Time Complexity
&
Space Complexity

Big-O notations

Big O's

It is known as the algorithm's upper bound since it analyses the worst-case situation.

O(1)	:	Constant	⇒	No loops (One line of code)
O(logN)	:	Logarithmic	⇒	Usually searching algorithms have log n if they are sorted (Divide & Conquer, Binary Search)
O(n)	:	Linear	⇒	For loops, While loops through n items
O(nlogn)	:	Log linear	⇒	Usually sorting operations (Sorting Algorithms)
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 (Complex Full Search)
O(n!)	:	Factorial	⇒	You are adding a loop for every element (Permutation, Combination)

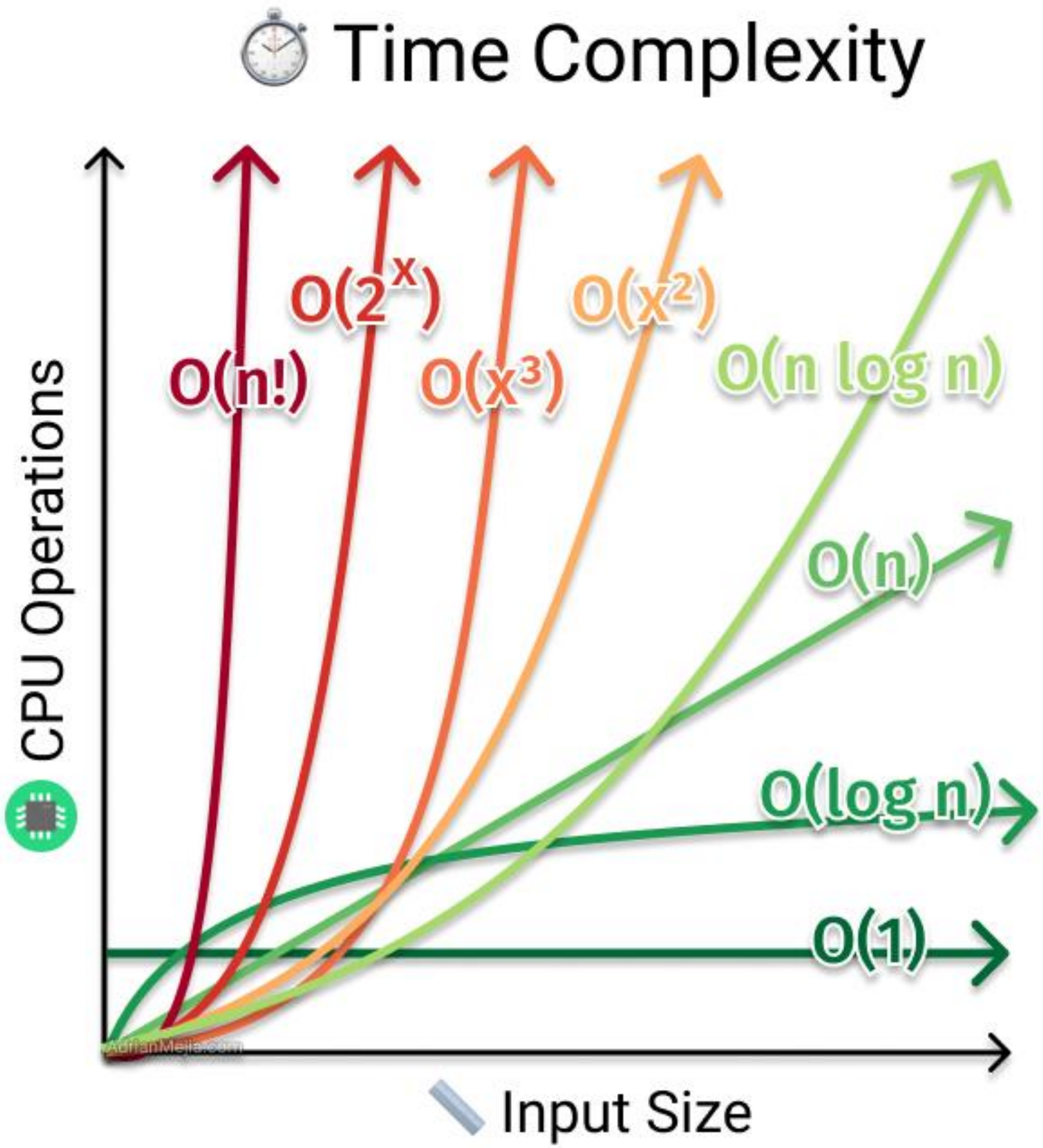
Graph Representation of Time Complexity

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.

The space complexity is overall amount of memory or space utilized by an algorithm/program, including the space of input values for execution. To determine space complexity, simply compute how much space the variables in an algorithm/a program take up.

Space Complexity = Auxiliary space
\+
Space used by input values.



Input Length	Worst Accepted Time Complexity	Usually type of solutions
N <= 12	$O(N!)$	Recursion and backtracking
N <= 25	$O(2^N * N)$	Recursion, backtracking, and bit manipulation
N <= 40	$O(2^{N/2} * N)$	Meet in the middle, Divide and Conquer
N <= 100	$O(N^4)$	Dynamic programming, Constructive
n <= 500	$O(N^3)$	Dynamic programming, Constructive
N <= 2000	$O(N^2 * \log N)$	Dynamic programming, Binary Search, Sorting, Divide and Conquer
N <= 10^4	$O(N^2)$	Dynamic programming, Graph, Trees, Constructive
N <= 10^6	$O(N \log N)$	Sorting, Binary Search, Divide and Conquer
