Lecture 2:
• Algorithm Complexity Analysis

Announcements
• Problem Set 1: due Tues, Sept 13
• Quiz 1: Tues, Sept 13

Algorithm Efficiency (review)
Algorithm efficiency analysis usually means predicting computation time
... versus memory (space) usage, network communication, etc.

Goal: predict, understand, compare an algorithm’s performance
• assuming a certain model of computation ... e.g., the RAM model
• under different cases ... e.g., worst, best, average case
• under different assumptions ... e.g., mostly sorted, sparse vs dense
The RAM model (in brief)

Assumptions:
- a generic processor with basic instructions (load, store, mov, add, sub, etc)
- each instruction takes same amount of time (one time unit)
- instructions executed one after the other (no concurrency)
- no cache or virtual memory (read/write from RAM directly)

In class, sometimes even higher-level (pseudocode "instructions")

There are other models of computation, e.g., pointer machines

Basic Ideas

Input size (usually denoted \( n \))
- Algorithms assumed to take longer the more input they have
- Note \( n \) is never a negative value!

“Worst Case” Analysis
- Cases that for all \( n \) take the most amount of time
- Can be many worst cases for an algorithm
- Big-O notation usually related to worst cases (depending on context)

“Best Case” Analysis
- Cases that for all \( n \) take the least amount of time
- Big-\( \Omega \) notation usually related to best cases (depending on context)
“Average Case” Analysis
• The average amount of time over all cases (for all $n$)
• Often based on “typical” (or “expected”) input frequency distributions

To predict performance cost: count algorithm’s primitive operations
• Where the computational model (e.g., RAM Model) comes into play
• Want to know how many primitive steps needed relative to input size
• Represented as a function $T(n)$ for input size $n$

Can test predictions using experimental evaluation (“performance testing”)
• run an implementation and time it (for different input sizes)
• results depend on specific implementation, language, machine, etc

Example: Finding $T(n)$

Q: What are the best and worst cases for linear search?

Algorithm: LinearSearch
Input: An integer array $A[]$ of size $n$ and an integer $v$.
Result: True or false depending on if $v$ is in $A$.
1 begin
2 for $i \leftarrow 0$ to $n - 1$ do
3 if $A[i] = v$ then
4 return true
5 return false


Q: What is $T(n)$? (with assignment, comparison, sub, add, array access)
• $T(n) \geq 5$
• $T(n) \leq 6n + 3$
Asymptotic Notation

Two algorithms with same “growth rate” treated as same “complexity”
• e.g., if \( T_1(n) = 3n + 2 \) and \( T_2(n) = 5n - 1 \), both are “linear”
• suppress constant factors ... system/language dependent
• suppress lower-order terms (e.g., \( n \) in \( n^2 + n \)) ... irrelevant for large input

Three main notations for comparing algorithm growth rates:
• “Big-O” (O-notation) ... “not slower than” (upper bound)
• “Big-\( \Omega \) (\( \Omega \)-notation) ... “not faster than” (lower bound)
• “Big-\( \Theta \) (\( \Theta \)-notation) ... “exactly” (tight bound)

Notations largely independent of different cases
• e.g., algorithm’s average case could have a linear-time Big-O growth rate
• worst case often implied with Big-O, best case with Big-\( \Omega \)

Asymptotic Notation: Big-O

Definition:
\[ T(n) \text{ is } O(f(n)) \text{ iff there exist positive constants } k \text{ and } n_0 \text{ such that } T(n) \leq k \cdot f(n) \text{ for all } n \geq n_0 \]

Comments:
• showing \( T(n) \) is \( O(f(n)) \) involves finding a suitable \( k \) and \( n_0 \)
• Big-O is not a tight upper bound by definition, which can be misleading ...
• e.g., if \( T(n) \) is \( O(n) \), \( T(n) \) is also \( O(n^2) \) ... since \( T(n) \leq k \cdot n \leq k \cdot n^2 \)

Can also describe Big-O using limits ...
\[ T(n) \text{ is } O(f(n)) \text{ iff } \lim_{n \to \infty} \frac{T(n)}{f(n)} < \infty \]
Asymptotic Notation: Big-$\Omega$

**Definition:**

\[ T(n) \text{ is } \Omega(f(n)) \text{ iff there exist positive constants } k \text{ and } n_0 \text{ such that } \]
\[ T(n) \geq k \cdot f(n) \text{ for all } n \geq n_0 \]

**Comments:**

- showing \( T(n) \) is \( \Omega(f(n)) \) involves finding a \( k \) and \( n_0 \)
- Big-$\Omega$ is not a tight lower bound by definition, which can be misleading ...
- e.g., \( T(n) \) being \( \Omega(n^2) \) implies \( T(n) \) is also \( \Omega(n) \)

