Data Structures and Algorithms
Running time and growth functions
January 18, 2018

Measuring Running Time of Algorithms

One way to measure the running time of an algorithm is to implement it and then study its running time by executing it on various inputs. This approach has the following limitations.

- It is necessary to implement and execute an algorithm to study its running time experimentally.
- Experiments can be done only on a limited set of inputs and may not be indicative of the running time on other inputs that were not included in the experiment.
- It is difficult to compare the efficiency of two algorithms unless they are implemented and executed in the same software and hardware environments.

Analyzing algorithms analytically does not require that the algorithms be implemented, takes into account all possible inputs, and allows us to compare the efficiency of two algorithms independent of the hardware and software environment.

Model of Computation

The model of computation that we use to analyze algorithms is called the *Random Access Machine*(RAM). In this model of computation we assume that high-level operations that are independent of the programming language used take 1 time step. Such high-level operations include the following: performing arithmetic operations, comparing two numbers, assigning a value to a variable, calling a function, returning from a function, and indexing into an array. Note that actually, each of the above high-level operations take a few primitive low-level instructions whose execution time depends on the hardware and software enviroment, but this is bounded by a constant.

Note that in this model we assume that each memory access takes exactly one time step and we have unlimited memory. The model does not take into account whether an item is in the cache or on the disk.

While some of the above assumptions do not hold on a real computer, the assumptions are made to simplify the mathematical analysis of algorithms. Despite being simple, RAM captures the essential behavior of computers and is an excellent model for understanding how an algorithm will perform on a real computer.

Average Case and Worst Case

As we saw in class (Insertion Sort), an algorithm works faster on some inputs than others. How should we express the running time of an algorithm? Let $T(n)$ denote the running time (number of high-level steps) of algorithm on an input of size n. Note that there are 2^n input instances with n bits. Below are three possibilities of inputs that we will consider.

- **A typical input:** The problem with considering a typical input is that different applications will have very different typical inputs.
- **Average Case:** Average case analysis is challenging. It requires us to define a probability distribution on the set of inputs, which is a difficult task.
- **Worst Case:** In this measure we consider the input on which the algorithm performs the slowest. When we say that $T(n)$ is the worst case running time of an algorithm we mean that the algorithm takes no more than $T(n)$ steps for *any* input of size n. In other words, it is an upper bound on the running time of the algorithm for any input. This measure is a nice clean mathematical definition and is easiest to analyze.

Order of Growth

To simplify our analysis we will ignore the lower-order terms in the function $T(n)$ and the multiplicative constants in front. That is, it is the *rate of growth* or the *order of growth* that will be of interest to us. We also ignore the function for small values of n and focus on the *asymptotic* behavior as n becomes very large. Some reasons are as follows.

- The multiplicative constants depends on how fast the machine is and the precise definition of an operation.
- Counting every operation that the algorithm takes is tedious and more work than it is worth.
- It is much more significant whether the time complexity is $T(n) = n^2$ or n^3 than whether it is $T(n) = 2n^2$ or $T(n) = 3n^2$.

Definitions of Asymptotic notations – $O, \Omega, \Theta, o, \omega$

Textbook (Chapter 3).

Example. Prove that $7n - 2 = \Theta(n)$.

Solution. Note that $7n - 2 \leq 14n$. Thus $7n - 2 = O(n)$ follows by setting $c = 14$ and $n_0 = 1$.

To show that $7n - 2 = \Omega(n)$, we need to show that there exists positive constants c and n_0 such that $7n - 2 \geq cn$, for all $n \geq n_0$. This is the same as showing that for all $n \geq n_0$, $(7 - c)n \geq 2$. Setting $c = 1$ and $n_0 = 1$ proves the lower bound. Hence $7n - 2 = \Theta(n)$.

Example. Prove that $10n^3 + 55n \log n + 23 = O(n^3)$.

Solution. The left hand side is at most $88n^3$. Thus setting $c = 88$ and $n_0 = 1$ proves the claim.

Example. Prove that $3^{60} = O(1)$.

Solution. This follows because $3^{60} \leq 3^{60} \cdot 1$, for all $n \geq 0$.

Example. Prove that $5/n = O(1/n)$.

Solution. This follows because $5/n \leq 5 \cdot (1/n)$, for all $n \geq 1$.

Example. Prove that $n^2/8 - 50n = \Theta(n^2)$.

Solution. **First Solution:** $n^2/8 - 50n \leq n^2$. To prove the upperbound, we need to prove that

$$0 \leq \frac{n^2}{8} - 50n \leq c_2 n^2, \forall n \geq n_0$$

Setting $c_2 = 1$ and $n_0 = 800$ satisfies the above inequalities.

To show that $n^2/8 - 50n = \Omega(n^2)$, we need to show that there exists positive constants c_1 and n_0 such that $n^2/8 - 50n \geq c_1 n^2$, for all $n \geq n_0$. This is the same as showing that for all $n \geq n_0$, $(1/8 - c_1)n \geq 50$, or equivalently $(1 - 8c_1)n \geq 400$. Setting $c_1 = 1/16$ and $n_0 = 800$ proves the lower bound.

Hence setting $c_1 = 1/16, c_2 = 1$, and $n_0 = 800$ in the definition of $\Theta(\cdot)$ proves that $n^2/8 - 50n = \Theta(n^2)$.

Second Solution: Note that $\lim_{n\to\infty}(n^2/(n^2/8 - 50n)) = \lim_{n\to\infty} 8n/(n - 400) = \lim_{n\to\infty} 8/(1 - 400/n) = 8$. Thus the claim follows.

