big O notation

By Nader Mohamed

Introduction

Definition

Big O notation is a way to describe how the runtime or space of an algorithm grows as the input size increases.

Purpose

It helps us understand how an algorithm behaves with large inputs and compare the efficiency of different algorithms.

Why Big O Matters

In programming, efficiency is key. As the amount of data grows, we need to know how our algorithms will perform. Big O helps us predict:

How fast an algorithm runs as the input gets larger.
How much memory an algorithm uses.

For example, choosing a slow algorithm can make a program take hours to run instead of seconds.

Common Big O Notations

O(1) – Constant Time

- The algorithm runs in the same amount of time, no matter how big the input is.
- Example: Accessing an element in an array.

O(n) – Linear Time

- The time grows proportionally with the input size.
- Example: Iterating through all elements in a list.

O(log n) – Logarithmic Time

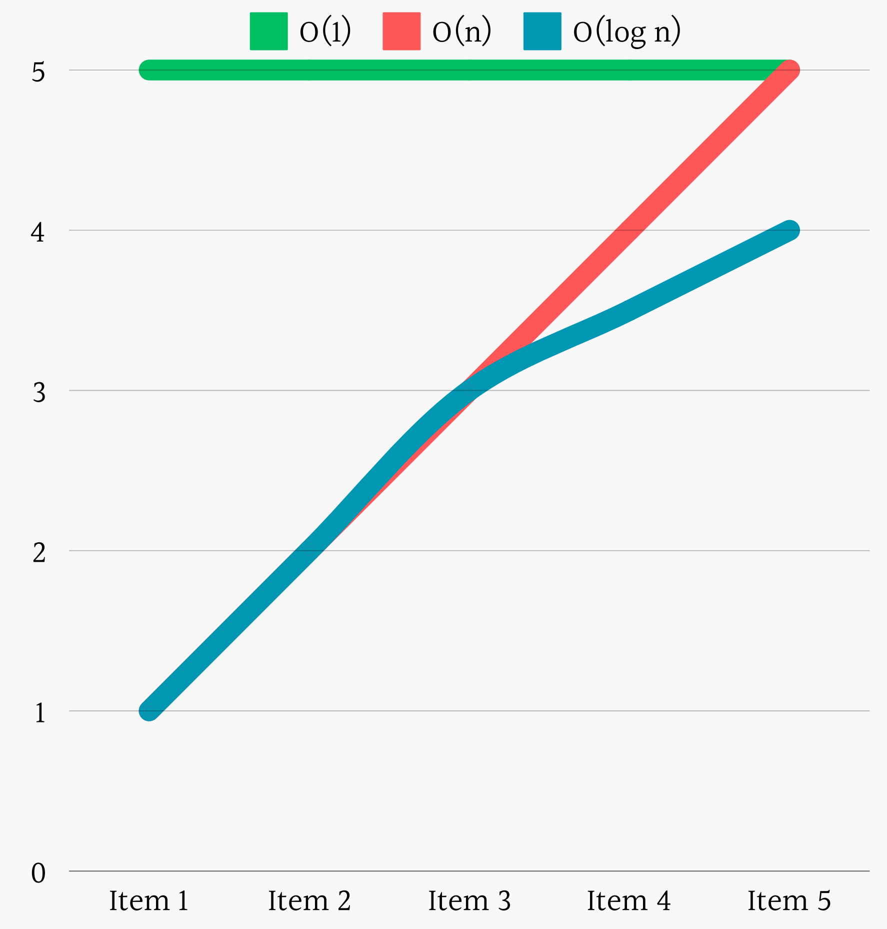
- The algorithm's time grows slowly as the input size increases.
- Example: Binary search (dividing a sorted list in half).

