

CS161

Design and Analysis of Algorithms

Serge Plotkin

Fall 2009

1

Administrative

Lecture 1, Sept. 22, 2009

- TAs:
 - » Shipra Agrawal [shipra@stanford.edu]
 - » Rohan Jain [rohanj@stanford.edu]
 - » Michael Muni [muni@stanford.edu]
- Office hours/offices – see web site
- Review session:
Friday 3:15-4:05, Gates B03 [no session first week]
- Web page <http://cs161.stanford.edu>
 - » Handouts
 - » Announcements
 - » Late breaking news
- Grading and course requirements
 - » Midterm/final/hw 30/40/30%
 - » Late HW policy – 2 "late periods"
 - » Importance of **readable HW**
 - » Collaboration
- Probability – Appendix C **READ NOW!**
(useful to go through A, B and D as well)

2

Textbook

- Cormen/Leiserson/Rivest/Stein, 3rd edition.
- <http://library.stanford.edu/books24>
(Only 2nd edition right now, requires Stanford login)
- No CD required
- Textbook by Kleinberg/Tardos highly recommended

3

Why Study Algorithms ? (why cs161?)

- Bag of tricks
 - » Sorting
 - » Data structures: queues/stacks/heaps/trees
 - » Search
- Methodology - how to design algorithms
 - » Divide & conquer
 - » Greedy
 - » Dynamic programming
 - » Randomized algorithms
- Useful abstractions.
 - » Graphs
 - Scheduling classes
 - Finding optimal routes in a network
 - Maps
 - » Balls and boxes
 - Job assignment
 - Managing packet queues
- Higher-level way of approaching problems
- Communicate with peers

4

What will we study ?

- Math background
 - » Asymptotic analysis
 - » A bit of probability
 - » Recurrences
- Data Structures
 - » Hash functions
 - » Heaps
 - » Binary search trees
- Techniques
 - » Divide & Conquer
 - » Greedy
 - » Dynamic Programming
- Specific algorithms
 - » Min-cost Spanning Tree
 - » Shortest paths
 - » Compression - Huffman encoding
 - » Longest common subsequence
 - » Stable marriage
 - » Closest pair of point
 - » more

5

Preliminary plan – Will change

- Algorithmic complexity and analysis (4 lectures)
- Randomization, divide and conquer (2 lectures)
- Heaps and counting sort (1 lecture)
- Hashing (2 lectures)
- Tree and graph definitions and properties (1 lecture)
- Binary Search Trees (1 lecture)
- Greedy Algorithms (2 lectures)
- Dynamic programming (3 lectures)
- Graph algorithms (4 lectures)

6

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

7

Example: Insertion Sort

- Pseudocode:
 - » Not really a program, just an **outline**
 - » Enough details to establish the **running time** and **correctness**.
 - » No error-handling mechanisms.
- Even pseudo-code can be **too complicated !**
Even for a trivial algorithm can hide/obscure what is really going on...
- Description of insertion sort (before trying to write pseudocode):
 - » 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.
 - » Can be done "in-place" (optimization).

8

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)

9

Back to insertion sort

- **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

} t_j each (at iteration j)

- **pseudocode:**

```

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
    
```

$\xrightarrow{\hspace{10em}}$ n
 n-1
 ...
 $\xrightarrow{\hspace{10em}}$ $\sum_{j=2}^n t_j$

10

Analysis

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

$$\sum_{j=2}^n t_j$$

$t_{j=j-1}$ in the worst case

⇒ running time proportional to $\frac{n(n-1)}{2}$
n² term dominates !

- Would like to formalize this statement !

11

Formalization

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

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

Idea: as $n \rightarrow \infty$, $\Theta(n^2)$ becomes better (faster) than $\Theta(n^4)$

12

Back to insertion sort analysis

- Inner loop (shift numbers to make space) was $\Theta(j)$

$$T(n) \approx \sum_{j=2}^n \Theta(j) = \Theta\left(\sum_{j=2}^n j\right) = \Theta(n^2)$$

- Is this formal ?
- Example, using the same logic:

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

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

- We need formalization !

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

13

Asymptotics

- **big-Oh notation:**

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

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

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

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

- 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!

14