Lecture #23: Complexity and Orders of Growth, contd.

Announcements:

- UCB Startup Fair, presented by CSUA, HKN, and IEEE.
  Bring resumes; find a job or internship!
  Tuesday, March 13 12-4pm in MLK Pauley Ballroom.

Review of Notation

- $O(f)$ is the set of functions that *eventually grow no faster than f*:

$$O(f) \stackrel{\text{def}}{=} \{g \textit{ such that } |g(x)| \leq p_g \cdot |f(x)| \text{ for all } x \geq M_g\}$$

  , where p_g and M_g are constants (possibly different for each g).

- $\Omega(f)$ is the set of functions that *eventually grow at least as fast as f*:

$$\Omega(f) \stackrel{\text{def}}{=} \{g \textit{ such that } |g(x)| >= p_g \cdot |f(x)| \text{ for all } x \geq M_g\}$$

  .

- Implies that

$$g \in O(f) \text{ iff } f \in \Omega(g)$$

  .

- Finally, $\Theta(f)$ is the set of functions *eventually that grows like f*:

$$\Theta(f) \stackrel{\text{def}}{=} O(f) \cap O(f)$$

Notational Quirks

- We'll sometimes write things like $f \in O(g)$ even when f and g are functions of something non-numeric (like lists). In that case, when we say $x > M$ in the definition of $O(\cdot)$, we are referring to some measure of x's size (like length).
- If $E_1(x)$ and $E_2(x)$ are two expressions involving x, we usually abbreviate $\lambda x : E_1(x) \in O(\lambda x : E_2(x))$ as just $E_1(x) \in O(E_2(x))$. For example, $n + 1 \in O(n^2)$.
- I write $f(x) \in O(g(x))$ where others write $f(x) = O(g(x))$, because the latter doesn't make sense.

Example: Linear Search

- Consider the following search function:

```
def near(L, x, delta):
    """True iff X differs from some member of sequence L by no
    more than DELTA."""
    for y in L:
        if abs(x-y) <= delta:
            return True
    return False
```

- There's a lot here we don't know:
  - How long is sequence L?
  - Where in L is x (if it is)?
  - What kind of numbers are in L and how long do they take to compare?
  - How long do abs and subtract take?
  - How long does it take to create an iterator for L and how long does its __next__ operation take?
- So what can we meaningfully say about complexity of near?

What to Measure?

- If we want general answers, we have to introduce some "strategic vagueness."
- Instead of looking at times, we can consider number of "operations." Which?
- The total time consists of
  1. Some fixed overhead to start the function and begin the loop.
  2. Per-iteration costs: subtraction, abs, __next__, <=
  3. Some cost to end the loop.
  4. Some cost to return.
- So we can collect total operations into one "fixed-cost operation" (items 1, 3, 4), plus M(L) "loop operations" (item 2), where $M(L)$ is the number of items in L up to and including the y that comes within delta of x (or the length of L if no match).

What Does an "Operation" Cost?

- But these "operations" are of different kinds and complexities, so what do we really know?
- Assuming that each operation represents some range of possible minimum and maximum values (constants), we can say that

$$\textit{min-fixed-cost} + M(\mathsf{L}) \times \textit{min-loop-cost}$$
$$\leq$$
$$C_{\rm near}(L)$$
$$\leq$$
$$\textit{max-fixed-cost} + M(\mathsf{L}) \times \textit{max-loop-cost}$$

where $C_{\rm near}(L)$ is the cost of near on a list where the program has to look at $M(L)$ items.

Using Asymptotic Estimates

- We have a rather clumsy description:

$$\textit{min-fixed-cost} + M(L) \times \textit{min-loop-cost} \leq C_{\text{near}}(L)$$
$$\leq \textit{max-fixed-cost} + M(L) \times \textit{max-loop-cost}$$

- Claim: we can state this more cleanly as $C_{\text{near}}(L) \in O(M(L))$ and $C_{\text{near}}(L) \in \Omega(M(L))$, or even more concisely: $C_{\text{near}}(L) \in \Theta(M(L))$.
- Why? $C_{\text{near}}(M(L)) \in O(M(L))$ if $C_{\text{near}}(M(L)) \leq K \cdot M(L)$ for sufficiently large $M(L)$, by definition.
- And if if K_1 and K_2 are any (non-negative) constants, then $K_1 + K_2 \cdot M(L) \leq (K_1 + K_2) \cdot M(L)$ for $M(L) > 1$.
- Likewise, $K_1 + K_2 \cdot M(L) \geq K_2 \cdot M(L)$ for $M > 0$.
- And we can go even farther. If the sequence, L, has length $N(L)$, then we know that $M(L) \leq N(L)$. Therefore, we can say $C_{\text{near}}(L) \in O(N(L))$.
- Is $C_{\text{near}}(L) \in \Omega(N(L))$?

