Asymptotic Notation and Running Time

Hòa Vũ Trọng • 21 Jan 2026

Introduction

Algorithms are the backbone of computer science, but the study of algorithms predates computers (e.g., Euclid algorithm). An algorithm is a set of unambiguous instructions to solve a particular problem. In computer science, analysis of algorithms consists of the following:

- Mathematically prove that the described algorithm is correct.
- Analyzing its efficiency, primarily in terms of running time and memory usage (other factors and computational models are also studied in algorithm research).

We use algorithms every day to efficiently sort large databases, schedule tasks, find the shortest path between two locations, routing traffic, etc.

Running time

The running time is measured based on the number of basic machine instructions such as memory access and pairwise arithmetic operations. Consider the following line of code

```
A[i] = A[i] + 1
```

We have memory access operations (accessing (A[i])) and a pairwise arithmetic operation.

This is too tedious if we want to count exactly (which might not be possible due to different computer architectures). Nonetheless, it is often acceptable to estimate the number of instructions up to a constant factor.

Let us look at the following example that sorts an array (A[1 . . . n]) of integers in increasing order. The algorithm is called selection sort. The idea is simple, scan through (A[1 . . . n]) to find the smallest element and put it in (A[1]), then scan through (A[2 . . . n]) to find the smallest element and put it in (A[2]), and so on.

```
function selection_sort!(A)
    n = length(A)                       # ln 1
    for j in 1:n-1                      # ln 2
        min_idx = j                     # ln 3
        for k in j+1:n                  # ln 4
            if A[k] < A[min_idx]        # ln 5
                min_idx = k             # ln 6
            end
        end
        A[j], A[min_idx] = A[min_idx], A[j] # ln 7
    end
    return A                            # ln 8
end
```

We observe that - Lines 1, 2, and 3 are executed (n-1) times. - Lines 3, 4, and 5 are executed at most (n-1 + n-2 + n-3 + . . . + 1) times respectively. - Line 8 is executed once.

Recall that $1 + 2 + \ldots + n = n(n + 1)/2$. The total running time is approximately

$$T(n) = n + \sum_{j=1}^{n-1} j = n + \frac{(n+1)n}{2} = O(n^2).$$

We will get to the $O(n^2)$ part later, but essentially, the notation says that the running time (number of instructions) grows quadratically in terms of the size of the input.

Exercise: What is the running time of the following dummy algorithm?

```
# A dummy algorithm
function dummy(n)
    for j = 1:n
        x = 0
        for k = 1:(2^j)
            y = 1
        end
    end
end
```

Hint: Use the formula for geometric series

$$a + ar + ar^2 + \cdots + ar^{n-1} = a\frac{1 - r^n}{1 - r}, \quad r \neq 1.$$

Asymptotic notation

We often want to measure the growth of the running time as n increases. For example, when $T(n) = n^2/2 + n/2 - 1$, the term n^2 dictates the growth in terms of n. We often use **asymptotic notation** to denote the running time to make our life easier. We use the notation $f(n) = O(g(n))$ to say that the growth of $f(n)$ is no more than the growth of $g(n)$.

Definition.
We say $f(n) = O(g(n))$ if there exist constants c and n_0 such that

$$f(n) \leq c\, g(n) \quad \text{for all } n \geq n_0.$$

Example. Let $f(n) = 3n^2 + 5n + 2$ and $g(n) = n^2$.

We claim that $f(n) = O(g(n))$.

