Big Oh Time Efficiency* Examples

***O*(*1*) — constant**

- finding a median value in a sorted array
- `push`, `pop`, `peek`, & `isEmpty` methods in `Stack`
- `add`, `remove`, `peek`, & `isEmpty` methods in `PriorityQueue`
- 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 (array must be sorted)
- 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)

$O(a_n)$ (where $a > 1$) **— exponential time**

- Recursive Fibonacci implementation
- Towers of Hanoi
- Generating all permutations of *n* letters

* most efficiencies are best case or average case unless noted