N <= 10^8	$O(N)$, $O(\log N)$, $O(1)$	Constructive, Mathematical, Greedy Algorithms

Arrays : Basic Operations Time Complexity

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Accessing an element	O(1)	θ (1)	Ω(1)
Updating an element	O(1)	θ (1)	Ω(1)
Deleting an element	O(n)	θ (n)	Ω(1)
Inserting an element	O(n)	θ (n)	Ω(1)
Searching for an element	O(n)	θ (n)	Ω(1)

Sorting & Searching : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Quick Sort	$O(n^2)$	$\theta\,(n \log n)$	$\Omega\,(n \log n)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$\theta\,(n \log n)$	$\Omega\,(n \log n)$	$O(n)$
Heap Sort	$O(n \log n)$	$\theta\,(n \log n)$	$\Omega\,(n \log n)$	$O(1)$
Bubble Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n)$	$O(1)$
Insertion Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n)$	$O(1)$
Selection Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n^2)$	$O(1)$
Binary Search	$O(n \log n)$	$\theta\,(n \log n)$	$\Omega\,(1)$	$O(1)$
Linear Search	$O(n)$	$\theta\,(n)$	$\Omega\,(1)$	$O(1)$

Strings : Basic Operations Time Complexity

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Accessing	$O(1)$	$\theta(1)$	$\Omega(1)$
Deleting	$O(n)$	$\theta(n)$	$\Omega(1)$
Inserting	$O(n)$	$\theta(n)$	$\Omega(1)$
Searching (n = string length m = pattern length)	$O(n * m)$	$\theta(n)$	$\Omega(1)$
Concatenating (n, m = string lengths)	$O(n + m)$	$\theta(n + m)$	$\Omega(n)$
Comparison (n = shorter string length)	$O(n)$	$\theta(n)$	$\Omega(n)$
Inserting (Trie) (m = key length)	$O(m)$	$\theta(m)$	$\Omega(1)$
Searching (Trie) (m = key length)	$O(m)$	$\theta(m)$	$\Omega(1)$

