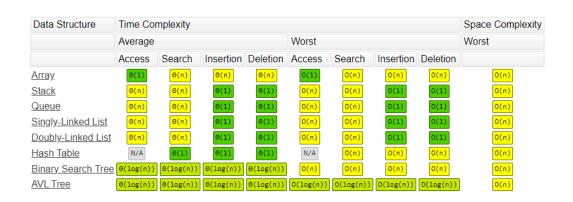
# Time Complexity (Big-O) Cheatsheet

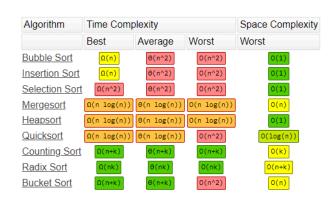
Big O, Big Theta and Big Omega notations express an algorithm's time and space complexity. It helps us to quantify the efficiency of our code.

- **Big O:** Worst-case performance of an algorithm.
- **Big Theta (O):** Average performance of an algorithm.
- **Big Omega (Ω):** Best-case performance of an algorithm.

#### **Common Data Structures**



# **Sorting Algorithms**



### **Others Algorithms**

Linear Search - O(n)

Binary Search - O(logn)

Set - Add/remove/search: O(1)

**Heap** (it is always balanced):

- add/delete O(logn)
- search O(n)
- search min/max O(1)

#### **Dynamic Programming**:

- Space Complexity O(n)
- Time Complexity O(n.k), k is work done in each step

**Recursion**: mostly true for Backtracking also

- if calls itself 2 times, O(2^n)
- if calls itself 3 times, O(3^n)

#### **Divide and Conquer**

#### Greedy

## Input size, Complexity & Algo guess

- $\mathbf{n} <= 10$ : maybe  $O(n^2.n!)$  or  $O(4^n)$  backtracking or brute-force recursive
- 10 < n <= 20: 0(2^n) backtracking and recursion
- 20 < n <= 100: 0(n^3) nested loops
- 100 < n <= 1,000:  $O(n^2)$  nested loops with something efficient
- 1,000 < n < 100,000: O(n.logn) or O(n) hashmap, 2 pointers, sliding window, monotonic stack, binary search, heap
- 100,000 < n < 1,000,000: O(n) or O(n.logn) with small k maybe hashmap or binary search
- 1,000,000 < n: 0(logn) or 0(1) math formula, binary search or hashmap