Lecture 1: Introduction
(CLAR 1+2.1-2.2)

January 10, 2002

1 Administration

1.1 Basic information

- CPS130: Introduction to the Design and Analysis of Algorithms
  - Course www page at http://www.cs.duke.edu/education/courses/cps130/
- Instructor: Lars Arge (D205 LSRC Bldg., large@cs.duke.edu)
  - Algorithms researcher with special emphasis on problems involving massive data sets.
- TAs:
  - Abhilinet Guria (N-03 North Bldg., guria@cs.duke.edu)
  - David Irwin (N-01 North Bldg., irwin@cs.duke.edu)
- UTAs and office hours to be determined shortly.

1.2 Course material

  - Note: Some chapters of the recent second edition of the book is substantially different from the first edition. I will often present material in a (slightly) different way than the book.
- Lecture notes
  - Draft handed out at lecture - final version on www after lecture.
- Handouts.

1.3 Course synopsis

- The course builds on the study of algorithms and data structures from CPS100.
  - Some material (especially in the beginning) will be repetition of CPS100 material (but covered in more depth).
- Prerequisites
  - Good math skills: Basic notation, proofs (induction), limits, basic probability theory,... (we will review all of these).
– Knowledge of programming: How to express algorithms in a computer language, recursion,... (We will not be doing implementations!).

• Topics covered:
  – Mathematical foundation (Growth of functions, summations, recurrences)
  – Sorting and selection
  – Searching
  – Amortized analysis
  – Algorithm design techniques
  – Graph algorithms
  – Complexity

• Lecture Schedule:
  – Lecture www page (http://www.cs.duke.edu/education/courses/cps130/spring02/lectures.html) contains information about covered material.

1.4 Grading
• Homework assignments – approximately 40%.
• Midterm (March 21) and a final exam (April 29) – approximately 60%.
• Class participation.

1.5 Homework
• Homework will be assigned (on the web) every second Tuesday and is always due at the start of the Tuesday class two weeks later.
• Collaboration is strongly encouraged but solutions must be written up individually.

1.6 Recitation Sessions
• Recitation sessions are held Thursday and Friday.
• Homework assignment will contain practice problems to be solved at recitation sessions.
• Recitation sessions will also be used to go through homework solutions as well as discuss questions about material covered in class.
2 Introduction

- Class is about designing and analyzing algorithms
  - Algorithm: A well-defined procedure that transfers an input to an output.
    * Not a program (but often specified like it): An algorithm can often be implemented in several ways.
  - Design: We will study methods/ideas/tricks for developing (fast!) algorithms.
  - Analysis: Abstract/mathematical comparison of algorithms (without actually implementing them).

- Math is needed in three ways:
  - Formal specification of problem
  - Analysis of correctness
  - Analysis of efficiency (time, memory use,...)

- Hopefully the class will show that algorithms matter!

3 Algorithm example: Insertion-sort

3.1 Specification

- Input: $n$ integers in array $A[1..n]$
- Output: $A$ sorted in increasing order

3.2 Insertion-sort algorithm

```plaintext
FOR j = 2 to n DO
  key = A[j]
  i = j - 1
  WHILE i > 0 and A[i] > key DO
    A[i + 1] = A[i]
    i = i - 1
  OD
  A[i + 1] = key
OD
```

- NOTE: Algorithm shows example of the (Pascal like) pseudo-code we will sometimes use to describe algorithms.

Example:
5 2 4 6 1 3  j=2  i=1  key=2
5 5 4 6 1 3  i=0
2 5  4 6 1 3

2 5 4 6 1 3  j=3  i=2  key=4
2 5 5 6 1 3  i=1
2 4 5  6 1 3

2 4 5 6 1 3  j=4  i=3  key=6
2 4 5 6  1 3

2 4 5 6 1 3  j=5  i=4  key=1
2 4 5 6 6 3  i=3
2 4 5 5 6 3  i=2
2 4 4 5 6 3  i=1
2 2 4 5 6 3  i=0
1  2 4 5 6 3

1 2 4 5 6 3  j=6  i=5  key=3
1 2 4 5 6 6  i=4
1 2 4 5 5 6  i=3
1 2 4 4 5 6  i=2
1 2 3 4 5 6  |

3.3 Correctness

- Induction (loop invariant):
  - The Invariant “A[1..j-1] is sorted” holds at the beginning of each iteration of FOR-loop.
  - When j=n+1 (Termination) we have correct output.

