Maksim Bannikov
@mbannikov

Stop Failing Tech Interviews

Complete Big O Cheat Sheet

Why Big O Matters

Interview Impact

- Top tech companies evaluate algorithmic efficiency in interviews
- Shows systematic thinking and optimization skills
- Critical for passing technical screenings

Real-World Impact

- Scale applications to handle millions of users
- Reduce infrastructure costs and response times
- Critical for mobile and resource-constrained environments

Understanding Big O is not just about passing interviews — it's about building efficient, scalable solutions that make a real difference.

Big O Notation Basics

What is Big O?

Big O notation describes the upper bound of growth rate of an algorithm's resource usage (time or space) relative to input size.

O(f(n)) = how algorithm's performance scales with input size n

Best case

Minimum time required in ideal conditions

Example: Finding value at start of array

Average Case

Expected time in typical conditions

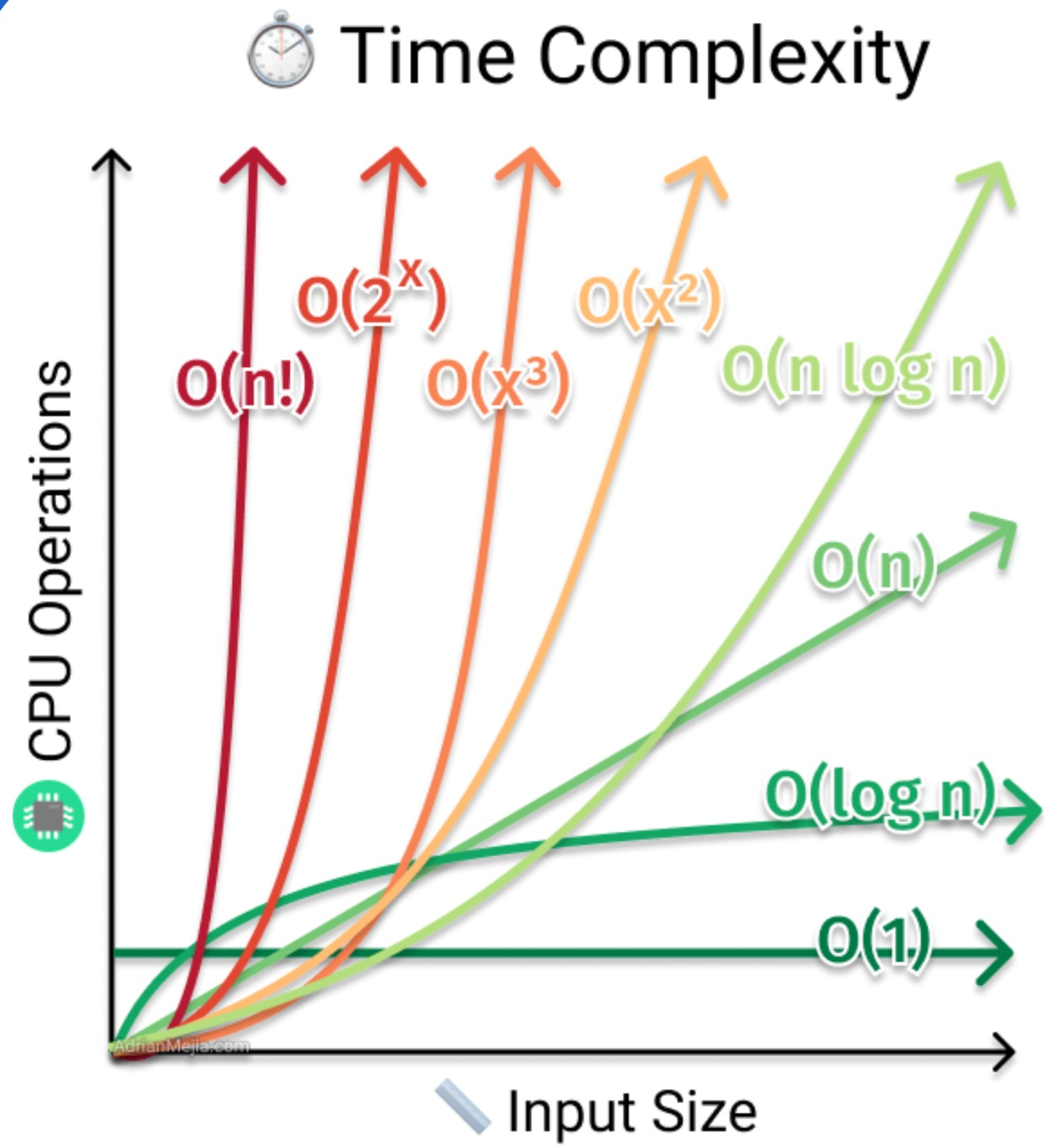
Example: Finding random value in array

Worst Case

Maximum time in worst conditions

Example: Finding value at end of array

Common Time Complexities



Common Time Complexities

$O(1)$ - Constant

Always takes same time regardless of input size

Array access, Hash table insertion

$O(\log n)$ - Logarithmic

Time increases slowly as input grows

Binary search, Balanced BST operations

$O(n)$ - Linear

Time grows linearly with input

Linear search, Array traversal

$O(n \log n)$ - Linearithmic

Common in efficient sorting algorithms

Merge sort, Quick sort

$O(n^2)$ - Quadratic

Time grows with square of input

Nested loops, Bubble sort

$O(2^n)$ - Exponential

Time doubles with each additional input

Recursive Fibonacci, Tower of Hanoi

Array

Access by index	O(1)
Insert/remove at end	O(1)
Insert/remove at beginning	O(n)
Search (unsorted)	O(n)
Search (sorted)	O(log n)

Excellent (O(1)) Good (O(log n)) Expensive (O(n) or worse)

Object / HashMap

Insert key-value	**O(1)***
Remove key-value	**O(1)***
Access by key	**O(1)***
Search by value	**O(n)**

** Average case, assuming good hash function*

Excellent (O(1)) Good (O(log n)) Expensive (O(n) or worse)

Linked List

Access by index	**O(n)**
Insert/remove at start	**O(1)**
Insert/remove at end	**O(n)***
Search	**O(n)**

** O(1) with tail pointer*

Excellent (O(1))   Good (O(log n))   Expensive (O(n) or worse)

Stack & Queue

Stack: Push	O(1)
Stack: Pop	O(1)
Queue: Enqueue	O(1)
Queue: Dequeue	O(1)
Peek (both)	O(1)

Excellent (O(1)) Good (O(log n)) Expensive (O(n) or worse)

Binary Search Tree

Search	**O(log n)***
