Lecture 3

Characterizing Running Times

Slides are based on the textbook and its notes

Overview

- A way to describe behavior of functions *in the limit*. We are studying ***asymptotic*** efficiency.
- Describe *growth* of functions.
- Focus on what is important by abstracting away low-order terms and constant factors.
- How we indicate running times of algorithms.
- A way to compare "sizes" of functions:

$$
\begin{array}{lll}
O & \approx & \leq \\
\Omega & \approx & \geq \\
\Theta & \approx & = \\
o & \approx & < \\
\omega & \approx & >
\end{array}
$$

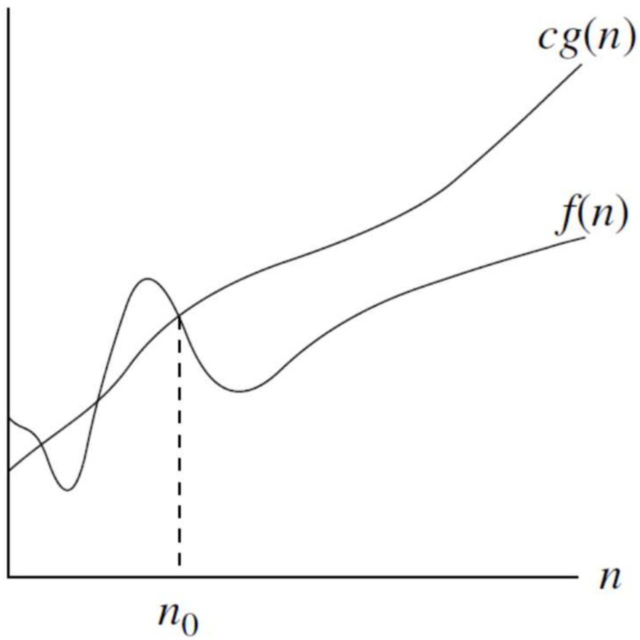
O-notation, Ω-notation, and Θ-notation

- O-notation
  - It characterizes an *upper bound* on the asymptotic behavior of a function: it says that a function grows *no faster* than a certain rate. This rate is based on the highest-order term.
  - For example, $f(n) = 7n^3 + 100n^2 - 20n + 6$ is $O(n^3)$, since the highest-order term is $7n^3$, and therefore the function grows no faster than n^3.
  - The function $f(n)$ is also $O(n^5)$, $O(n^6)$, and $O(n^c)$ for any constant $c \geq 3$.
- Ω-notation
  - It characterizes a *lower bound* on the asymptotic behavior of a function: it says that a function grows *at least as fast* as a certain rate. This rate is again based on the highest-order term.
  - For example, $f(n) = 7n^3 + 100n^2 - 20n + 6$ is $\Omega(n^3)$, since the highest-order term, n^3, grows at least as fast as n^3.
  - The function $f(n)$ is also $\Omega(n^2)$, $\Omega(n)$, and $\Omega(n^c)$ for any constant $c \leq 3$.
- Θ-notation
  - It characterizes a *tight bound* on the asymptotic behavior of a function: it says that a function grows *precisely* at a certain rate, again based on the highest-order term.
  - If a function is both $O(f(n))$ and $\Omega(f(n))$, then a function is $\Theta(f(n))$.

O-notation: formal definition

- O-notation

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\} .$$



Examples of functions in $O(n^2)$:

n^2
$n^2 + n$
$n^2 + 1000n$
$1000n^2 + 1000n$
Also,
n
$n/1000$
$n^{1.99999}$
$n^2/ \lg\lg\lg n$

$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

If $f(n) \in O(g(n))$, we write $f(n) = O(g(n))$ (will precisely explain this soon).

O-notation: formal definition

- **Example 1**: Let us formally prove $4n^2 + 100n + 500 = O(n^2)$.