Choose $c = 4$ and $n_0 = 10$. For all $n \geq 6$,

$$3n^2 + 5n + 2 \leq 4n^2.$$

One can visualize this by plotting

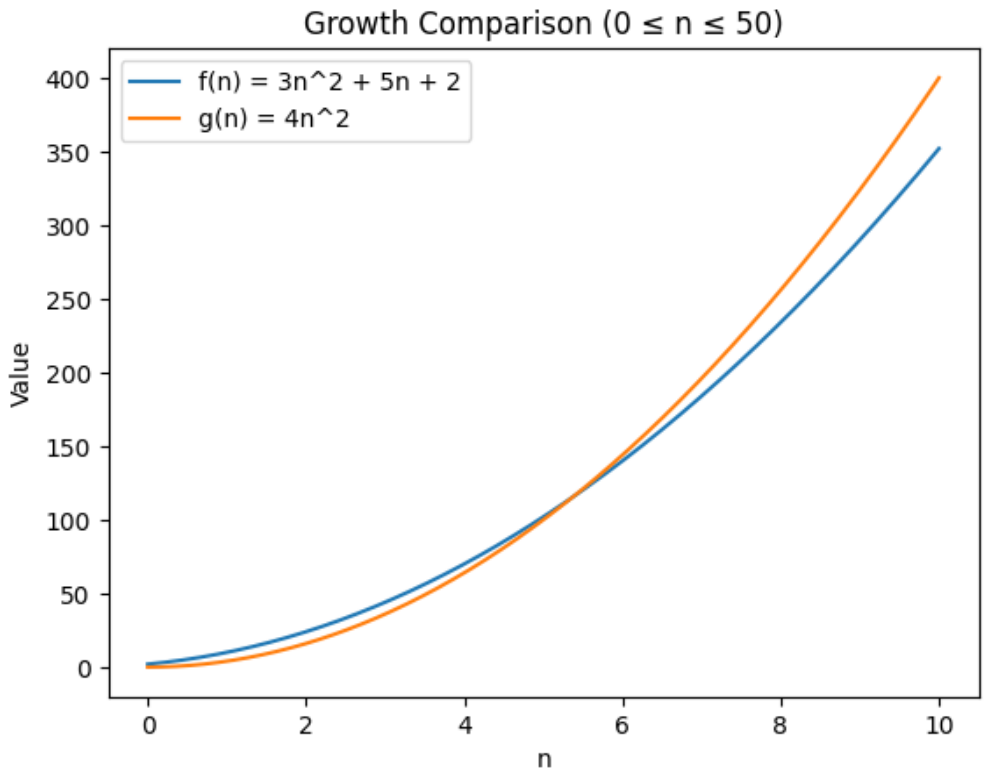

image description

Therefore, there exist constants c and n_0 such that

$$f(n) \leq \underbrace{4}_{c} g(n) \quad \text{for all } n \geq \underbrace{6}_{n_0},$$

and hence $f(n) = O(n^2)$.

Here is another way to show $f(n) = O(g(n))$. Suppose

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = d < \infty,$$

where d is some constant, then $f(n) = O(g(n))$.

Example.
Let $f(n) = 3n^2 + 5n + 2$ and $g(n) = n^2$. Then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \left(3 + \frac{5}{n} + \frac{2}{n^2}\right) = 3 < \infty.$$

Therefore, $f(n) = O(n^2)$.

Example.
Let $f(n) = n + 10\ln n$ and $g(n) = n$. Then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \left(1 + \frac{10\ln n}{n}\right).$$

Using the fact that $\ln n/n \to 0$ as $n \to \infty$, we get

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1 < \infty.$$

Hence, $n + 10\ln n = O(n)$.

If you wonder how we get $\ln n/n \to 0$ as $n \to \infty$, recall from calculus that logarithmic functions grow more slowly than any positive power of n. In particular, we can use L'Hospital's rule:

$$\lim_{n\to\infty} \frac{\ln n}{n} = \lim_{n\to\infty} \frac{(d/dn)(\ln n)}{(d/dn)(n)} = \lim_{n\to\infty} \frac{1/n}{1} = 0.$$

Thus, $\ln n$ grows strictly slower than n, and hence $\ln n/n \to 0$ as $n \to \infty$.

Definition.
We say $f(n) = \Omega(g(n))$ if there exist constants c and n_0 such that

$$f(n) \le c\, g(n) \quad \text{for all } n \ge n_0.$$

Alternatively, this means $g(n) = O(f(n))$.
