CS161
Design and Analysis of Algorithms

Dan Boneh

Spring 2001

Lecture 1, April 3, 2001

Administrative

Web page http://theory.stanford.edu/~dabo/cs161
  » Handouts
  » Announcements
  » Late breaking news

• Grading and course requirements
  » Midterm/final/hw
  » Project
  » Late HW policy
  » Importance of readable HW
  » Collaboration

• Probability - Ch. 6.2, pp 104–115 READ NOW !
Why Study Algorithms? (why cs161?)

- Bag of tricks
  - Sorting
  - Data structures: queues/stacks/heaps/trees
  - Search

- Methodology - how to design algorithms
  - Divide & conquer
  - Recursive algorithms
  - Randomized algorithms
  - Dynamic programming

- Useful abstractions.
  - Scheduling classes $\rightarrow$ graphs.
  - Job assignment $\rightarrow$ balls and boxes.

- Higher-level way of approaching problems

How to compare algorithms?

- Code and run - experiment
  - Inputs?
  - Parameters?
  - Bad implementations?

- Average case
  - what is “average input”??

- Worst case
  - Asymptotics
  - rough idea on performance
  - analytical dependence between parameters
Example from Ch. 2

- Insertion sort:
  
  ```
  for j = 2 to n
    key = A(j)
    i = j - 1
    while i > 0 and A(i) > key
      A(i+1) = A(i)
      A(i) = key
      i--
    end
  end
  ```

- Example:
  
  7 3 5 8 1 2
  3 7
  3 5 7
  3 5 7 8
  1 3 5 7 8
  1 2 3 5 7 8

About Pseudo-Code

- Not really a program, just an outline
- Enough details to establish the running time and correctness.
- No error-handling mechanisms.

- Even pseudo-code is too complicated!
  Note that for a trivial algorithm it obscures what is really going on...

- The "in-place" part is an optimization.
  We could start by a simpler description:
  
  » Go over the numbers one-by-one, starting from the first, copy to new array.
  » Each time copy to the correct place in the new array.
  » In order to create empty space, shift the numbers that are larger than the currently considered number one cell to the right.
Analysis

- Correctness and termination.

- Running time:
  - Depends on input size
  - input properties

- Want an upper bound on:
  - Worst case: max $T(n)$, any input.
  - Expected: $E[T(n)]$, input taken from a distribution. which ??
    - example: sorting arriving TCP/IP packets - they are mostly sorted already.
  - Best case: Can be used to argue that the algorithm is really bad.
    (any algorithm can be rewritten to have an excellent "best case" performance)

Back to insertion sort

- Insertion sort:
  ```
  for j = 2 to n
      key = A(j)
      i = j-1
      while i > 0 and A(i) > key
          A(i+1) = A(i)
          A(i) = key
          i--
      end
  end
  ```

- Simplified algorithm:
  - Go over the numbers one-by-one, starting from the first, copy to new array.
  - Each time copy to the correct place in the new array.
  - In order to create empty space, shift the numbers that are larger than the currently considered number one cell to the right.
  - n times
  - $\sum_{j=1}^{n} (t_j - 1)$
  - $t_j$ each
Analysis

- Best running time: Outer loop always executed, Inner loop - not executed if input already sorted.
- Assume each operation takes 1 time unit - approximation.

\[
\sum_{j=1}^{\frac{n}{2}} t_j = \frac{n(n+1)}{2} - 1
\]

- Would like to formalize this statement!
- Do we really need to pay close attention to all the indices in the summations? Maybe some or them are not really important??

Formalization

- How to formalize that \( \frac{n(n+1)}{2} \) was the main issue??
- The answer is asymptotic analysis:
  - Ignore machine-dependent constants.
  - Look at growth of \( T(n) \) as \( \infty \)
- Intuition: drop low-order terms
eg:

\[
5n^4 + 10n^2 - 3n + 2 = \Theta(n^4)
\]

Idea: as \( n \to \infty \), \( \Theta(n^2) \) becomes better (faster) than \( \Theta(n^4) \)
Back to insertion sort analysis

- Inner loop was \( \Theta(j) \)
  \[
  T(n) = \frac{1}{2} \sum_{j=1}^{n} \Theta(j) = \Theta\left(\frac{1}{2} n^2\right)
  \]

- Is this formal? **NO!**
  
  Example, using the same logic:
  
  \[ \Theta(1) + \Theta(1) = \Theta(1) \]
  
  seems to imply that \( \sum_{j=1}^{n} \Theta(1) = \Theta(1) \) ← Incorrect!

- We need formalization!
  
  Another example: \( \log n = \Theta(1/n^{1/10}) \)

---

Asymptotics

- **big-Oh notation:**
  
  \( f(n) = O(g(n)) \iff \exists \text{const } c, n_0 \text{ s.t. } \forall n \geq n_0 \colon 0 \leq f(n) \leq c g(n) \)

- Example: \( 2n^2 = O(n^6) \) but not vice versa!!