Example. Prove that $\lg n = O(n)$.

Solution. We will show using induction on n that $\lg n \leq n$, for all $n \geq 1$.
Base Case: $n = 1$: the claim holds for this case as the left hand side is 0 and the right hand side is 1.
Induction Hypothesis: Assume that $\lg k \leq k$, for some $k \geq 1$.
Induction Step: We want to show that $\lg(k + 1) \leq (k + 1)$. We have

$$\begin{aligned} \text{LHS} &= \lg(k+1) \\ &\leq \lg(2k) \\ &= \lg 2 + \lg k \\ &\leq 1 + k \qquad \text{(using induction hypothesis)} \end{aligned}$$

Example. Prove that $3n^{100} = O(2^n)$.

Solution. We will show using induction on n that $3n^{100} \leq 3(100^{100})(2^n)$, for all $n \geq 200$. That is $c = 3(100^{100})$ and $n_0 = 200$.
Base Case: $n = 200$: the claim holds for this case as the left hand side is $3(100^{100})(2^{100})$ and the right hand side is $3(100^{100})(2^{200})$.

Induction Hypothesis: Assume that $3k^{100} \leq 3(100^{100})(2^k)$, for some $k \geq 200$.
Induction Step: We want to show that $3(k+1)^{100} \leq 3(100^{100})(2^{k+1})$.

$$\begin{aligned}
\text{LHS} &= 3(k+1)^{100} \\
&\leq 3\left(k^{100} + 100k^{99} + \binom{100}{2}k^{98} + \cdots + 1\right) \qquad \text{(using Binomial Theorem)} \\
&\leq 3\left(k^{100} + 100k^{99} + 100^2k^{98} + \cdots + 100^{100}k^0\right)\big) \\
&= 3k^{100}\left(1 + (100/k) + (100/k)^2 + (100/k)^3 + \cdots + (100/k)^{100}\right) \\
&\leq 3(100^{100})(2^k)\sum_{i=0}^{\infty}(1/2)^i \qquad \text{(using induction hypothesis and the fact that } k \geq 200) \\
&\leq 3(100^{100})(2^{k+1})
\end{aligned}$$

Another way to prove the claim is as follows. Consider $\lim_{n\to\infty} 2^n/3n^{100}$. This can be written as

$$\lim_{n\to\infty} 2^n/2^{\lg 3n^{100}} = \lim_{n\to\infty} 2^{n-\log 3n^{100}} = \infty.$$

Example. Prove that $7n-2$ is not $\Omega(n^{10})$.

Solution. Note that $\lim_{n\to\infty} n^{10}/(7n-2) = \infty$ and hence $7n-2 = o(n^{10})$. Thus it cannot be $\Omega(n^{10})$.

We can also prove the claim as follows using the definition of Ω. Assume for the sake of contradiction that $7n-2 = \Omega(n^{10})$. This means that for some constants c and n_0, $7n-2 \geq cn^{10}$, for all $n \geq n_0$. This implies that

$$\begin{aligned}
7n &\geq cn^{10} \\
7 &\geq cn^9
\end{aligned}$$

This is a contraction as the left hand side is a constant and the right had side keeps growing with n. More specifically, the right hand side becomes greater than the left hand side as soon as $n > 2c^{-1/9}$.

Example. Prove that $n^{1+0.0001}$ is not $O(n)$.

Solution. Assume for the sake of contradiction that $n^{1+0.0001} = O(n)$. This means that for some constants c and n_0, $n^{1+0.0001} \leq cn$, for all $n \geq n_0$. This is equivalent to $n^{0.0001} \leq c$, for all $n \geq n_0$ This is impossible as the left hand side grows unboundedly with n. More specifically, when $n > c^{10^4}$, the left hand side becomes greater than c.

We now state without proof some of the properties of asymptotic growth functions.

1. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.
2. If $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ then $f(n) = \Omega(h(n))$.

3. If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $f(n) = \Theta(h(n))$.
4. Suppose f and g are two functions such that for some other function h, we have $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Then $f(n) + g(n) = O(h(n))$.

We now state asymptotic bounds of some common functions.

1. Recall that a polynomial is a function that can be written in the form

   $$a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d \text{ for some integer constant } d > 0 \text{ and } a_d \text{ is non-zero.}$$

   Let f be a polynomial of degree d in which the coefficient $a_d > 0$. Then $f = O(n^d)$.
2. For every $b > 1$ and any real numbers $x, y > 0$, we have $(\log_b n)^x = O(n^y)$. Note that $(\log_b n)^x$ is also written as $\log_b^x n$.
3. For every $r > 1$ and every $d > 0$, we have $n^d = O(r^n)$.

Example. As a result of the above bounds we can conclude that asymptotically, $10^5 \log^{38} n$ grows slower than $n/10^{10}$.

Example. Prove that $n^{10} = o(2^{\lg^3 n})$ and $2^{\lg^3 n} = o(2^n)$.

Solution. Note that

$$2^{\lg^3 n} = (2^{\lg n})^{\lg^2 n} = n^{\lg^2 n}.$$

Since we know that $10 = o(\lg^2 n)$, we conclude that

$$n^{10} = o(2^{\lg^3 n})$$

Since we also know that any poly-logarithmic function in n such as $\lg^3 n$ is asymptotically upper-bounded by a polynomial function in n, we conclude that

$$\lg^3 n = O(n)$$

Actually, $\lg^3 n = o(n)$ and hence

$$2^{\lg^3 n} = o(2^n).$$