We need to find positive constants c and n_0 such that $4n^2 + 100n + 500 \leq cn^2$ for all $n \geq n_0$. Dividing both sides by n^2 gives $4 + 100/n + 500/n^2 \leq c$. This inequality is satisfied for many choices of c and n_0. For example, if we choose $n_0 = 1$, then this inequality holds for $c = 604$. If we choose $n_0 = 10$, then $c = 19$ works, and choosing $n_0 = 100$ allows us to use $c = 5.05$.

Therefore, $4n^2 + 100n + 500 = O(n^2)$.

O-notation: formal definition

- **Example 2**: Let us formally prove $n^3 - 100n^2$ does not belong to the set $O(n^2)$.

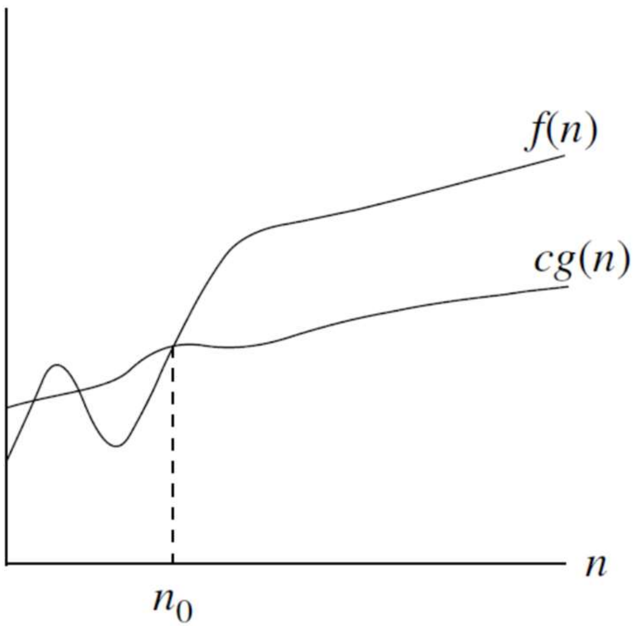
Even though the coefficient of n^2 is a large negative number. If we had $n^3 - 100n^2 = O(n^2)$, then there would be positive constants c and n_0 such that $n^3 - 100n^2 \leq cn^2$ for all $n \geq n_0$. Again, we divide both sides by n^2, giving $n - 100 \leq c$. Regardless of what value we choose for the constant c, this inequality does not hold for any value of $n > c + 100$.

Therefore, $n^3 - 100n^2$ does not belong to the set $O(n^2)$.

Ω-notation: formal definition

- Ω-notation

$$\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \} .$$



$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Examples of functions in $\Omega(n^2)$:

n^2
$n^2 + n$
$n^2 - n$
$1000n^2 + 1000n$
$1000n^2 - 1000n$
Also,
n^3
$n^{2.00001}$
$n^2 \lg \lg \lg n$
2^{2^n}

Ω-notation: formal definition

- **Example**: Let us formally prove

$$4n^2 + 100n + 500 = \Omega(n^2).$$

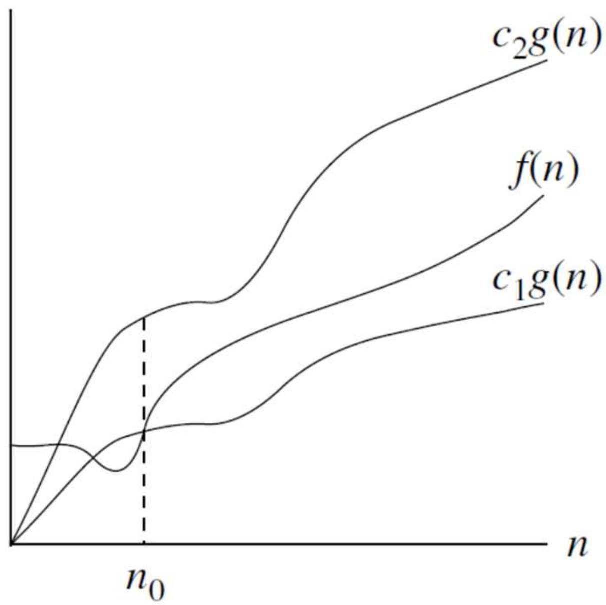
We need to find positive constants c and n_0 such that $4n^2 + 100n + 500 \geq cn^2$ for all $n \geq n_0$. As before, we divide both sides by n^2 gives $4 + 100/n + 500/n^2 \geq c$. This inequality holds when n_0 is any positive integer and $c = 4$.

