Outline

Announcements:
- PA1 will be online later today; Due: Thu, 10/4, 1:00 pm, online submission

Last time:
- ComputeRank in $N \log N$ time
- BubbleSort
- Review of asymptotic analysis
- Empirical performance evaluation

Today:
- Review of asymptotic analysis
- Review of foundational data structures
Asymptotic Algorithm Analysis

An algorithm with complexity \( f(n) \) (e.g., \( f(n) = n \), \( f(n) = n^2 \), etc.), is said to be not slower than another algorithm with complexity \( g(n) \) if \( f(n) \) is bounded by \( g(n) \) for large \( n \) (See Fig. 4.1)

Commonly written as \( f(n) = O(g(n)) \) (read: \( f(n) \) is big-Oh \( g(n) \)), a.k.a. the asymptotic (or big-Oh) notation

**Definitions:**

\[ f(n) = O(g(n)) \text{ iff } \exists c > 0, n_o \geq 0 \mid \forall n, n \geq n_o, f(n) \leq c \cdot g(n), \]

\[ O(g(n)) = \{ f(n) : \exists c > 0, n_o \geq 0 \mid \forall n, n \geq n_o, 0 \leq f(n) \leq c \cdot g(n) \}, \]

and \( c \) is a constant, i.e., it doesn’t change with \( n \)

So more accurately: \( f(n) \in O(g(n)) \)

But NOT \( f(n) \leq O(g(n)) \)
Big-Oh Fallacies (Part I)

**Fallacy 0:** If $f(n) = O(g(n)) \Rightarrow f(n) = g(n)$. (**NOT**)

**Fallacy 1:** Let $f_1(n) = h(n^2)$ and $f_2(n) = h(n^2) \Rightarrow f_1(n) = f_2(n)$. Therefore if $f_1(n) = O(n^2)$ and $f_2(n) = O(n^2) \Rightarrow f_1(n) = f_2(n)$. (**NOT**) (See Fig. 4.1.)

**Fallacy 2:** $f(n) = O(g(n)) \Rightarrow g(n) = O^{-1}(f(n))$. (**NOT**) (There’s no such thing as a $O^{-1}()$ function.)

Sugih Jamin (jamin@eecs.umich.edu)
“Landmark” Functions

Well-known functions you compare the complexity of your algorithm to, in order of increasing complexity (see Figs. 4.2, 4.3):

- constant: $O(1)$
- logarithmic: $O(\log n)$
- square root: $O(\sqrt{n})$
- linear (or Oh-n): $O(n)$
- n-log-n: $O(n \log n)$
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- in general, polynomial: $O(n^k)$, $k \geq 1$
- exponential: $O(a^n)$, $a > 1$, usually $a \approx 2$
- factorial: $O(n!)$
Tight Bound

While \( f(n) = O(g(n)) \not\Rightarrow f(n) = g(n) \),
you nevertheless want to pick a \( g(n) \) as close to \( f(n) \) as possible
(then it is called a “tight” bound)
Big-Oh Rules

Rule 1: \( f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n)), \) then 
\[ f_1(n) + f_2(n) = O(max(g_1(n), g_2(n))) \]
Example: \( f_1(n) = n^3 \in O(n^3), \) 
\( f_2(n) = n^2 \in O(n^2), \) then \( f_1(n) + f_2(n) = O(n^3). \)

Rule 2: \( f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n)), \) then 
\[ f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n)) \]
If your code calls a function within a loop, the complexity of your code is the product of your code’s complexity times the complexity of the function you call.

Rule 3: \( f(n) = O(g(n)), g(n) = O(h(n)) \) then \( f(n) = O(h(n)). \)
Big-Oh Fallacies (Part II)

**Fallacy 3:** Let \( f(n) = g_1(n) \cdot g_2(n) \).
If \( f(n) \leq c \cdot g_1(n) \) and \( c = g_2(n) \), is \( f(n) = O(g_1(n)) \)?

**Fallacy 4:** Let \( f_1(n) = O(g_1(n)) \), \( f_2(n) = O(g_2(n)) \).
Does \( g_1(n) < g_2(n) \Rightarrow f_1(n) < f_2(n) \)?
Siblings of Big-Oh

**big-Omega(Ω())**: lower bound

For \( f(n) > 0, \forall n \geq 0, f(n) = \Omega(g(n)) \) if

\[ \exists n_o > 0, c > 0 | \forall n \geq n_o, f(n) \geq c \cdot g(n). \]

\( h_1(n) = O(h_2(n)) \Leftrightarrow h_2(n) = \Omega(h_1(n)). \)

**big-Theta(Θ())**: 

\( f(n) = \Theta(g(n)) \) iff \( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \).