- “=” is not equality but membership in a set.
  
  Set notation is cumbersome:
  
  \[ O(g(n)) = \{ f(n) \mid \exists \text{const } c, n_0 \text{ s.t. } \forall n \geq n_0 \colon 0 \leq f(n) \leq c g(n) \} \]

- What do we mean by \( f(n) = O(n) + n^2 \)
  
  \[ \iff \exists h(n) = O(n), f(n) = h(n) + n^2 \]

- We are too lazy to specify \( h(n) \) exactly!
Asymptotics

- **Small-o notation:**
  \[ f(n) = o(g(n)) \iff \forall c, \exists n_0 \text{ s.t. } \forall n \geq n_0; \ 0 \leq f(n) < cg(n) \]

Prove that \( n = o(n^2) \):
Given \( c \), let \( n_0 = 2/c \)
\[ \Rightarrow \text{for } n \geq n_0, \ n^2 \geq \frac{2}{c} n \Rightarrow cn^2 \geq c\left(\frac{2}{c} n\right) = 2n > n \quad \text{QED} \]

\[ f = O(g) \text{ in both cases!} \]

Omega notation

- **Big-Omega:**
  \[ f(n) = \Omega(g(n)) \iff \exists c, n_0 \text{ s.t. } \forall n \geq n_0; \ 0 \leq cg(n) \leq f(n) \]

- **Small-omega:**
  \[ f(n) = \omega(g(n)) \iff \forall c, \exists n_0 \text{ s.t. } \forall n \geq n_0; \ 0 \leq cg(n) < f(n) \]

\[ O: \leq \quad \omega: < \]
\[ \Omega: \geq \quad \omega: > \]
Transitivity etc.

- **Most rules apply:**
  - Example: transitivity

\[ a \leq b, \quad b \leq c \Rightarrow a \leq c \]

\[ f = O(g), \quad g = O(h) \Rightarrow f = O(h) \]

Proof:
- \( f = O(g) \Rightarrow \exists c, n_1 \text{ s.t. } \forall n \geq n_1: \quad 0 \leq f(n) \leq c \cdot g(n) \)
- \( g = O(h) \Rightarrow \exists c, n_2 \text{ s.t. } \forall n \geq n_2: \quad 0 \leq g(n) \leq c \cdot h(n) \)

Take \( n = \max(n_1, n_2) \), \( c = c \cdot c \cdot \cdot c \cdot c \cdot c \cdot c \). Then:
- \( \forall n \geq n_3: \quad 0 \leq f(n) \leq c \cdot g(n) \leq c \cdot c \cdot h(n) = c \cdot h(n) \)
- \( f(n) = O(g(n)) \quad \text{QED} \)

- **Not all rules apply!**

\[ \exists f, g \text{ s.t. } f \neq O(g) \text{ and } g \neq O(f) \]

Example: \( f = n \cdot g(n) = n^{1+\sin n} \)

Theta notation

- **Theta:**

\[ f(n) = \Theta(g(n)) \Leftrightarrow \exists c, c_0, n_0 \text{ s.t. } \forall n \geq n_0: \quad 0 \leq c \cdot g(n) \leq f(n) \leq c_0 \cdot g(n) \]

- Often confused with Big-Oh notation!

- **Example:**

\[ n^2 / 2 - 2n = \Theta(n^2) \]

Proof:
- take \( n_0 = 8 \), then for \( n \geq n_0 \):
\[ n^2 / 2 - 2n \geq n^2 / 4 + 8n / 4 - 2n = n^2 / 4 \]

On the other hand, we have:
- \( n^2 / 2 - 2n < n^2 / 2 \)

Thus:
- \( n^2 / 4 \leq n^2 / 2 - 2n \leq n^2 / 2 \)

\[ i.e. c_1 = 1/4, c_2 = 1/2. \]

- **Claim:** Low order terms do not matter. Needs a proof! (HW?)
Simple Theorem

- **Claim** $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n)) \Rightarrow f(n) = \Theta(g(n))$

Proof:
\[\exists n_1, c_1 \text{ s.t. } \forall n \geq n_1: 0 \leq f(n) \leq c_1 g(n)\]
\[\exists n_2, c_2 \text{ s.t. } \forall n \geq n_2: 0 \leq g(n) \leq c_2 f(n)\]

\[\Rightarrow \forall n \geq \max(n_1, n_2): 0 \leq \frac{1}{c_2} g(n) \leq f(n) \leq c_2 g(n) \quad \text{QED}\]

Summary

- **Remember the definitions.**
- **Formally prove from definitions.**
- **Use intuition from the properties of \(=\), \(\geq\), etc.**