CS161
Design and Analysis of Algorithms

Dan Boneh

Spring 2001

1

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 !**

2

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 → graphs.
  - Job assignment → balls and boxes.
- Higher-level way of approaching problems

3

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

4

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 algorithms 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)

7

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
```

n

n-1

...

$\sum_{j=2}^{n}(t_j - 1)$

- Simplified algorithm:
  - » Go over the numbers one-by-one, starting from the first, copy to new array. } n times
  - » 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.

  } t_j each

8

Analysis

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

$$n+(n-1)+(n-1)+\sum_{j=2}^{n} t_j + 2\sum_{j=2}^{n}(t_j-1)+(n-1)$$
$$t_j \text{ worst case } \approx j$$
$$\Rightarrow \sum_{j=2}^{n} t_j = \underbrace{\frac{n(n+1)}{2}}_{\text{This dominates !}} -1$$

- **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 ??**

9

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** ∞
- **Intuition: drop low-order terms**
  eg:

$$5n^4+10n^2\text{-}3n+2=\Theta(n^4)$$
Idea: as $n\rightarrow\infty$, $\Theta(n^2)$ becomes better (faster) than $\Theta(n^4)$

10

Back to insertion sort analysis

- **Inner loop was** Q**(j)**

$$T(n) \approx \sum_{2}^{n} \Theta(t_j) = \Theta(\sum_{2}^{n} t_j) = \Theta(n^2)$$

- **Is this formal ?** **NO !**
  Example, using the same logic:

$\Theta(1) + \Theta(1) = \Theta(1)$

seems to imply that $\sum_{i=1}^{n} \Theta(1) = \Theta(1) \quad \leftarrow$ Incorrect !

- **We need formalization !**

Another example: $\log n \overset{??}{\sim} n^{1/10}$

11

Asymptotics

- **big-Oh notation:**

$f(n) = O(g(n)) \Leftrightarrow \exists \text{const } c, n_0 \text{ s.t. } \forall n \geq n_0\text{:} \;\; 0 \leq f(n) \leq cg(n)$

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

- **"=" is not equality but membership in a set.**
  Set notation is cumbersome:

$O(g(n)) = \{ f(n) | \exists \text{const } c, n_0 \text{ s.t. } \forall n \geq n_0\text{:} \;\; 0 \leq f(n) \leq cg(n) \}$

- **What do we mean by** $f(n) = O(n) + n^2$
  $\Leftrightarrow \exists h(n) = O(n), f(n) = h(n) + n^2$

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

12

Asymptotics

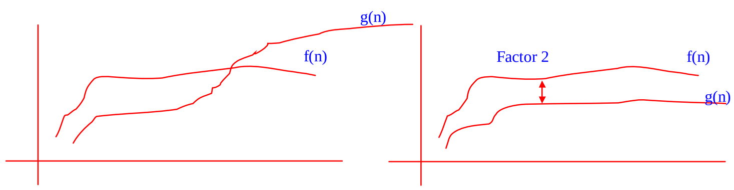
- **Small-oh notation:**

$$f(n) = o(g(n)) \Leftrightarrow \forall \text{const } c, \exists n_0 \text{ s.t. } \forall n \geq n_0 \text{: } 0 \leq f(n) < cg(n)$$

Differences from big-Oh

Prove that $n = o(n^2)$:
Given c, lets take $n_0 = 2/c$

$\Rightarrow$ for $n \geq n_0$, $n^2 \geq \frac{2}{c} n \;\Rightarrow\; cn^2 \geq c(\frac{2}{c} n) = 2n > n$ QED



f=O(g) in both cases !

Omega notation

- **Big-Omega:**

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists \text{const } c, n_0 \text{ s.t. } \forall n \geq n_0 \text{: } 0 \leq cg(n) \leq f(n)$$

- **Small-omega:**

$$f(n) = w(g(n)) \Leftrightarrow \forall \text{const } c, \exists n_0 \text{ s.t. } \forall n \geq n_0 \text{: } 0 \leq cg(n) < f(n)$$

O: $\leq$ $\quad$ o: $<$
Ω: $\geq$ $\quad$ ω: $>$

Transitivity etc.

- **Most rules apply:** $a \le b,\ b \le c \Rightarrow a \le c$
  Example: transitivity $f = O(g), g = O(h) \Rightarrow f = O(h)$

  Proof:
  $f = O(g) \Rightarrow \exists \text{const } c_1, n_1 \text{ s.t. } \forall n \ge n_1:\ 0 \le f(n) \le c_1 g(n)$
  $g = O(h) \Rightarrow \exists \text{const } c_2, n_2 \text{ s.t. } \forall n \ge n_2:\ 0 \le g(n) \le c_2 h(n)$

  Take $n_3 = \max(n_1, n_2), c_3 = c_1 c_2$
  Then: $\forall n \ge n3:\ 0 \le f(n) \le c_1 g(n) \le c_1 c_2 h(n) = c_3 h(n)$
  $\Rightarrow f(n) = O(g(n))$ QED

- **Not all rules apply !**

  $\exists f, g \text{ s.t. } f \ne O(g) \text{ and } g \ne O(f)$
  example: $f = n,\ g = n^{1+\sin n}$

15

Theta notation

- **Theta:**
  $f(n) = \Theta(g(n)) \Leftrightarrow \exists \text{const } c_1, c_2, n_0 \text{ s.t. } \forall n \ge n_0:\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n)$
- **Often confused with Big-Oh notation !**
- **Example:** $n^2/2 - 2n = \Theta(n^2)$

  Proof:
  take $n_0 = 8$, then for $n \ge n_0$:
  $n^2/2 - 2n \ge n^2/4 + 8n/4 - 2n = n^2/4$

  On the other hand, we have: $n^2/2 - 2n < n^2/2$

  Thus: $n^2/4 \le n^2/2 - 2n \le n^2/2$ i.e. $c_1 = 1/4, c_2 = 1/2$.

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

16

Simple Theorem

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

  Proof:
  $\exists n_1, c_1$ s.t. $\forall n \geq n_1$: $0 \leq f(n) \leq c_1 g(n)$
  $\exists n_2, c_2$ s.t. $\forall n \geq n_2$: $0 \leq g(n) \leq c_2 f(n)$

  $\Rightarrow \forall n \geq \max(n_1, n_2)$: $0 \leq \frac{1}{c_2} g(n) \leq f(n) \leq c_1 g(n)$ QED

17

Summary

- **Remember the definitions.**
- **Formally prove from definitions.**
- **Use intuition from the properties of “£”, “3“, etc.**

18