Algorithms: Complexity of Algorithms

Analyzing Algorithms

Predict resource utilization

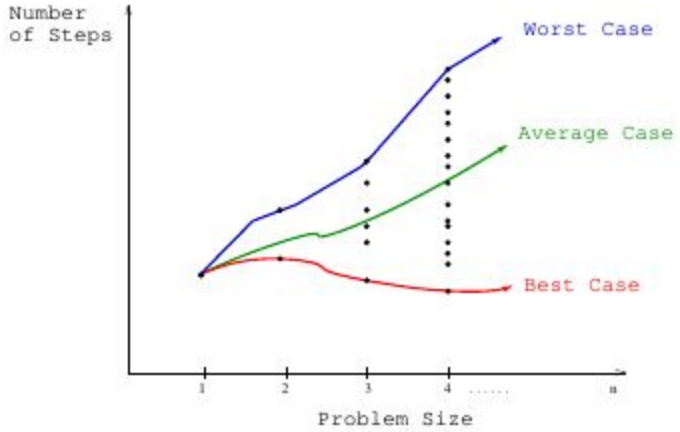
1. Memory (space complexity)

2. Running time (time complexity)

Remark: Really depends on the model of computation (sequential or parallel). We usually assume sequential.

Worst-Case Complexity

The *worst case complexity* of an algorithm is the function defined by the maximum number of steps taken on any instance of size n.



Best-Case and Average-Case Complexity

The *best case complexity* of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.

The *average-case complexity* of the algorithm is the function defined by an average number of steps taken on any instance of size n.

Each of these complexities defines a numerical function: time vs. size!

Our Position on Complexity Analysis

What would the reasoning be on buying a lottery ticket on the basis of best, worst, and average-case complexity?

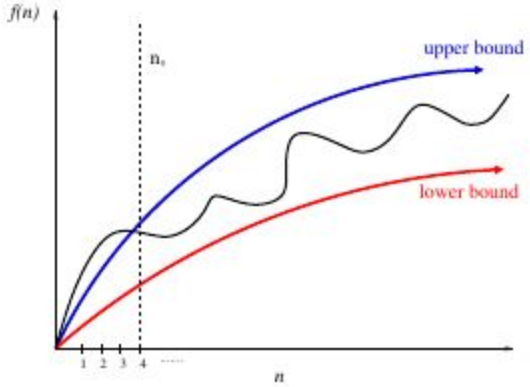
Generally speaking, we will use the worst-case complexity as our preferred measure of algorithm efficiency.

Worst-case analysis is generally easy to do, and "usually" reflects the average case. Assume I am asking for worst-case analysis unless otherwise specified!

Randomized algorithms are of growing importance, and require an average-case type analysis to show off their merits.

Exact Analysis is Hard!

Best, worst, and average case are difficult to deal with because the *precise* function details are very complicated:



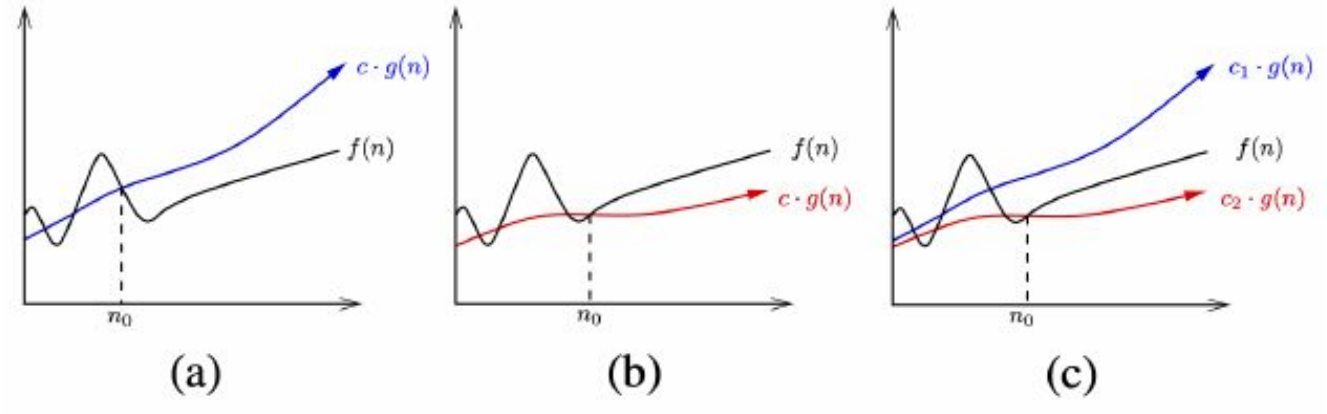
It easier to talk about *upper and lower bounds* of the function. Asymptotic notation (O, Θ, Ω) are as well as we can practically deal with complexity functions.

Names of Bounding Functions

- $g(n) = O(f(n))$ means $C \times f(n)$ is an *upper bound* on $g(n)$.
- $g(n) = \Omega(f(n))$ means $C \times f(n)$ is a *lower bound* on $g(n)$.
- $g(n) = \Theta(f(n))$ means $C_1 \times f(n)$ is an upper bound on $g(n)$ and $C_2 \times f(n)$ is a lower bound on $g(n)$.

C, C_1, and C_2 are all constants independent of n.

O, Ω, and Θ



The definitions imply a constant n_0 *beyond which* they are satisfied. We do not care about small values of n.

Formal Definitions

- $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $c \cdot g(n)$.
- $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $c \cdot g(n)$.
- $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that to the right of n_0, the value of $f(n)$ always lies between $c_1 \cdot g(n)$ and $c_2 \cdot g(n)$ inclusive.