# 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(n2))

***O*(*n2) —* 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*(*n3) —* cubic**

* Conventional algorithm for matrix multiplication

***O*(*an)*** (where *a* > 1) ***—* exponential time**

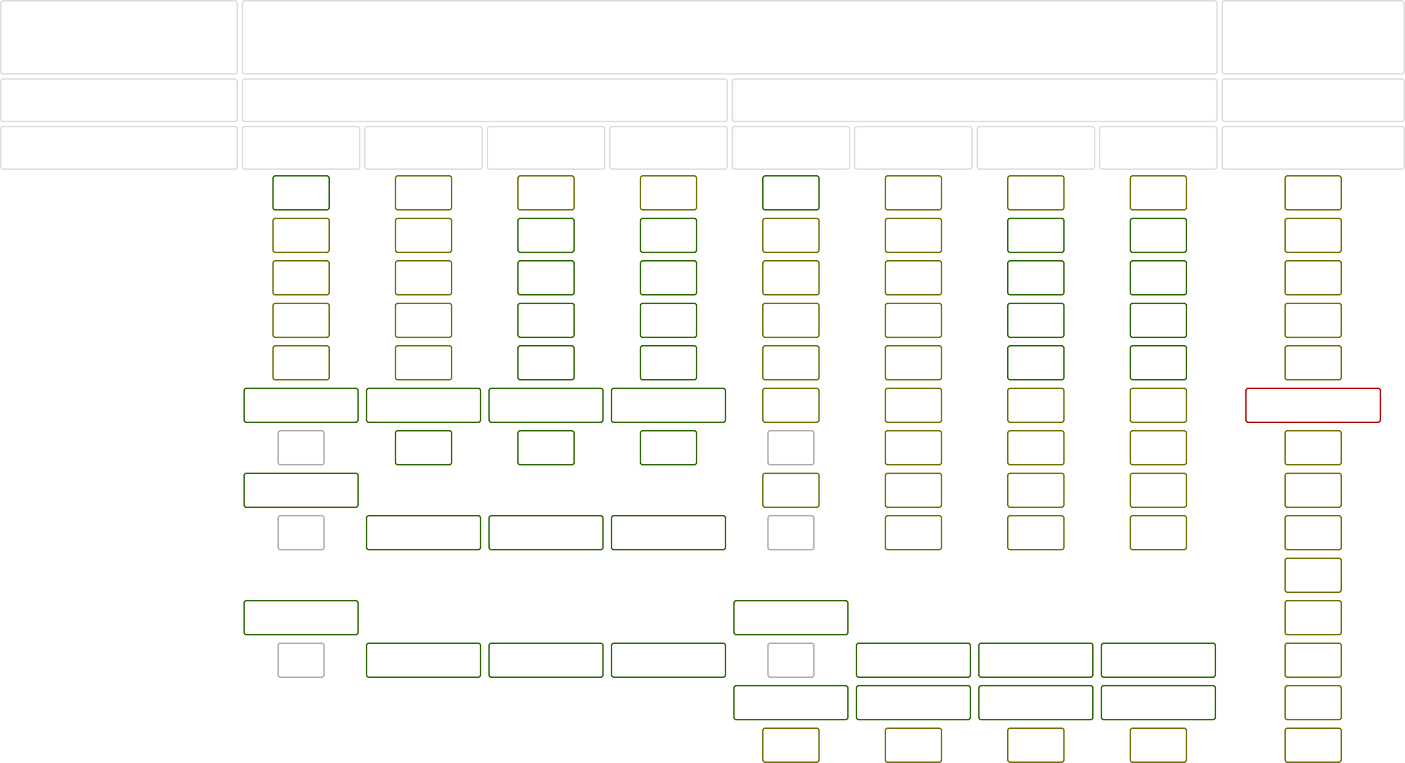
* Recursive Fibonacci implementation
* Towers of Hanoi
* Generating all permutations of *n* letters
* Set partitioning
* Traveling salesman problem – dynamic programming solution