Insert	**O(log n)***
Delete	**O(log n)***
Find Min/Max	**O(log n)***
Inorder Traversal	**O(n)**

** O(n) worst case for unbalanced tree*

Excellent (O(1))

Good (O(log n))

Expensive (O(n) or worse)

Heap (Priority Queue)

Get Min/Max	O(1)
Insert	O(log n)
Remove Min/Max	O(log n)
Heapify	O(n)
Search	O(n)

Common Use Cases:

- Top K problems
- Priority scheduling
- Dijkstra's algorithm

Excellent (O(1)) Good (O(log n)) Expensive (O(n) or worse)

Space Complexity

Complexity	Description	Example
O(1)	Fixed amount of space	Variables, simple loops
O(n)	Linear space growth	Arrays, Hash tables
O(log n)	Logarithmic space	Binary search recursion
$O(n^2)$	Quadratic space	2D arrays, adjacency matrix

ⓘ Common Pitfalls

- Recursive call stack (often forgotten)
- String concatenation creating copies
- Copying arrays/objects in loops
- Cache/memoization trade-offs

ⓘ Interview Tips

- Consider in-place algorithms when possible
- Discuss space-time trade-offs
- Watch for recursion depth
- Mention both auxiliary and input space

Sorting Algorithms

Algorithm	Time Complexity	Space
Merge Sort	**O(n log n)**	**O(n)**
Quick Sort	**O(n log n)***	**O(log n)**
Heap Sort	**O(n log n)**	**O(1)**
Bubble Sort	**$O(n^2)$**	**O(1)**
Insertion Sort	**$O(n^2)$**	**O(1)**

** $O(n^2)$ worst case for bad pivot selection*

When to Use What

Merge Sort: When stable sorting is needed and extra space is available
Quick Sort: General purpose, when in-place sorting is needed
Heap Sort: When guaranteed O(n log n) and constant space is required
Insertion Sort: For small arrays or nearly sorted data

Good (O(n log n)) Average (O(n log n) with caveats) Poor ($O(n^2)$)

Search Algorithms

Algorithm	Time Complexity	Space
Linear Search	$O(n)$	$O(1)$
Binary Search	$O(\log n)$	$O(1)$
BFS	$O(V + E)$	$O(V)$
DFS	$O(V + E)$	$O(V)$

ⓘ When to Use What

Linear Search: Small arrays or unsorted data
Binary Search: Sorted arrays, search space reduction
BFS: Shortest path, level-order traversal
DFS: Path finding, topological sort, cycles

Good ($O(n \log n)$) Average ($O(n \log n)$ with caveats) Poor ($O(n^2)$)

Algorithm Interview Tips

Problem Analysis

- Clarify inputs & constraints
- Discuss edge cases
- Think out loud
- Start with brute force

While Coding

- Write clean, readable code
- Use meaningful names
- Handle edge cases
- Explain your approach

Watch Out For

- Off-by-one errors
- Null/undefined checks
- Array bounds
- Integer overflow

Maksim Bannikov
@mbannikov

Found this helpful?

Help others find this content!

Like the post

Share with colleagues

Follow **@mbannikov**
for more tech content