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

Common Data Structures

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)
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)
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)

Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
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)
Mergesort	Ω(n log(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)
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)

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

- **n <= 10:** `maybe O(n^2.n!) or O(4^n)` - backtracking or brute-force recursive
- **10 < n <= 20:** `O(2^n)` - backtracking and recursion
- **20 < n <= 100:** `O(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:** `O(logn) or O(1)` - math formula, binary search or hashmap