Big O Notation for Coding Interviews

By comparing the efficiency of different approaches to a problem, Big O helps you write better software, and ace coding interview problems.

> What is Big O?

Big O notation measures the efficiency and performance of an algorithm by analyzing its time and space complexity.

- **Time Complexity:** Measures the total amount of time an algorithm takes to execute as a function of its input size
- **Space Complexity:** Measures the total amount of memory or space required by an algorithm to execute as a function of its input size

> Constant Time: O(1)

The running time does not change with the size of the input. The algorithm always takes the same amount of time to complete.

- **Example:** Accessing an element in an array by index

> Linear Time: O(n)

The running time increases linearly with the size of the input. If the input size doubles, the running time also doubles.

- **Example:** Iterating through an array

> Logarithmic Time: O(log n)

The running time increases logarithmically as the input size increases. If the input size doubles, the running time increases by a constant amount(very slowly as compared to the input size).

- **Example:** Binary search

> Quadratic Time: O(n^2)

The running time increases quadratically with the size of the input. If the input size doubles, the running time increases by a factor of four.

- **Example:** Nested loops and bubble

> Quasilinear Time: O(n log n))

The running time grows in proportion to n multiplied by the logarithm of n.

- **Example:** Merge sort and heap

> Exponential Time: O(2^n)

The running time doubles with each additional element in the input. If the input size increases by one, the running time increases by a factor of two.

- **Example:** Recursive algorithms solving the traveling salesman problem

> Factorial Time: O(n!)

The running time grows in proportion to the factorial of the input size, n. This complexity indicates extremely rapid growth, making such algorithms impractical for large inputs.

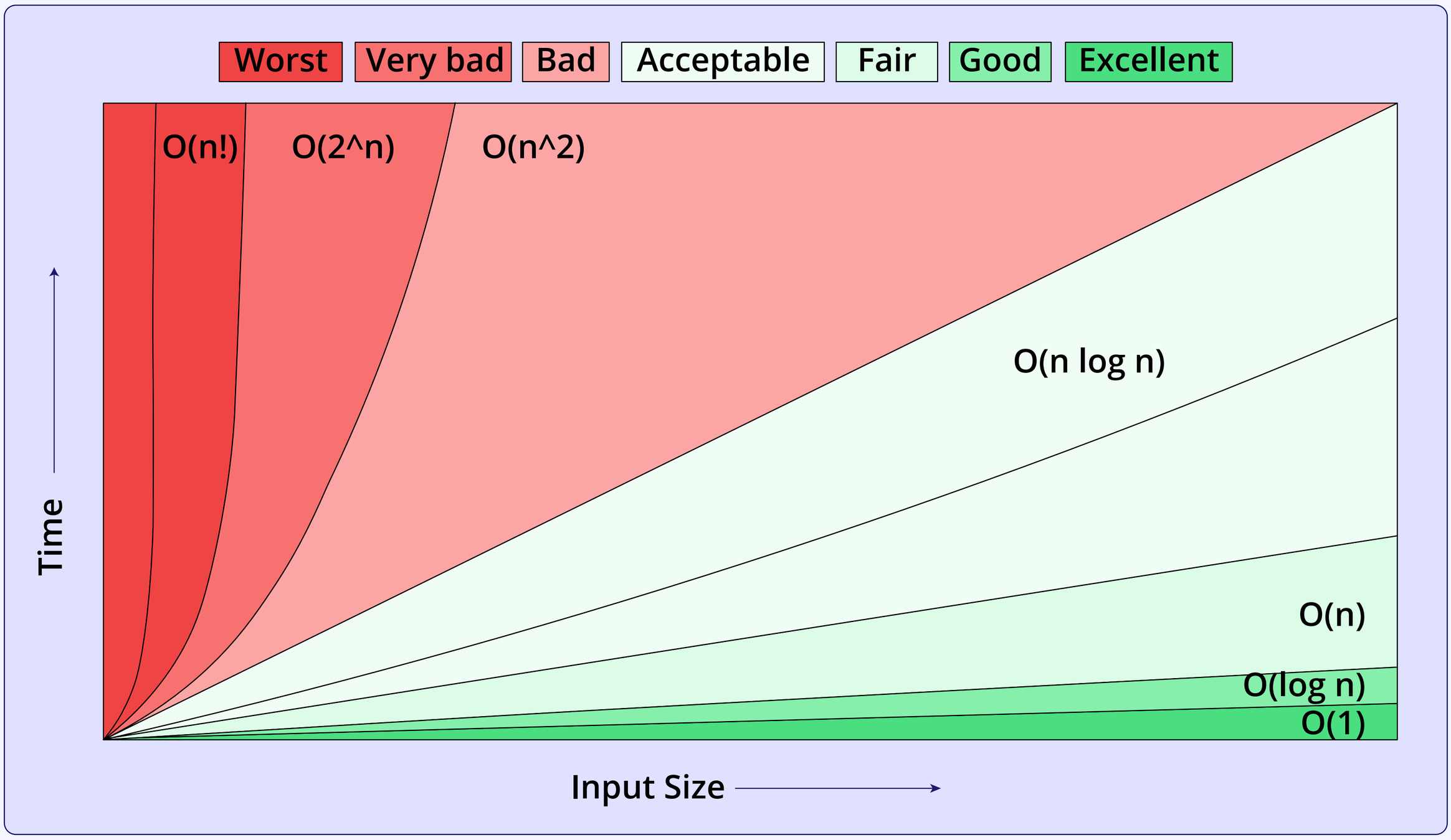
- **Example:** Generate all permutations of a string