\( f(n) \) grows as fast as \( g(n) \).
Cousins of Big-Oh

**little-o(\(o()\)):** \(f(n) = o(g(n))\) if \(f(n) = O(g(n))\) but \(f(n) \neq \Theta(g(n))\). That is, if \(g(n)\) is not an asymptotically tight upper bound of \(f(n)\). In long hand:

\[ f(n) = o(g(n)) \text{ if } \forall c > 0 \mid \forall n : n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n). \]

In contrast to \(O()\), \(o()\) is **forall** \(c > 0\), whereas \(O()\) only requires **there exists** \(c > 0\).

Example: \(2n^2 = O(n^2)\) is asymptotically tight, but \(2n = O(n^2)\) is not. So \(O()\) is sloppier than \(o()\), which is why we use it more often!

**little-omega(\(\omega()\)):** \(f(n) = \omega(g(n))\) iff \(g(n) = o(f(n))\).
Relatives of Big-Oh (cont)

Does \( f(n) = \Theta(g(n)) \Rightarrow g(n) = \Theta(f(n)) \)?

Does \( f(n) = \Theta(g(n)) \Rightarrow f(n) = g(n) \)?

Does \( f(n) = \Theta(g(n)) \Rightarrow f(n) \) is in the same order as \( g(n) \)?
In the Limit (FYI)

$O()$: $f(n) = O(g(n)) \iff f(n) \leq c_1 \cdot g(n)$ and $\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c_1$

$\Omega()$: $f(n) = \Omega(g(n)) \iff f(n) \geq c_2 \cdot g(n)$ and $\lim_{n \to \infty} \frac{g(n)}{f(n)} \leq c_2$

$\Theta()$: $f(n) = \Theta(g(n)) \iff$ both $\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c_1$ and $\lim_{n \to \infty} \frac{g(n)}{f(n)} \leq c_2$

$o()$: $f(n) = o(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$\omega()$: $f(n) = \omega(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$
Summary

- asymptotic analysis deals only with LARGE $n$
- $O()$ provides a short-hand to express upper bound, it is not an exact notation
- be careful how big $c$ is
- be careful how big $n_o$ must be
- run experiments to check the common case behavior
- trade-off space vs. time
- be scalable, but optimize only when and where necessary (don’t lose the forest . . .)
- think outside the box

Done with Preiss Chs. 2 and 3, with the exceptions per Syllabus

Sugih Jamin (jamin@eecs.umich.edu)
Evaluation Questions

1. Let \( f(n) = \frac{(n - 1)n}{2}, c = \frac{n}{2} \), can we say that \( f(n) = O(n) \)? If so, why? If not, what is the big-Oh of \( f(n) \)? Why?

2. Is \( \log n = O(n) \)? \( \log n = O(n^2) \)? Which is a tighter bound?

3. Given 4 consecutive statements: \( S_1; S_2; S_3; S_4 \). Let \( S_1 = O(\log n), S_2 = O(\log n^3), S_3 = O(n), S_4 = O(3n) \). What is the big-Oh time complexity of the four consecutive statements together? Prove it mathematically using the definition of big-Oh.

4. You ran two programs to completion. Both have running time of 10 ms. Can you say that the two programs have the same big-Oh time complexity? Why or why not?
Foundational Data Structures

Array and linked list

Why are they called *foundational* data structures?

What are *abstract data types*? For example?
Array

What is an array?

Name 2 advantages of using an array.

What are the disadvantages of using array?
Linked List

What is a linked list?

Why use linked list?

Name three disadvantages of linked list
Array vs. Linked List

Timing complexity of:

- find_length:
- prepend
- append
- delete_last
- access_i_th
- search(key)
- replace_i_th
- insert_after_i_th
- delete_i_th

Sugih Jamin (jamin@eecs.umich.edu)
Linked List Optimization

- tail pointer (Section 4.2): advantage?
- circular list with or without sentinel (Section 4.2): advantage?
- freelist, with elements allocated in batches (Section 13.2)
Common Bugs

Array:

- index variable not initialized
- bounds not checked
- pointers in array not de-allocated (memory leak)
- when moved (or realloc-ed), pointers to array elements not moved

Linked-list:

- head and tail not initialized to NULL
- extraction or insertion dangles pointer  
  (delete last, prev→next not set to NULL)
- deallocation doesn’t walk down list (memory leak)
- accessing de-allocated element