Strings : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Radix sort (m = longest string length)	$O(n * m)$	$\theta(n * m)$	$\Omega(n * m)$	$O(n + m)$
Naive string search (m = size of pattern)	$O(m * (n-m+1))$	$\theta(n * m)$	$\Omega(n)$	$O(1)$
Knuth-Morris Pratt search	$O(m + n)$	$\theta(n)$	$\Omega(n)$	$O(m)$
Boyer-Moore string search	$O(n * m)$	$\theta(n)$	$\Omega(n / m)$	$O(m)$
Rubin-Karp Algorithm	$O(m * (n-m+1))$	$\theta(n + m)$	$\Omega(m)$	$O(m)$

Stacks & Queues : Basic Operations Time Complexity

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Delete (Stack)	$O(1)$	$\theta(1)$	$\Omega(1)$
Insert (Stack)	$O(1)$	$\theta(1)$	$\Omega(1)$
Search (Stack)	$O(n)$	$\theta(n)$	$\Omega(1)$
Peek/Top (Stack)	$O(1)$	$\theta(1)$	$\Omega(1)$
Delete (Queue)	$O(1)$	$\theta(1)$	$\Omega(1)$
Insert (Queue)	$O(1)$	$\theta(1)$	$\Omega(1)$
Search (Queue)	$O(n)$	$\theta(n)$	$\Omega(1)$

Stacks & Queues : Algorithm Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Linear Search	$O(n)$	$\theta(n)$	$\Omega(1)$	$O(1)$

Stack: LIFO (Last In First Out)

Queue: FIFO (First In First Out)

Linked Lists : Basic Operations Time Complexity

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Accessing	$O(n)$	$\theta(n)$	$\Omega(1)$
Deleting (after search)	$O(1)$	$\theta(1)$	$\Omega(1)$
Inserting (after search)	$O(1)$	$\theta(1)$	$\Omega(1)$
Searching	$O(n)$	$\theta(n)$	$\Omega(1)$
Traversing	$O(n)$	$\theta(n)$	$\Omega(n)$
Access (Skip List)	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$
Delete (Skip List)	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$
Insert (Skip List)	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$
Search (Skip List)	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$

Linked Lists : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Merge Sort	$O(n \log n)$	$\theta\,(n \log n)$	$\Omega\,(n \log n)$	$O(n)$
Bubble Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n)$	$O(1)$
Selection Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n^2)$	$O(1)$
Insertion Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n)$	$O(1)$
Linear Search	$O(n)$	$\theta\,(n)$	$\Omega\,(1)$	$O(1)$

Maps : Basic Operations Time Complexity

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Updating an element	O(n)	θ (1)	Ω(1)
Inserting an element	O(n)	θ (1)	Ω(1)
Deleting an element	O(n)	θ (1)	Ω(1)
Searching for an element	O(n)	θ (1)	Ω(1)
Insert (TreeMap)	O(log n)	θ (log n)	Ω(1)
Delete (TreeMap)	O(log n)	θ (log n)	Ω(1)
Search (TreeMap)	O(log n)	θ (log n)	Ω(1)

Maps : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Bucket Sort (k = buckets)	$O(n^2)$	$\theta\,(n + k)$	$\Omega\,(n + k)$	$O(n)$
Insertion Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n)$	$O(1)$
Selection Sort	$O(n^2)$	$\theta\,(n^2)$	$\Omega\,(n^2)$	$O(1)$
Heap Sort	$O(n \log n)$	$\theta\,(n \log n)$	$\Omega\,(n \log n)$	$O(1)$
Hash-based Search	$O(n)$	$\theta\,(1)$	$\Omega\,(1)$	$O(1)$
Binary Search	$O(\log n)$	$\theta\,(\log n)$	$\Omega\,(1)$	$O(1)$
Linear Search	$O(n)$	$\theta\,(n)$	$\Omega\,(1)$	$O(1)$
Rabin-Karp Algorithm	$O(m * (n-m+1))$	$\theta\,(n + m)$	$\Omega\,(m)$	$O(m)$