Using Asymptotic Estimates

- We have a rather clumsy description:

$$\begin{aligned} & \textit{min-fixed-cost} + M(L) \times \textit{min-loop-cost} \le C_{\rm near}(L) \\ \le \; & \textit{max-fixed-cost} + M(L) \times \textit{max-loop-cost} \end{aligned}$$

- Claim: we can state this more cleanly as $C_{\rm near}(L) \in O(M(L))$ and $C_{\rm near}(L) \in \Omega(M(L))$, or even more concisely: $C_{\rm near}(L) \in \Theta(M(L))$.
- Why? $C_{\rm near}(M(L)) \in O(M(L))$ if $C_{\rm near}(M(L)) \le K \cdot M(L)$ for sufficiently large $M(L)$, by definition.
- And if if K_1 and K_2 are any (non-negative) constants, then $K_1 + K_2 \cdot M(L) \le (K_1 + K_2) \cdot M(L)$ for $M(L) > 1$.
- Likewise, $K_1 + K_2 \cdot M(L) \ge K_2 \cdot M(L)$ for $M > 0$.
- And we can go even farther. If the sequence, L, has length $N(L)$, then we know that $M(L) \le N(L)$. Therefore, we can say $C_{\rm near}(L) \in O(N(L))$.
- Is $C_{\rm near}(L) \in \Omega(N(L))$? No: can only say $C_{\rm near}(L) \in \Omega(1)$.

Best/Worst Cases

- We can simplify still further by not trying to give results for particular inputs, but instead giving summary results for *all inputs of the same "size."*
- Here, "size" depends on the problem: could be magnitude, length (of list), cardinality (of set), etc.
- Since we don't consider specific inputs, we have to be less precise.
- Typically, the figure of interest is the *worst case over all inputs of the same size.*
- Also makes sense to talk about the *best case* over all inputs of the same size, or the *average case* over all inputs of the same size (weighted by likelihood). These are rarer, though.
- From preceding discussion, since $C_{\rm near}(N(L)) \in O(N(L))$, it follows that $C_{\rm wc}(N) \in O(N)$, where $C_{\rm wc}(N)$ is "worst-case cost of near over all lists of size N."

Best of the Worst

- We just saw that $C_{\rm wc}(N) \in O(N)$.
- But in addition, it's also clear that $C_{\rm wc}(N) \in \Omega(N)$.
- So we can say, most concisely, $C_{\rm wc}(N) \in \Theta(N)$.
- Generally, when a worst-case time is not $\Theta(\cdot)$, it indicates either that
  - We don't know (haven't proved) what the worst case really is, just put limits on it, or
    - Most often happens when we talk about the worst-case for a *problem:* "what's the worst case for the best possible algorithm?"
  - We know what the worst-case time is, but it's not an easy formula, so we settle for approximations that are easier to deal with.

Example: A Nested Loop

- Consider:

```
def are_duplicates(L):
    for i in range(len(L)-1):
        for j in range(i+1, len(L)):
            if L[i] == L[j]:
                return True
    return False
```

- What can we say about $C(L)$, the cost of computing are_duplicates on L?
- How about $C_{\mathrm{wc}}(N)$, the worst-case cost of running are_duplicates over all sequences of length N?

Example: A Nested Loop (II)

- Ans: Worst case is no duplicates. Outer loop runs len(L)-1 times. Each time, the inner loop runs len(L)-i-1 times. So total time is proportional to $(N-2)+(N-3)+\ldots+1=(N-1)(N-2)/2 \in \Theta(N^2)$, where $N=N(L)$ is the length of L.
- Best case is first two elements are duplicates. Running time is $\Theta(1)$ (i.e., bounded by constant).
- So, $C(L) \in O(N(L)^2)$, $C(L) \in \Omega(1)$,
- And $C_{\rm wc}(N) \in \Theta(N^2)$.

Example: A Tricky Nested Loop

- What can we say about this one (assume pred counts as one constant-time operation.)

```
def is_unduplicated(L, pred):
    """True iff the first x in L such that pred(x) is not
    a duplicate. Also true if there is no x with pred(x)."""
    i = 0
    while i < len(L):
        x = L[i]
        i += 1
        if pred(x):
            while i < len(L):
                if x == L[i]:
                    return False
                i += 1
    return True
```

Example: A Tricky Nested Loop (II)

- In this case, despite the nested loop, we read each element of L at most once.
- Best case is that pred(L[0]) and L[0]=L[1].
- So $C(L) \in O(N(L))$, $C(L) \in \Omega(1)$.
- And $C_{\text{wc}}(N) \in \Theta(N)$.

