Big Oh Time Efficiency* Examples

***O*(*1*) — constant**
- finding a median value in a sorted array
- accessing any element in an array or ArrayList
- Push, Pop, Peek, and isEmpty operations for a stack (containing n elements);
- `add`, `remove`, `peek`, `&` `isEmpty` methods in `PriorityQueue`
- Insert and Remove operations for a queue.
- finding a key in a lookup table
- finding a key in an efficient, sparsely populated hash table
- retrieving a target value in an efficient, sparsely populated hash table
- adding an element to the end of an `ArrayList`
- `addFirst`, `addLast`, `getFirst`, `getLast`, `removeFirst`, & `removeLast` methods in `LinkedList`
- `put`, `get`, `containsKey`, & `size` methods in `HashMap`
- `add`, `remove`, `contains`, & `size` methods in `HashSet`

***O*(log *n*) — logarithmic**
- Binary Search in a sorted list of *n* elements
- searching a balanced binary search tree (worst case is O(n) if BST is unbalanced)
- inserting a node into a binary search tree
- `add` and `remove` methods in `PriorityQueue` (implemented as a heap)
- `containsKey`, `get`, & `put` methods in `TreeMap`

***O*(*n*) — linear**
- traversing a `List` (e.g. finding max or min)
- sequential search through an array or `ArrayList`
- calculating the sum of *n* elements in an array, `ArrayList`, `List`, or `Set`
- calculating *n*-factorial with a loop
- calculating Fibonacci numbers with a loop
- traversing a tree with *n* nodes

***O*(*n* log *n*) — "n log n" time**
- Mergesort
- Quicksort
- Heapsort
- creating a binary search tree if nodes inputted in random order leading to a balanced BST (worst case is $O(n^2)$)

$O(n^2)$ — quadratic
- Selection Sort
- Insertion Sort
- Bubble Sort
- traversing a two-dimensional array
- finding duplicates in an unsorted list of *n* elements (using nested loops)

$O(n^3)$ — cubic
- Conventional algorithm for matrix multiplication

$O(a^n)$ (where $a > 1$) **— exponential time**
- Recursive Fibonacci implementation
- Towers of Hanoi
- Generating all permutations of *n* letters
- Set partitioning
- Traveling salesman problem – dynamic programming solution

***O(n!)* — factorial**

- Traveling salesman problem – brute force solution
- Determinant Expansion by Minors

* most efficiencies are best case or average case unless noted

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)