Big O's

`O(1)` Constant - no loops

`O(log N)` Logarithmic - usually searching algorithms have log n if they are sorted (Binary Search)

`O(n)` Linear - for loops, while loops through n items

`O(n log(n))` Log Linear - usually sorting operations

`O(n^2)` Quadratic - every element in a collection needs to be compared to ever other element. Two nested loops

`O(2^n)` Exponential - recursive algorithms that solves a problem of size N

`O(n!)` Factorial - you are adding a loop for every element

Iterating through half a collection is still `O(n)`

Two separate collections: `O(a * b)`

Big O	Name	Description
1	Constant	statement, one line of code
log(n)	Logarithmic	Divide and conquer (binary search)
n	Linear	Loop
n*log(n)	Linearithmic	Effective sorting algorithms
n^2	Quadratic	Double loop
n^3	Cubic	Triple loop
2^n	Exponential	Complex full search

What Can Cause Time in a Function?

- Operations (`+`, `-`, `\*`, `/`)
- Comparisons (`<`, `>`, `===`)
- Looping (`for`, `while`)
- Outside Function call (`function()`)

Sorting Algorithms

Sorting Algorithms	Space complexity	Time complexity	Time complexity
	Worst case	Best case	Worst case
Insertion Sort	O(1)	O(n)	O(n^2)
Selection Sort	O(1)	O(n^2)	O(n^2)
Bubble Sort	O(1)	O(n)	O(n^2)
Mergesort	O(n)	O(n log n)	O(n log n)
Quicksort	O(log n)	O(n log n)	O(n^2)
Heapsort	O(1)	O(n log n)	O(n log n)

Common Data Structure Operations

Worst Case→	Access	Search	Insertion	Deletion	Space Complexity
Array	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)
Hash Table	N/A	O(n)	O(n)	O(n)	O(n)

Rule Book

Rule 1: Always worst Case

Rule 2: Remove Constants

Rule 3:

- Different inputs should have different variables: `O(a + b)`.
- A and B arrays nested would be: `O(a * b)`

+ for steps in order

* for nested steps

Rule 4: Drop Non-dominant terms

What Causes Space Complexity?

- Variables
- Data Structures
- Function Call
- Allocations