Therefore, $4n^2 + 100n + 500 = \Omega(n^2)$.

Θ-notation: formal definition

- Θ-notation

  $$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \} .$$



  $g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

- ***Theorem 3.1***: $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- Can express a constant factor as $O(1)$ or $\Theta(1)$, as it is within a constant factor of 1.

Θ-notation: formal definition

- **Example**: Let us formally prove $\frac{1}{2}n(n-1) = \Theta(n^2)$.

First, we prove the right inequality (the upper bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \quad \text{for all } n \ge 0.$$

Second, we prove the left inequality (the lower bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \ge \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \text{ (for all } n \ge 2) = \frac{1}{4}n^2.$$

Hence, we can select $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$, and $n_0 = 2$ to prove that $\frac{1}{2}n(n-1) = \Theta(n^2)$.

Example: Insertion sort

- We will characterize insertion sort's $\Theta(n^2)$ worst-case running time as an example of how to work with asymptotic notation.

- Here is the INSERTION-SORT procedure, from Lecture 2:

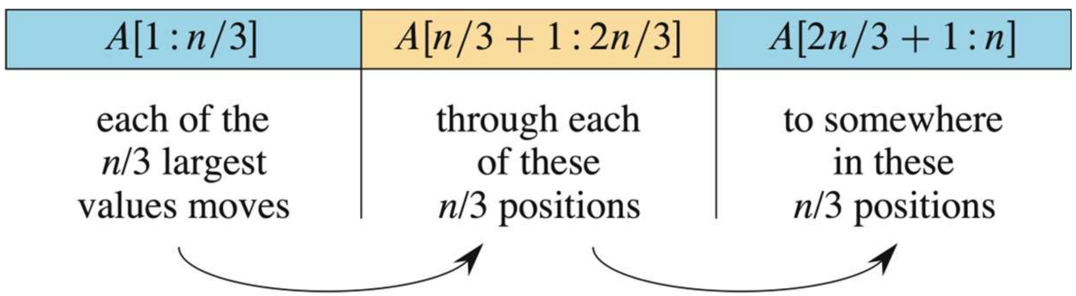
```
INSERTION-SORT(A, n)
for i = 2 to n
    key = A[i]
    // Insert A[i] into the sorted subarray A[1 : i − 1].
    j = i − 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j − 1
    A[j + 1] = key
```

Example: Insertion sort

- First, show that INSERTION-SORT runs in $O(n^2)$ time, regardless of the input:
  - The outer **for** loop runs $n - 1$ times regardless of the values being sorted.
  - The inner **while** loop iterates at most $i - 1$ times.
  - The exact number of iterations the **while** loop makes depends on the values it iterates over, but it will definitely iterate between 0 and $i - 1$ times.
  - Since i is at most n, the total number of iterations of the inner loop is at most $(n - 1)(n - 1)$, which is less than n^2.
- Since each iteration of the inner loop takes constant time, the total time spent in the inner loop is at most cn^2 for some constant c, or $O(n^2)$.

Example: Insertion sort

- Now show that INSERTION-SORT has a worst-case running time of $\Omega(n^2)$ by demonstrating an input that makes the running time be at least some constant times n^2:
  - Observe that for a value to end up k positions to the right of where it started, the line $A[j + 1] = A[j]$ must have been executed k times.
  - Assume that n is a multiple of 3 so that we can divide the array A into groups of $n/3$ positions.



