and polynomials:* $\log_b n$ logbn is $O(n^d)$ O(nd) for every $b > 0$ b>0 and every $d > 0$ d>0.
- *Exponentials and polynomials:* n^d nd is $O(r^n)$ O(rn) for every $r > 0$ r>0 and every $d > 0$ d>0.
- *Factorials:* $n!$ n! is $2^{\Theta(n \log n)}$ 2Θ(nlogn).
- *Limits:* If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ limn→∞f(n)g(n)=c for some constant $0 < c < \infty$ 0<c<∞, then $f(n)$ f(n) is $\Theta(g(n))$ Θ(g(n)).
- *Limits:* If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ limn→∞f(n)g(n)=0, then $f(n)$ f(n) is $O(g(n))$ O(g(n)) but not $\Theta(g(n))$ Θ(g(n)).
- *Limits:* If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$ limn→∞f(n)g(n)=∞, then $f(n)$ f(n) is $\Omega(g(n))$ Ω(g(n)) but not $O(g(n))$ O(g(n)).

Here are some examples.

FUNCTION	$o(n^2)$	$O(n^2)$	$\Theta(n^2)$	$\Omega(n^2)$	$\omega(n^2)$	$\sim 2n^2$	$\sim 4n^2$
$\log_2 n$	✓	✓					
$10n + 45$	✓	✓					
$2n^2 + 45n + 12$		✓	✓	✓		✓	
$4n^2 - 2\sqrt{n}$		✓	✓	✓			✓
$3n^3$				✓	✓		
2^n				✓	✓		

Divide-and-conquer recurrences.

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

RECURRENCE	$T(n)$	EXAMPLE
$T(n) = T(n/2) + 1$	$\sim \lg n$	*binary search*
$T(n) = 2T(n/2) + n$	$\sim n \lg n$	*mergesort*
$T(n) = T(n-1) + n$	$\sim \frac{1}{2} n^2$	*insertion sort*
$T(n) = 2T(n/2) + 1$	$\sim n$	*tree traversal*
$T(n) = 2T(n-1) + 1$	$\sim 2^n$	*towers of Hanoi*
$T(n) = 3T(n/2) + \Theta(n)$	$\Theta(n^{\log_2 3}) = \Theta(n^{1.58...})$	*Karatsuba multiplication*
$T(n) = 7T(n/2) + \Theta(n^2)$	$\Theta(n^{\log_2 7}) = \Theta(n^{2.81...})$	*Strassen multiplication*
$T(n) = 2T(n/2) + \Theta(n \log n)$	$\Theta(n \log^2 n)$	*closest pair*

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 / b means either $\lfloor n/b \rfloor$ or either $\lceil n/b \rceil$.

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

Remark: there are many different versions of the master theorem. The Akra–Bazzi theorem is among the most powerful.

Last modified on July 05, 2021.