Some Useful Properties

- We've already seen that $\Theta(K_0N + K_1) = \Theta(N)$ (K, k, K_i here and elsewhere are constants).
- $\Theta(N^k + N^{k-1}) = \Theta(N^k)$. Why?
- $\Theta(|f(N)| + |g(N)|) = \Theta(\max(|f(N)|, |g(N)|))$. Why?
- $\Theta(\log_a N) = \Theta(\log_b N)$. Why? (As a result, we usually use $\log_2 N = \lg N$ for all logarithms.)
- Tricky: why *isn't* $\Theta(f(N) + g(N)) = \Theta(\max(f(N), g(N)))$?
- $\Theta(N^{k_1}) \subset \Theta(k_2^N)$, if $k_2 > 1$. Why?

More Programs

- How long does the tree_find program (search binary tree) take in the worst case
  - 1. As a function of D, the depth of the tree?
  - 2. As a function of N, the number of keys in the tree?
  - 3. As a function of D if the tree is as shallow as possible for the amount of data?
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?
- How about the gen_tree_find program from HW#8? Consider all trees where the inner nodes all have *at least* $K_1 > 2$ children and at most K_2 (both constants). What is the worst-case time to search as a function of N?

More Programs

- How long does the tree_find program (search binary tree) take in the worst case
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree?
  - 3. As a function of D if the tree is as shallow as possible for the amount of data?
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?
- How about the gen_tree_find program from HW#8? Consider all trees where the inner nodes all have *at least* $K_1 > 2$ children and at most K_2 (both constants). What is the worst-case time to search as a function of N?

More Programs

- How long does the tree_find program (search binary tree) take in the worst case
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data?
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?
- How about the gen_tree_find program from HW#8? Consider all trees where the inner nodes all have *at least* $K_1 > 2$ children and at most K_2 (both constants). What is the worst-case time to search as a function of N?

More Programs

- How long does the tree_find program (search binary tree) take in the worst case
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?
- How about the gen_tree_find program from HW#8? Consider all trees where the inner nodes all have *at least* $K_1 > 2$ children and at most K_2 (both constants). What is the worst-case time to search as a function of N?

More Programs

- How long does the tree_find program (search binary tree) take in the worst case
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of N if the tree is as shallow as possible for the amount of data? $\Theta(\lg N)$
- How about the gen_tree_find program from HW#8? Consider all trees where the inner nodes all have *at least* $K_1 > 2$ children and at most K_2 (both constants). What is the worst-case time to search as a function of N?

More Programs

- How long does the tree_find program (search binary tree) take in the worst case
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of N if the tree is as shallow as possible for the amount of data? $\Theta(\lg N)$
- How about the gen_tree_find program from HW#8? Consider all trees where the inner nodes all have *at least* $K_1 > 2$ children and at most K_2 (both constants). What is the worst-case time to search as a function of N? $\Theta(\lg N)$

Fast Growth

- Consider Hackenmax from Test#2 (with some name changes):

```
def Hakenmax(board, X, Y, N):
    if N <= 0:
        return 0
    else:
        return board(X, Y) \
               + max(Hakenmax(board, X+1, Y, N-1),
                     Hakenmax(board, X, Y+1, N-1))
```

- Time clearly depends on N. Counting calls to board, $C(N)$,the cost of calling Hackenmax(board,X,Y,N), is

$$C(N) = \begin{cases} 0, & \text{for } N \le 0 \\ 1 + 2C(N-1), & \text{otherwise.} \end{cases}$$

- Using simple-minded expansion,

$$C(N) = 1{+}2C(N{-}1) = 1{+}2{+}4C(N{-}2) = \ldots = 1{+}2{+}4{+}8{+}\ldots{+}2^{N-1} \in \Theta(2^N).$$

Some Intuition on Meaning of Growth

- How big a problem can you solve in a *given* time?
- In the following table, left column shows time in microseconds to solve a *given* problem as a function of problem size N (assuming perfect scaling and that problem size 1 takes 1μsec).
- Entries show the *size of problem* that can be solved in a second, hour, month (31 days), and century, for various relationships between time required and problem size.
- $N =$ problem size

Time (μsec) for problem size N	**Max N Possible in** 1 second	1 hour	1 month	1 century
$\lg N$	10^{300000}	$10^{1000000000}$	$10^{8\cdot 10^{11}}$	$10^{9\cdot 10^{14}}$
N	10^6	$3.6 \cdot 10^9$	$2.7 \cdot 10^{12}$	$3.2 \cdot 10^{15}$
$N \lg N$	63000	$1.3 \cdot 10^8$	$7.4 \cdot 10^{10}$	$6.9 \cdot 10^{13}$
N^2	1000	60000	$1.6 \cdot 10^6$	$5.6 \cdot 10^7$
N^3	100	1500	14000	150000
2^N	20	32	41	51