Big O Notation and Algorithm Analysis

Dang Quynh Chi SWH00648

COS10009

In COS10009 class, we’ve been briefed through algorithms and Big O notation. In this report, I will delve deeper into the Big O notation and algorithm concept.

“Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann–Landau notation or asymptotic notation.”

— Wikipedia’s definition of Big O notation.

In plain words, Big O notation describes the complexity of your code using algebraic terms.

In Big O notation, we express the upper bound (worst-case scenario) of the time or space complexity of an algorithm using a simplified notation. The notation is represented as O(f(n)), where "f(n)" represents the rate of growth of the algorithm in terms of the input size "n."

Here's an explanation of how Big O notation is used to analyze algorithm efficiency:

1. Time Complexity:

Time complexity (often denoted as T(n)) describes the amount of time an algorithm takes to run as a function of the input size "n." It measures how the running time increases with the size of the input by counting the number of basic operations an algorithm performs relative to the input size "n."

Examples of common time complexity classes:

- O(1) - Constant Time: The algorithm's running time remains constant regardless of the input size. For example, accessing an element in an array by index is O(1) because it takes the same amount of time, regardless of the array's size

- O(log n) - Logarithmic Time: The running time increases logarithmically with the input size. Algorithms with this complexity often halve the search space at each step. Binary search is an example of O(log n) as it repeatedly eliminates half of the remaining elements to find the target value.

- O(n) - Linear Time: The running time increases linearly with the input size. Algorithms with this complexity often perform a constant amount of work for each element in the input. Linear search is an example of O(n) as it checks each element until it finds the target value (or confirms its absence).

- O(n log n) - Linearithmic Time: This complexity often appears in efficient sorting and divide-and-conquer algorithms like Merge Sort and QuickSort.

- O(n^2) - Quadratic Time: The running time grows quadratically with the input size. Algorithms with this complexity often involve nested loops. Selection Sort and Bubble Sort are examples of O(n^2). One typical example is when a loop is inside a loop

- O(n^k) - Polynomial Time: Polynomial time complexities describe algorithms where the running time is proportional to the input size raised to some constant power "k."

- O(2^n) - Exponential Time: The running time doubles with each additional element in the input. Algorithms with exponential time complexity are generally considered inefficient. The "Towers of Hanoi" problem's recursive solution is an example of O(2^n).

- O(n!) - Factorial Time: The running time grows rapidly with the input size and is considered highly inefficient. The "Traveling Salesman Problem" solved by brute force is an example of O(n!).



2. Space Complexity:

Space complexity (denoted as S(n)) describes the amount of memory or space an algorithm requires as a function of the input size "n."

Runtime measurement is one important indicator for performance analysis of an algorithm, but we also need to take the program's memory use into account. This is known as the algorithm's Memory Footprint, also referred to as Space Complexity.

For the performance analysis, we also need to quantify and compare the worst-case theoretical space complexity of the algorithms.

Essentially, it is based on the following two important factors:

First, memory utilization is caused by how the application is implemented. For instance, we can assume that a recursive approach of a certain issue will always reserve more memory than the comparable iterative version.

The second is n, which refers to the input size or the quantity of storage needed for each item. A simple algorithm with a large amount of input, for instance, may use more memory than a sophisticated algorithm with a smaller amount of information.

Examples of common space complexity classes:

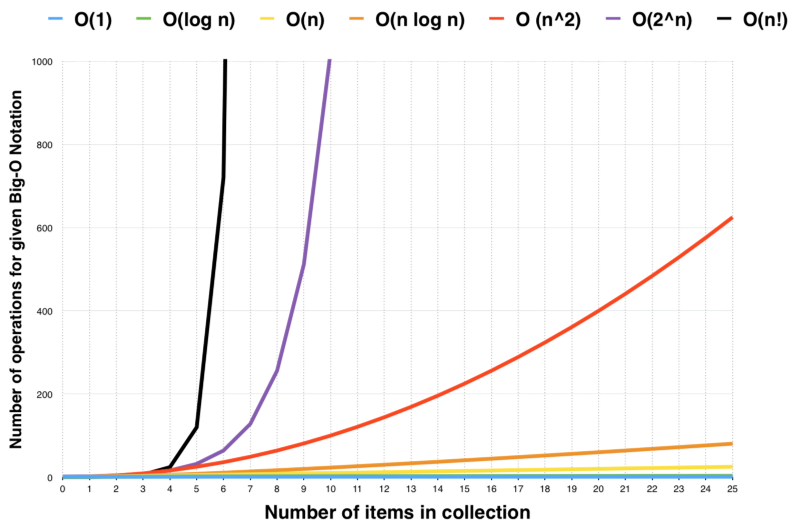
- O(1) - Constant Space: The algorithm uses a constant amount of memory regardless of the input size. Algorithms with no additional data structures or recursion usually have O(1) space complexity.

- O(n) - Linear Space: The algorithm's memory usage grows linearly with the input size. Algorithms that store elements in an array or list without resizing it have O(n) space complexity.

- O(n^2) - Quadratic Space: Algorithms that create a two-dimensional data structure or matrix with dimensions proportional to the input size have O(n^2) space complexity.

- O(log n) - Logarithmic Space: Algorithms with recursive calls or data structures that reduce the problem size by half at each step often have O(log n) space complexity.

Big O notation helps us analyze algorithms' scalability and efficiency, enabling us to choose the most suitable algorithm for a given problem and optimize the performance of our programs. By understanding and comparing different time and space complexity classes, we can make informed decisions when designing algorithms for various tasks.



3. Conclusion:

Tradeoff in Space-Time and Efficiency

Typically, there is a trade-off between runtime performance and optimal memory use.

The space and time efficiency of an algorithm often reach two opposite extremes, with a specific time and space efficiency at each point in between. Therefore, your space efficiency will decrease as your time efficiency increases, and vice versa.

For instance, the Mergesort algorithm operates really quickly yet uses a lot of storage space. On the other hand, Bubble Sort uses the least amount of space but is incredibly sluggish.

Finding an algorithm that operates with a shorter running time and requires less memory space, as this topic comes to a finish, can significantly improve how well an algorithm runs.

Credit:

FreeCodeCamp

<https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/>

Wikipedia

<https://en.wikipedia.org/wiki/Big_O_notation>

GeeksforGeeks

https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/