Big O Complexity Chart

> Complexity Classes

- O(1): Excellent
- O(log n): Good
- O(n): Fair
- O(n log n): Acceptable
- O(n^2): Bad
- O(2^n): Very Bad
- O(n!): Worst

> Visual Representation



> Complexity of Common Data Operations

Static Data Structures

Data Structure	Operation	Average Case	Worst Case
Array	Access	O(1)	O(1)
	Search	O(n)	O(n)
	Insertion	O(n)	O(n)
	Deletion	O(n)	O(n)

> Complexity of Common Data Operations

Dynamic Data Structures			
Data Structure	**Operation**	**Average Case**	**Worst Case**
Stack	Access	O(n)	O(n)
	Search	O(n)	O(n)
	Insertion (Push)	O(1)	O(1)
	Deletion (Pop)	O(1)	O(1)
Queue	Access	O(n)	O(n)
	Search	O(n)	O(n)
	Insertion (Enqueue)	O(1)	O(1)
	Deletion (Dequeue)	O(1)	O(1)

Dynamic Data Structures			
Data Structure	**Operation**	**Average Case**	**Worst Case**
Singly Linked List	Access	O(n)	O(n)
	Search	O(n)	O(n)
	Insertion (at head)	O(1)	O(1)
	Insertion (at tail or any position)	O(n)	O(n)
	Deletion (head)	O(1)	O(1)
	Deletion (middle or any position)	O(n)	O(n)
Doubly Linked List	Access	O(n)	O(n)
	Search	O(n)	O(n)
	Insertion (at head)	O(1)	O(1)
	Insertion (at tail or any position)	O(n)	O(n)
	Deletion (head)	O(1)	O(1)
	Deletion (middle or any position)	O(n)	O(n)

> Complexity of Common Data Operations

Hash-Based Data Structures			
Data Structure	**Operation**	**Average Case**	**Worst Case**
Hash Table	Access	O(1)	O(n)
	Search	O(1)	O(n)
	Insertion	O(1)	O(n)
	Deletion	O(1)	O(n)

Tree-Based Data Structures			
Data Structure	**Operation**	**Average Case**	**Worst Case**
Binary Tree	Access	O(n)	O(n)
	Search	O(n)	O(n)
	Insertion	O(n)	O(n)
	Deletion	O(n)	O(n)
Binary Search Tree	Access	O(log n)	O(n)
	Search	O(log n)	O(n)
	Insertion	O(log n)	O(n)
	Deletion	O(log n)	O(n)

> Complexity of Common Data Operations

Tree-Based Data Structures			
Data Structure	**Operation**	**Average Case**	**Worst Case**
Red-Black Tree	Access	O(log n)	O(log n)
	Search	O(log n)	O(log n)
	Insertion	O(log n)	O(log n)
	Deletion	O(log n)	O(log n)
AVL Tree	Access	O(log n)	O(log n)
	Search	O(log n)	O(log n)
	Insertion	O(log n)	O(log n)
	Deletion	O(log n)	O(log n)