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

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.

Review of Notation

- \( O(f) \) is the set of functions that eventually grow no faster than \( f \):
  \[ O(f) \triangleq \{ g \text{ such that } |g(x)| \leq p_0 \cdot |f(x)| \text{ for all } x \geq M \} \]
  , where \( p_0 \) and \( M \) are constants (possibly different for each \( g \)).
- \( \Omega(f) \) is the set of functions that eventually grow at least as fast as \( f \):
  \[ \Omega(f) \triangleq \{ g \text{ such that } |g(x)| \geq p_0 \cdot |f(x)| \text{ for all } x \geq M \} \]
  .
- Implies that
  \[ g \in O(f) \iff f \in \Omega(g) \]
  .
- Finally, \( \Theta(f) \) is the set of functions eventually that grows like \( f \):
  \[ \Theta(f) \triangleq O(f) \cap \Omega(f) \]

Example: Linear Search

- Consider the following search function:
  ```python
  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 \( \text{abs} \) and \( \text{subtract} \) 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 \( \text{near} \)?

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: \( \text{abs} \), \( \_\_next\_ \), <=
  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 \( \text{delta of } x \) (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
  \[
  \begin{align*}
  \text{min-fixed-cost} + M(L) \times \text{min-loop-cost} & \leq C_{\text{total}}(L) \\
  & \leq \text{max-fixed-cost} + M(L) \times \text{max-loop-cost}
  \end{align*}
  \]
  where \( C_{\text{total}}(L) \) is the cost of \( \text{near} \) on a list where the program has to look at \( M(L) \) items.
Using Asymptotic Estimates

- We have a rather clumsy description:
  
  $\min\text{-fixed-cost} + M(L) \times \min\text{-loop-cost} \leq C_{\text{wc}}(L)$
  
  $\leq \max\text{-fixed-cost} + M(L) \times \max\text{-loop-cost}$

- Claim: we can state this more cleanly as $C_{\text{wc}}(L) \in O(M(L))$ and $C_{\text{wc}}(L) \in \Omega(M(L))$, or even more concisely: $C_{\text{wc}}(L) \in \Theta(M(L))$.

- Why? $C_{\text{wc}}(L) \in O(M(L))$ if $C_{\text{wc}}(L) \leq K \cdot M(L)$ for sufficiently large $M(L)$, by definition.

- And 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 further. If the sequence, $L$, has length $N(L)$, then we know that $M(L) \leq N(L)$. Therefore, we can say $C_{\text{wc}}(L) \in O(N(L))$.

- Is $C_{\text{wc}}(L) \in \Omega(N(L))$? No; can only say $C_{\text{wc}}(L) \in \Omega(1)$.

Best/Worst Cases

- We can simplify 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 $C_{\text{wc}}(N(L)) \in O(N(L))$, it follows that $C_{\text{wc}}(N) \in O(N)$, where $C_{\text{wc}}(N)$ is "worst-case cost near over all lists of size N."

Best of the Worst

- We just saw that $C_{\text{wc}}(N) \in O(N)$.

- But in addition, it’s also clear that $C_{\text{wc}}(N) \in \Omega(N)$.

- So we can say, most concisely, $C_{\text{wc}}(N) \in \Theta(N)$.

- 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:

  ```python
  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 $C_{\text{wc}}(N)$, 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, $C(L) \in O(N(L)^2), C(L) \in \Omega(1), C(L) \in \Theta(N^2)$.

- And $C_{\text{wc}}(N) \in \Theta(N^2)$.

Example: A Tricky Nested Loop

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

  ```python
  def is_unduplicated(L, pred):
      """True if 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 \( \text{pred}(L[0]) \) and \( L[0] \) is \( L[1] \).
- So \( C(L) \in O(N|L|), C(L) \in \Omega(1) \).
- And \( C_{\text{mc}}(N) \in \Theta(N) \).

Some Useful Properties

- We've already seen that \( \Theta(K_0 N + K_1) = \Theta(N) \) \((K_k, K_i \text{ here and elsewhere are constants})\).
- \( \Theta(N^k + N^{k-1}) = \Theta(N^k) \). Why?
- \( \Theta(f(N) + |g(N)|) = \Theta(\max(|f(N)|, |g(N)|)). \) Why?
- \( \Theta(\log_2 N) = \Theta(\log_2 N) \). Why? (As a result, we usually use \( \log_2 N = \lg N \) for all logarithms.)
- Tricky: \( \text{isn't} \) \( \Theta(f(N) + g(N)) = \Theta(\max(f(N), g(N))) \)?
- \( \Theta(N^k) \subset \Theta(k^2) \), if \( k_2 > 1 \). Why?

More Programs

- How long does the \text{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 \text{gen_tree_find} program from HW#8? Consider all trees where the inner nodes all have at least \( K_i \geq 2 \) children and at most \( K_0 \) (both constants). What is the worst-case time to search as a function of \( N \)? \( \Theta(\lg N) \)

Fast Growth

- Consider \text{Hackenmax} from Test#2 (with some name changes):
  
  ```python
  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))
  ```

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 \( 1 \mu \text{sec} \)).
- 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.

\[
\begin{array}{l|cccc}
\text{Time (\mu sec) for} & \text{Max } N \text{ Possible in} & \text{1 second} & \text{1 hour} & \text{1 century} \\
\text{problem size } N & & 1000000 & 10^{10} & 10^{10} \\
\lg N & & 10^6 & 3.6 \cdot 10^9 & 2.7 \cdot 10^{12} \\
N & & 63000 & 1.3 \cdot 10^8 & 7.4 \cdot 10^{10} \\
\sqrt{N} & & 1000 & 60000 & 1.6 \cdot 10^8 \\
N^2 & & 100 & 1500 & 14000 \\
2^N & & 20 & 32 & 41 \\
\end{array}
\]