Tilde Notation - Highest Order Term - Worst case usually

$3N^3 + 2N^2 + 1 \Rightarrow \sim 3N^3$

Asymptotic Notation - Order of Growth - Worst case usually

$3N^3 + 2N^2 + 1 \Rightarrow \Theta N^3$

Amortised - Average cost per operation in worst case - Worst case ÷ Number of operations

notation	provides	example	shorthand for	used to
Tilde	leading term	$\sim 10 N^2$	$10 N^2$ $10 N^2 + 22 N \log N$ $10 N^2 + 2 N + 37$	provide approximate model
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$	develop lower bounds

Linked List	
Add to front	O(1)
Add to back	O(1), improves upon single linked list's O(n)
Get at index	O(n), still need to walk the list but can walk from the back if the index is in the back half of the list
Remove at index	O(n), same rationale as get
Remove from front or back	O(1), same reasoning as add to front and add to back.

Heap	Average	Worst Case
Space	O(n)	O(n)
Search	O(n)	O(n)
Insert	O(1)	O(log n)
Delete	O(log n)	O(log n)
Peek	O(1)	O(1)

Search Table	Average Case	Guaranteed
Search	N/2	N
Insert	N	N

Priority Queues

implementation	time	space
sort	$N \log N$	N
elementary PQ	$M N$	M
binary heap	$N \log M$	M
best in theory	N	M

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
goal	$\log N$	$\log N$	$\log N$

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
binary heap	$\log N$	$\log N$	1
d-ary heap	$\log_d N$	$d \log_d N$	1
Fibonacci	1	$\log N$ †	1
Brodal queue	1	$\log N$	1
impossible	1	1	1

Sorting Algorithms

	inplace?	stable?	best	average	worst	remarks
selection	✔		$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	N exchanges
insertion	✔	✔	N	$\frac{1}{4} N^2$	$\frac{1}{2} N^2$	use for small N or partially ordered
shell	✔		$N \log_3 N$	?	$c N^{3/2}$	tight code; subquadratic
merge		✔	$\frac{1}{2} N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee; stable
timsort		✔	N	$N \lg N$	$N \lg N$	improves mergesort when preexisting order
quick	✔		$N \lg N$	$2 N \ln N$	$\frac{1}{2} N^2$	$N \log N$ probabilistic guarantee; fastest in practice
3-way quick	✔		N	$2 N \ln N$	$\frac{1}{2} N^2$	improves quicksort when duplicate keys
heap	✔		N	$2 N \lg N$	$2 N \lg N$	$N \log N$ guarantee; in-place
?	✔	✔	N	$N \lg N$	$N \lg N$	holy sorting grail

Symbol Tables

ST implementations: summary

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	½ N	N	½ N		`equals()`
binary search (ordered array)	$\lg N$	N	N	$\lg N$	½ N	½ N	✔	`compareTo()`
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	$\sqrt{N}$	✔	`compareTo()`
red-black BST	$2 \lg N$	$2 \lg N$	$2 \lg N$	$1.0 \lg N$	$1.0 \lg N$	$1.0 \lg N$	✔	`compareTo()`
separate chaining	N	N	N	3-5 *	3-5 *	3-5 *		`equals()` `hashCode()`
linear probing	N	N	N	3-5 *	3-5 *	3-5 *		`equals()` `hashCode()`

* under uniform hashing assumption

	sequential search	binary search
search	N	$\log N$
insert / delete	N	N
min / max	N	1
floor / ceiling	N	$\log N$
rank	N	$\log N$
select	N	1
ordered iteration	$N \log N$	N

Binary Search Trees

	sequential search	binary search	BST
search	N	$\lg N$	h
insert	N	N	h
min / max	N	1	h
floor / ceiling	N	$\lg N$	h
rank	N	$\lg N$	h
select	N	1	h
ordered iteration	$N \log N$	N	N

h = height of BST
(proportional to log N
if keys inserted in random order)

Worst case: h = O(N)

Balanced Binary Search Trees

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	$\frac{1}{2} N$	N	$\frac{1}{2} N$		`equals()`
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$\frac{1}{2} N$	$\frac{1}{2} N$	✔	`compareTo()`
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	$\sqrt{N}$	✔	`compareTo()`
goal	$\log N$	$\log N$	$\log N$	$\log N$	$\log N$	$\log N$	✔	`compareTo()`

Undirected Graphs

representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	$E + V$	1	$degree(v)$	$degree(v)$

* disallows parallel edges

Union Find

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N
weighted QU	N	lg N †	lg N	lg N

Directed Graphs

	best	worst	amortized
construct	1	1	1
push	1	N	1
pop	1	N	1
size	1	1	1

doubling and halving operations

order of growth of running time for resizing stack with N items

Digraph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices pointing from v.
- Real-world digraphs tend to be sparse.

huge number of vertices, small average vertex degree

representation	space	insert edge from v to w	edge from v to w?	iterate over vertices pointing from v?
list of edges	E	1	E	E
adjacency matrix	V^2	$1^{\dagger}$	1	V
adjacency lists	$E + V$	1	$outdegree(v)$	$outdegree(v)$

Dijkstra

Dijkstra's algorithm: which priority queue?

Depends on PQ implementation: V insert, V delete-min, E decrease-key.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$\cancel{E \log V}$ $(E+V)\log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$\cancel{E \log_{E/V} V}$ $(E+V)\log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

† amortized

Operations on an array resized

	best	worst	amortized
construct	1	1	1
push	1	N	1
pop	1	N	1
size	1	1	1

doubling and halving operations

order of growth of running time for resizing stack with N items