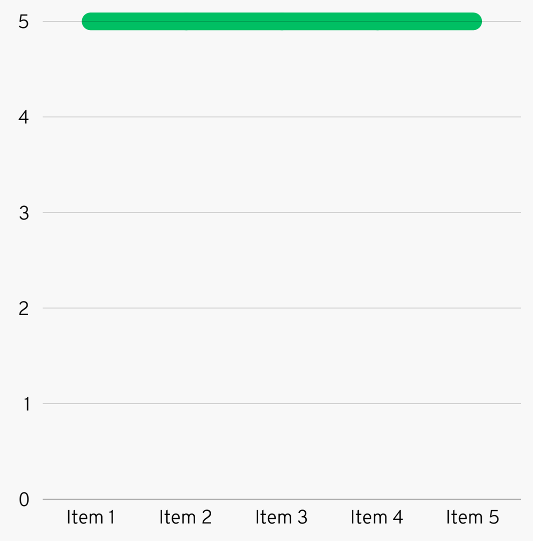
$O(n^2)$ – Quadratic Time

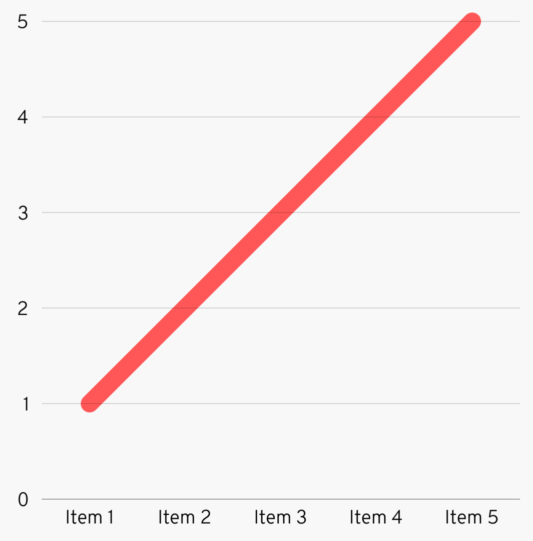
- Time grows quickly, often due to nested loops.
- Example: Selection sort.

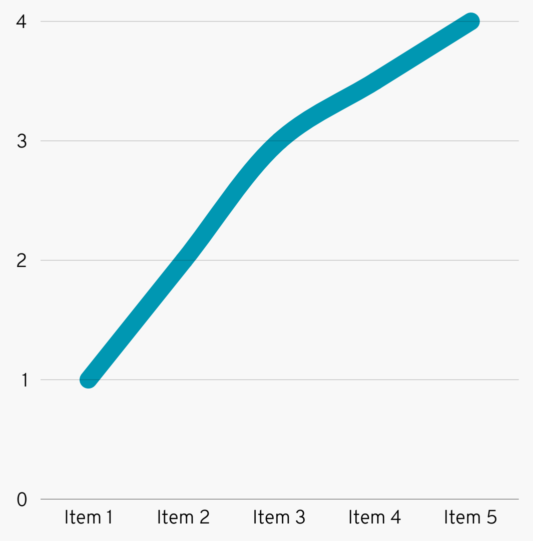
Visualizing Big O

Let's compare how these complexities look as the input size increases:









O(1)

stays flat (same time regardless of size).

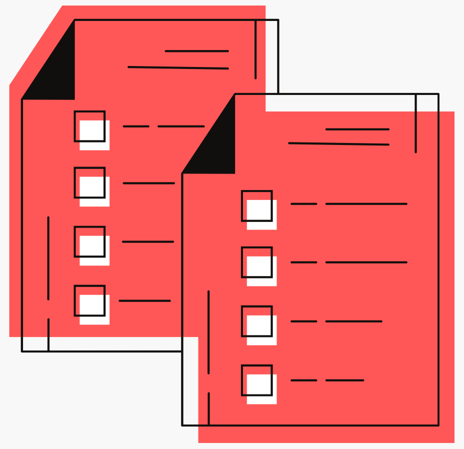
O(n)

grows steadily.

O(log n)

grows slowly.

Examples



Binary Search

Imagine searching for a name in a phone book. With binary search, you can divide the book in half each time, drastically reducing the number of pages to look through. This is much faster than checking every page.

Selection Sort ($O(n^2)$)

In selection sort, you find the smallest element in the list and move it to the front. You repeat this for every element, so as the list grows, the number of comparisons grows very quickly, making it slow.

Merge Sort ($O(n \log n)$)

Merge sort is a faster sorting algorithm that splits the list into halves and sorts them. It's more efficient than simple algorithms like selection sort, especially for large datasets.

CONCLUSION

- Big O notation helps us understand and compare the efficiency of algorithms.
- It's crucial to choose algorithms that can handle large input sizes effectively.
- By thinking about Big O, you can write programs that are faster and use less memory.

08

References

ChatGpt

Grokking Algorithms