***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

Average

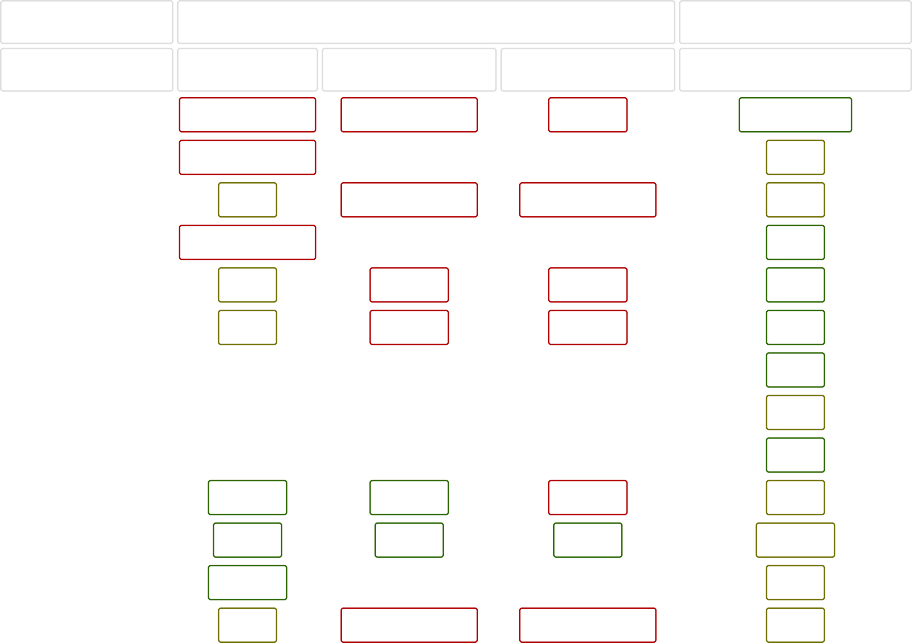
Worst

Space Complexity

Worst

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion |  |
| [Array](http://en.wikipedia.org/wiki/Array_data_structure) | Θ(1) | Θ(n) | Θ(n) | Θ(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| [Stack](http://en.wikipedia.org/wiki/Stack_(abstract_data_type)) | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Queue](http://en.wikipedia.org/wiki/Queue_(abstract_data_type)) | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Singly-Linked List](http://en.wikipedia.org/wiki/Singly_linked_list#Singly_linked_lists) | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Doubly-Linked List](http://en.wikipedia.org/wiki/Doubly_linked_list) | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Skip List](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Hash_table) | N/A | Θ(1) | Θ(1) | Θ(1) | N/A | O(n) | O(n) | O(n) | O(n) |
| [Binary Search Tree](http://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/K-d_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

[Quicksort](http://en.wikipedia.org/wiki/Quicksort) [Mergesort](http://en.wikipedia.org/wiki/Merge_sort) [Timsort](http://en.wikipedia.org/wiki/Timsort) [Heapsort](http://en.wikipedia.org/wiki/Heapsort) [Bubble Sort](http://en.wikipedia.org/wiki/Bubble_sort) [Insertion Sort](http://en.wikipedia.org/wiki/Insertion_sort) [Selection Sort](http://en.wikipedia.org/wiki/Selection_sort) [Tree Sort](https://en.wikipedia.org/wiki/Tree_sort) [Shell Sort](http://en.wikipedia.org/wiki/Shellsort) [Bucket Sort](http://en.wikipedia.org/wiki/Bucket_sort) [Radix Sort](http://en.wikipedia.org/wiki/Radix_sort) [Counting Sort](https://en.wikipedia.org/wiki/Counting_sort)

[Cubesort](https://en.wikipedia.org/wiki/Cubesort)

Best

Ω(n log(n))

Average Worst

Θ(n log(n)) O(n^2)

Space Complexity Worst

O(log(n))

Ω(n log(n))

Θ(n log(n))

O(n log(n))

O(n)

Ω(n)

Θ(n log(n))

O(n log(n))

O(n)

Ω(n log(n))

Θ(n log(n))

O(n log(n))

O(1)

Ω(n)

Θ(n^2)

O(n^2)

O(1)

Ω(n)

Θ(n^2)

O(n^2)

O(1)

Ω(n^2)

Θ(n^2)

O(n^2)

O(1)

Ω(n log(n))

Θ(n log(n))

O(n^2)

O(n)

Ω(n log(n))

Θ(n(log(n))^2)

O(n(log(n))^2)

O(1)

Ω(n+k)

Θ(n+k)

O(n^2)

O(n)

Ω(nk)

Θ(nk)

O(nk)

O(n+k)

Ω(n+k)

Θ(n+k)

O(n+k)

O(k)

Ω(n)

Θ(n log(n))

O(n log(n))

O(n)