3.4 Analysis

- We want to predict the resource use of the algorithm.
- We can be interested in different resources
  - but normally running time.
- To analyze running time we need mathematical model of a computer:
Random-access machine (RAM) model:
- Memory consists of infinite array
- Instructions executed sequentially one at a time
- All instructions take unit time:
  * Load/Store
  * Arithmetics (e.g. +, -, *, /)
  * Logic (e.g. >)

- Running time of an algorithm is the number of RAM instructions it executes.
- RAM model not completely realistic, e.g.
  - memory not infinite (even though we often imagine it is when we program)
  - not all memory accesses take same time (cache, main memory, disk)
  - not all arithmetic operations take same time (e.g. multiplications expensive)
  - instruction pipelining
  - other processes

- But RAM model often enough to give relatively realistic results (if we don’t cheat too much).

- Running time of insertion-sort depends on many things
  - How sorted the input is
  - How big the input it
  - ...

- Normally we are interested in running time as a function of input size
  - in insertion-sort: $n$.

- We don’t really want to count every RAM instruction
  - Let us analyze insertion-sort by assuming that line $i$ in the program use $c_i$ RAM instructions.
  - How many times are each line of the program executed?
    * Let $t_j$ be the number of times line 4 (the WHILE statement) is executed in the $j$’th iteration.

```
FOR $j = 2$ to $n$ DO
  $key = A[j]$
  $i = j - 1$
  WHILE $i > 0$ and $A[i] > key$ DO
    $A[i + 1] = A[i]$
    $i = i - 1$
  OD
  $A[i + 1] = key$
OD
```

<table>
<thead>
<tr>
<th></th>
<th>cost</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$n$</td>
<td></td>
</tr>
<tr>
<td>$c_2$</td>
<td>$n - 1$</td>
<td></td>
</tr>
<tr>
<td>$c_3$</td>
<td>$n - 1$</td>
<td></td>
</tr>
<tr>
<td>$c_4$</td>
<td>$\sum_{j=2}^{n} t_j$</td>
<td></td>
</tr>
<tr>
<td>$c_5$</td>
<td>$\sum_{j=2}^{n} (t_j - 1)$</td>
<td></td>
</tr>
<tr>
<td>$c_6$</td>
<td>$\sum_{j=2}^{n} (t_j - 1)$</td>
<td></td>
</tr>
<tr>
<td>$c_7$</td>
<td>$n - 1$</td>
<td></td>
</tr>
</tbody>
</table>
• Running time: (depends on $t_j$)

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j + c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1)$$

- Best case: $t_j = 1$ (already sorted)

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

$$= k_1 n - k_2$$

**Linear function of $n$**

- Worst case: $t_j = j$ (sorted in decreasing order)

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j + c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \left(\frac{n(n+1)}{2} - 1\right) + c_5 \left(\frac{(n-1)n}{2}\right) + c_6 \left(\frac{(n-1)n}{2}\right) + c_7(n-1)$$

$$= (c_4/2 + c_5/2 + c_6/2)n^2 + (c_1 + c_2 + c_3 + c_4/2 - c_5/2 - c_6/2 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

$$= k_3 n^2 + k_4 n - k_5$$

**Quadratic function of $n$**

Note: We used $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$ (Next week!)

- “Average case”: Be careful! (average over what?)

We assume $n$ numbers chosen randomly $\Rightarrow t_j = j/2$

$$T(n) = k_6 n^2 + k_7 n + k_8$$

Still **Quadratic function of $n$**

• Note:

- We will normally be interested in worst-case running time.
  * Upper bound on running time for any input.
  * For some algorithms, worst-case occur fairly often.
  * Average case often as bad as worst case (but not always!).

- We will only consider order of growth of running time:
  * We already ignored cost of each statement and used the constants $c_i$.
  * We even ignored $c_i$ and used $k_i$.
  * We just said that best case was **linear in $n$** and worst/average case **quadratic in $n$**.

$\Rightarrow$ **$O$-notation** (best case $O(n)$, worst/average case $O(n^2)$) (next lecture!)

6