  - Suppose that the input to INSERTION-SORT has the $n/3$ largest values in the first $n/3$ array positions $A[1 : n/3]$. The order within the first $n/3$ positions does not matter.
  - Once the array has been sorted, each of these $n/3$ values will end up somewhere in the last $n/3$ positions $A[2n/3 + 1 : n]$.
  - For that to happen, each of these $n/3$ values must pass through each of the middle $n/3$ positions $A[n/3 + 1 : 2n/3]$.

Example: Insertion sort

- Because at least $n/3$ values must pass through at least $n/3$ positions, the line $A[j + 1] = A[j]$ executes at least $(n/3)(n/3) = n^2/9$ times, which is $\Omega(n^2)$. For this input, INSERTION-SORT takes time $\Omega(n^2)$.

- Since we have shown that INSERTION-SORT runs in $O(n^2)$ time in all cases and that there is an input that makes it take $\Omega(n^2)$ time, we can conclude that the worst-case running time of INSERTION-SORT is $\Theta(n^2)$.

- The constant factors for the upper and lower bounds may differ. That does not matter. The important point is to characterize the worst-case running time to within constant factors.

- We are focusing on just the worst-case running time here, since the best-time running for insertion sort is $\Theta(n)$.

Asymptotic notation and running times

- Need to be careful to use asymptotic notation correctly when characterizing a running time.
- Asymptotic notation describes functions, which in turn describe running time. Must be careful to specify *which* running time.
- For example, the worst-case running time for insertion sort is $O(n^2)$, $\Omega(n^2)$ and $\Theta(n^2)$; all are correct. Prefer to use $\Theta(n^2)$ here, since it is the most precise. The best-case running time for insertion sort is $O(n)$, $\Omega(n)$, and $\Theta(n)$; prefer $\Theta(n)$.
- But *cannot* say that the running time for insertion sort is $\Theta(n^2)$, with "worst-case" omitted. Omitting the case means making a blanket statement that covers *all* cases, and insertion sort does *not* run in $\Theta(n^2)$ time in all cases.
- For merge sort, its running time is $\Theta(n \lg n)$ in all cases, so it is OK to omit which case.

Asymptotic notation and running times

- Common errors:
  - Conflating O-notation with Θ-notation by using O-notation to indicate an asymptotically tight bound. O-notation gives only an asymptotic upper bound.
  - Saying "an $O(n \lg n)$-time algorithm runs faster than an $O(n^2)$-time algorithm" is not necessarily true.
  - An algorithm that runs in $\Theta(n)$ time also runs in $O(n^2)$ time. If you really mean an asymptotically tight bound, then use Θ-notation.
  - Use the simplest and most precise asymptotic notation that applies. Suppose that an algorithm's running time is $3n^2 + 20n$. Best to say that it is $\Theta(n^2)$. Could say it is $O(n^3)$, but that is less precise. Could say that it is $\Theta(3n^2 + 20n)$, but that obscures the order of growth.

Asymptotic notation in equations

- ***When on right-hand side***
  - $O(n^2)$ stands for some anonymous function in the set $O(n^2)$.
  - $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means $2n^2 + 3n + 1 = 2n^2 + f(n)$ *for some* $f(n) \in \Theta(n)$. In particular, $f(n) = 3n + 1$.
  - Interpret the number of anonymous functions as equaling the number of times the asymptotic notation appears:

$\sum_{i=1}^{n} O(i)$ OK: 1 anonymous function

$O(1) + O(2) + \cdots + O(n)$ not OK: n hidden constants $\Rightarrow$ no clean interpretation