*Can also describe Big-$\Omega$ using limits ...*

\[ T(n) \text{ is } \Omega(f(n)) \text{ iff } \lim_{n \to \infty} \frac{T(n)}{f(n)} > 0 \]

Asymptotic Notation: Big-$\Theta$

**Definition:**

\[ T(n) \text{ is } \Theta(f(n)) \text{ iff there exist positive constants } k_1, k_2, \text{ and } n_0 \text{ such that } \]
\[ k_1 \cdot f(n) \leq T(n) \leq k_2 \cdot f(n) \text{ for all } n \geq n_0 \]

**Comments:**

- showing \( T(n) \) is \( \Theta(f(n)) \) by finding suitable \( k_1, k_2, \text{ and } n_0 \) constants
- or by showing \( \Omega(f(n)) \) and \( O(f(n)) \)

*Can also describe Big-$\Theta$ using limits ...*

\[ T(n) \text{ is } \Theta(f(n)) \text{ iff } 0 < \lim_{n \to \infty} \frac{T(n)}{f(n)} < \infty \]
Check In

Q: If \( T(n) = 5n^2 + 3n \), what values of \( k \) and \( n_0 \) show that \( T(n) \) is \( O(n^2) \)?
- \( k = 5 \) and \( n_0 = 1 \)
- \( k = 6 \) and \( n_0 = 2 \)
- \( k = 6 \) and \( n_0 = 3 \)
- \( k = 2 \) and \( n_0 = -1 \)
- \( k = 8 \) and \( n_0 = 1 \)

Q: Does the above change if \( T(n) \leq 5n^2 + 3n \)?

Q: Which of the following are true?
- If \( T(n) \) is \( O(n) \) then \( T(n) \) is also \( O(n^2) \)
- If \( T(n) \) is \( O(n \log n) \) then \( T(n) \) is also \( O(\log n) \)
- Algorithm’s with worst case \( O(1) \) are faster than those with best case \( O(1) \)

Example 1: Checking for Duplicates

Assume counting assign/initialize, compare, increment, array access, logical and

\[
\text{Algorithm: CheckDups (inefficient)}
\]

\[
\text{Input: An integer array } A[] \text{ of size } n.
\]

1 begin
2 \hspace{1em} found ← false
3 \hspace{1em} for \( i \leftarrow 0 \) to \( n - 1 \) do
4 \hspace{2em} for \( j \leftarrow 0 \) to \( n - 1 \) do
5 \hspace{3em} if \( A[i] = A[j] \) and \( i \neq j \) then
6 \hspace{4em} found ← true
7 return found

Q: What are the best and worst cases?
- \textit{Best: } \( A \) has unique values. \hspace{1em} \textit{Worst: } \( A \) has all the same value (all dups).

Q: What is \( T(n) \)?
- \( 5n^2 + 6n + 3 \leq T(n) \leq 8n^2 + 3n + 3 \)
- \( n(n - 1) \) calls of “\( \text{found } \leftarrow \text{true} \)” in worst case

Q: What is the corresponding Big-\( \Omega \) and Big-\( O \)?
- \( \Omega(n^2) \) and \( O(n^2) \)
Example 2: Selection Sort

Assume only counting array comparisons (i.e., \(A[j] < A[k]\) as one operation)

Algorithm: SelectionSort
Input: An integer array \(A[]\) of size \(n\).
Result: \(A[]\) sorted in ascending order.

\[
\text{begin} \\
\text{for } i \leftarrow 0 \text{ to } n-2 \text{ do} \\
\quad k \leftarrow i \\
\quad \text{for } j \leftarrow i + 1 \text{ to } n-1 \text{ do} \\
\quad \quad \text{ /* note } j \text{ depends on } i */ \\
\quad \quad \text{ if } A[j] < A[k] \text{ then} \\
\quad \quad \quad k \leftarrow j \\
\quad \quad \text{ /* only counting this line */} \\
\quad \quad \text{ swap}(A[i], A[k]) \\
\text{end}
\]

Q: What is \(T(n)\)? ... note no best/worst case (for comparisons)

- inner loop executes \(n-1\) (\(i = 0\)), \(n-2\) (\(i = 1\)), \ldots, \(1\) (\(i = n-2\)) times
- giving: \(\sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n\) (where \(\sum_{x=1}^{n} x = \frac{n(n+1)}{2}\))

Q: What is the corresponding Big-\(\Theta\)? ... \(\Theta(n^2)\)

Properties of Big-O

Sum of two functions:

\[O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))\]

- e.g., duplicate check \(O(n^2)\) followed by member check \(O(n)\) is \(O(n^2)\)

Product of two functions:

\[O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))\]

- e.g., doing \(O(n)\) work \(O(\log n)\) times is \(O(n \log n)\)
- usually requires more careful analysis to ensure really a product