



hash table (separate-chaining)	<a href="#">SeparateChainingHashST.java</a>	$n$	$n$	$n$	$1^\dagger$	$1^\dagger$	$1^\dagger$
hash table (linear-probing)	<a href="#">LinearProbingHashST.java</a>	$n$	$n$	$n$	$1^\dagger$	$1^\dagger$	$1^\dagger$

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

## 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	$H_n$	$1 + 1/2 + 1/3 + \dots + 1/n$
factorial	$n!$	$1 \times 2 \times 3 \times \dots \times n$

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:*  $1 + 1/2 + 1/3 + \dots + 1/n \sim \ln n$
- Triangular sum:*  $1 + 2 + 3 + \dots + n = n(n+1)/2 \sim n^2/2$
- Sum of squares:*  $1^2 + 2^2 + 3^2 + \dots + n^2 \sim n^3/3$
- Geometric sum:* If  $r \neq 1$ , then  $1 + r + r^2 + r^3 + \dots + r^n = (r^{n+1} - 1) / (r - 1)$ 
  - $r = 1/2$ :  $1 + 1/2 + 1/4 + 1/8 + \dots + 1/2^n \sim 2$
  - $r = 2$ :  $1 + 2 + 4 + 8 + \dots + n/2 + n = 2n - 1 \sim 2n$  , when  $n$  is a power of 2
- Stirling's approximation:*  $\lg(n!) = \lg 1 + \lg 2 + \lg 3 + \dots + \lg n \sim n \lg n$
- 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  $f(x)$  is a monotonically increasing function, then  $\int_0^n f(x) \, dx \leq \sum_{i=1}^n f(i) \leq \int_1^{n+1} f(x) \, dx$

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:*  $\log_b(x_1 \times x_2 \times \dots \times x_n) = \log_b x_1 + \log_b x_2 + \dots + \log_b x_n$
- 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)	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$
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)	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
Little omega	$f(n)$ is $\omega(g(n))$	$f(n)$ dominates $g(n)$ asymptotically (ignoring constant factors)	$g(n)$ is $o(f(n))$

Common orders of growth.

NAME	NOTATION	EXAMPLE	CODE FRAGMENT
Constant	$O(1)$	array access arithmetic operation function call	<code>op();</code>
Logarithmic	$O(\log n)$	binary search in a sorted array insert in a binary heap search in a red–black tree	<code>for (int i = 1; i &lt;= n; i = 2*i)   op();</code>
Linear	$O(n)$	sequential search grade-school addition BFPRM median finding	<code>for (int i = 0; i &lt; n; i++)   op();</code>
Linearithmic	$O(n \log n)$	mergesort heapsort fast Fourier transform	<code>for (int i = 1; i &lt;= n; i++)   for (int j = i; j &lt;= n; j = 2*j)     op();</code>
Quadratic	$O(n^2)$	enumerate all pairs insertion sort grade-school multiplication	<code>for (int i = 0; i &lt; n; i++)   for (int j = i+1; j &lt; n; j++)     op();</code>
Cubic	$O(n^3)$	enumerate all triples Floyd–Warshall grade-school matrix multiplication	<code>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();</code>
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  $O(g_1(n) \cdot g_2(n))$ .
- 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  $O(h(n))$ .
- Polynomials:* Let  $f(n) = a_0 + a_1n + \dots + a_dn^d$  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  $d > 0$ .
- 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  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$  for some constant  $0 < c < \infty$ , then  $f(n)$  is  $\Theta(g(n))$ .
- Limits:* If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ , then  $f(n)$  is  $O(g(n))$  but not  $\Theta(g(n))$ .
- Limits:* If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ , then  $f(n)$  is  $\Omega(g(n))$  but not  $O(g(n))$ .

Here are some examples.

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

$4n^2 - 2\sqrt{n}$	✓	✓	✓	✓
$3n^3$			✓	✓
$2^n$			✓	✓

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

## 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  $T(n) = \Theta(n^c)$