Heaps : Basic Operations Time Complexity

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Insert	O(log n)	θ (log n)	Ω(1)
Delete	O(log n)	θ (log n)	Ω(1)
Find min/max	O(1)	θ (1)	Ω(1)
Search	O(n)	θ (n)	Ω(1)
Insert (Fibonacci/Binomial)	O(log n)	θ (1)	Ω(1)
Increase/Decrease key	O(log n)	θ (log n)	Ω(1)
Extract min/max	O(log n)	θ (log n)	Ω(log n)

Heaps : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Heap Sort	$O(n \log n)$	$\theta(n \log n)$	$\Omega(n \log n)$	$O(1)$
Smooth Sort	$O(n \log n)$	$\theta(n \log n)$	$\Omega(n)$	$O(n)$
Quick Sort	$O(n^2)$	$\theta(n)$	$\Omega(n)$	$O(1)$
Linear Search	$O(n)$	$\theta(n)$	$\Omega(1)$	$O(1)$
Dijkstra's shortest path	$O(V^2)$	$\theta(E * \log(V))$	$\Omega(E * \log(V))$	$O(V)$

Trees : Basic Operations Time Complexity

		Worst Case Scenario	Average Case Scenario	Best Case Scenario
Binary Search Tree, Cartesian Tree, KD Tree	Delete	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$
	Insert	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$
	Search	$O(n)$	$\theta(\log n)$	$\Omega(\log n)$
B-Tree, Red-Black Tree, Splay Tree, AVL Tree	Delete	$O(\log n)$	$\theta(\log n)$	$\Omega(\log n)$
	Insert	$O(\log n)$	$\theta(\log n)$	$\Omega(\log n)$
	Search	$O(\log n)$	$\theta(\log n)$	$\Omega(\log n)$
Traversal		$O(n)$	$\theta(n)$	$\Omega(n)$

Trees : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Depth-First Search (In-order, pre-order, & post-order traversal)	$O(n)$	$\theta\ (n)$	$\Omega(n)$	$O(n)$
Breadth-First Search (Level-order traversal)	$O(n)$	$\theta\ (n)$	$\Omega(n)$	$O(n)$
Tree Sort	$O(n^2)$	$\theta\ (n \log n)$	$\Omega(n \log n)$	$O(n)$
Splay Sort	$O(n \log n)$	$\theta\ (n \log n)$	$\Omega(n)$	$O(n)$
Cartesian Tree Sort	$O(n \log n)$	$\theta\ (n \log n)$	$\Omega(n)$	$O(n)$

Graphs : Basic Operations Time Complexity

		Worst Case Scenario	Average Case Scenario	Best Case Scenario
Insert Vertex	Adjacency List	$O(1)$	$\theta(1)$	$\Omega(1)$
	Adjacency Matrix	$O(V^2)$	$\theta(V^2)$	$\Omega(V^2)$
Remove Vertex	Adjacency List	$O(E)$	$\theta(E)$	$\Omega(E)$
	Adjacency Matrix	$O(V^2)$	$\theta(V^2)$	$\Omega(V^2)$
Insert Edge	Adjacency List	$O(1)$	$\theta(1)$	$\Omega(1)$
	Adjacency Matrix	$O(1)$	$\theta(1)$	$\Omega(1)$

Graphs : Basic Operations Time Complexity

		Worst Case Scenario	Average Case Scenario	Best Case Scenario
Remove Edge	Adjacency List	O(V)	θ(V)	Ω(V)
	Adjacency Matrix	O(1)	θ(1)	Ω(1)
Check if Vertices Adjacent	Adjacency List	O(V)	θ(V)	Ω(V)
	Adjacency Matrix	O(1)	θ(1)	Ω(1)

Graphs : Algorithms Complexity

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Breadth-First Search	O(V+E)	θ (V+E)	Ω(V+E)	O(V)
Depth-First Search)	O(V+E)	θ (V+E)	Ω(V+E)	O(V)
A* Search	O(E)	θ (E)	Ω(E)	O(V)
Dijkstra’ s algorithm	$O(V^2)$	θ (E * log(V))	Ω(E * log(V))	O(V)
Floyd-Warshall	O(V3)	θ (V3)	Ω(V3)	$O(V^2)$