Asymptotic notation in equations

- ***When on left-hand side***
  - In some cases, asymptotic notation appears on the left-hand side of an equation: $2n^2 + \Theta(n) = \Theta(n^2)$.
  - Interpret such equations with the rule: Not matter how the anonymous functions are chosen on the left-hand side, there is a way to choose the anonymous functions on the right-hand side to make the equation valid.
  - Thus, our example means that *for all* functions $f(n) \in \Theta(n)$, there is some function $g(n) \in \Theta(n^2)$ such that $2n^2 + f(n) = g(n)$ for all n. In other words, the right-hand side of an equation provides a coarser level of details than the left-hand side.
  - We can chain together: $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$.
    - By the rules above: interpret each equation separately.
    - First equation: There exists function $f(n) \in \Theta(n)$ such that $2n^2 + 3n + 1 = 2n^2 + f(n)$ for all n.
    - Second equation: For all $g(n) \in \Theta(n)$ (such as the $f(n)$ used to make the first equation hold), there exists $h(n) \in \Theta(n^2)$ such that $2n^2 + g(n) = h(n)$.

Standard notations and common functions

- **Monotonicity**
  - $f(n)$ is ***monotonically increasing*** if $m \leq n \Rightarrow f(m) \leq f(n)$.
  - $f(n)$ is ***monotonically decreasing*** if $m \leq n \Rightarrow f(m) \geq f(n)$.
  - $f(n)$ is ***strictly increasing*** if $m < n \Rightarrow f(m) < f(n)$.
  - $f(n)$ is ***strictly decreasing*** if $m < n \Rightarrow f(m) > f(n)$.
- **Exponentials**
  - Useful identities:

    $a^{-1} = 1/a$,

    $(a^m)^n = a^{mn}$,

    $a^m a^n = a^{m+n}$.
  - A surprisingly useful inequality: for all real x,

    $e^x \geq 1 + x$.

    As x gets closer to 0, e^x gets closer to $1 + x$.

Standard notations and common functions

- **Logarithms**
  - Notations:

$$\lg n = \log_2 n \quad \text{(binary logarithm)},$$
$$\ln n = \log_e n \quad \text{(natural logarithm)},$$
$$\lg^k n = (\lg n)^k \quad \text{(exponentiation)},$$
$$\lg \lg n = \lg(\lg n) \quad \text{(composition)}.$$

  - Logarithm functions apply only to the next term in the formula, so that $\lg n + k$ means $(\lg n) + k$, and not $\lg(n + k)$.
  - In the expression $\log_b a$:
    - Hold b constant => the expression is strictly increasing as a increases.
    - Hold a constant => the expression is strictly decreasing as b increases.

Standard notations and common functions

- Useful identities for all real $a > 0$, $b > 0$, $c > 0$, and n, and where logarithm bases are not 1:

$$
\begin{aligned}
a &= b^{\log_b a} , \\
\log_c(ab) &= \log_c a + \log_c b , \\
\log_b a^n &= n \log_b a , \\
\log_b a &= \frac{\log_c a}{\log_c b} , \\
\log_b(1/a) &= -\log_b a , \\
\log_b a &= \frac{1}{\log_a b} , \\
a^{\log_b c} &= c^{\log_b a} .
\end{aligned}
$$

- Changing the base of a logarithm from one constant to another only changes the value by a constant factor, so we usually do not worry about logarithm bases in asymptotic notation.

Factorials

- $n! = 1 \cdot 2 \cdot 3 \cdot n$. Special case: $0! = 1$. Can use ***Stirling's approximation***,

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) ,$$

To derive that $\lg(n!) = \Theta(n \lg n)$.

Reading

- Sections 3.1 ~ 3.3

Written exercise 3.1

- Using reasoning similar to what we used for insertion sort, analyze the running time of the selection sort algorithm from Written exercise 2.2.

Written exercise 3.2

- Explain why the statement, "The running time of algorithm A is at least $O(n^2)$," is meaningless.

Written exercise 3.3

- Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

  (Hint: Use O-notation definition to answer them.)

Written exercise 3.4

- Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.