Institut for Matematik og Datalogi
Syddansk Universitet October 1, 2003
JFB

Introduction to Computer Science
E03 – Lecture 6

Lecture, September 29

Peter Kornerup lectured on networks from sections 3.5 and 3.6. We also began on chapter 4, Algorithms.

Lecture, October 6

We will finish chapter 4 (starting with section 4.4, which we began on last week) on algorithms.

Lecture, October 20

We will finish chapter 4 on algorithms. Since concepts from chapters 5, 6 and 7 should be covered in some of your other first year courses, we will be skipping them. We will begin on chapter 8.

Discussion section: week 41, in the Terminal Room

Discussion in groups (only two, or possibly three, people per group, since you will sit at a computer):

The goal of this lab is to help you to gain some understanding of the fact that most problems have more than one algorithmic solution and that these solutions can differ greatly as to how practical they are. You will experiment with three different sorting algorithms and compare them. Review sections 4.4, 4.5, and 4.6 in the textbook before coming to the lab.

To start up the program you should find the home page for this course

http://www.imada.sdu.dk/Courses/DM35/

Near the bottom that page, there is a link to a sorting simulator. Click on that link. Note that the program sorts bars of different lengths, rather than numbers. It is easy to think of the bars as numbers, and it is easiest to see what is happening with the bars. To the right, you will see program code for a sorting algorithm in the programming language Java.

1. Under **Algorithm**, choose **Insertion Sort**.

   Set the **# of Blocks** to 8. Click on **Sort** and watch the algorithm execute (both the code on the right and the sorting of the bars on the left). *What does a red bar mean? A blue bar? A green bar? How many comparisons are done and how many swaps?*

   After it has stopped, try starting it again on the sorted bars. *Now how many comparisons are done and how many swaps?*

   Note that if you want to start with fresh data, you can select the **Arrangement** to be **Random** at any time. You may have to click on **Stop** before clicking on **Sort** again.

2. Under **Algorithm**, choose **Selection Sort**. (See problem 6 on page 181 of your textbook.) Try running Selection Sort on random data. *What does the algorithms do? How does it work?* Write down the current values for **# of Blocks**, **# of Swaps**, and **# of Comparisons**. If the number of bars is n, the number of comparisons should be $\sum_{i=2}^{n}(i-1) = \frac{1}{2}n^2 - \frac{1}{2}n$. *Why is this the number of comparisons? How many comparisons should there theoretically be in this case, where $n = 8$? How does this compare with practice? Explain why there were 7 swaps.*

   Increase the **Speed** by clicking on the arrows or moving the bar between them. Try running Selection Sort with 25 bars, 50 bars, and 100 bars. *Write down in each case the* **# of Bars**, **# of Swaps**, *and* **# of Comparisons**. *How do these compare with the predicted values? What values would you expect if the number of bars was 10,000?*

3. Change the **Algorithm** to **Quick Sort**. Try running Quick Sort on random data with 8 bars and speed 1. *What does the algorithm do? How does it work?* It may actually be easier to see what is happening with 100 bars and speed 100. If the bar chosen to partition on ends up in the middle of the current range every time, then you are left with

two problems (the left and right) half as big as the original, and it is not too hard to show that the number of comparisons is $\Theta(n \log n)$. This does not happen every time, and a complete average case analysis is beyond the scope of this course. It is not so hard to show that if the pivot element is always within the middle 15/16 of the elements, then you get $\Theta(n \log n)$ time anyway. Intuitively, it seems likely that this will happen most of the time, so that is why one generally gets this behavior.

Increase the **Speed**. Try running it three times with 25 random bars, writing down the **# of Blocks**, **# of Swaps**, and **# of Comparisons**. *Why didn't you get the same answer every time?*

Try repeating this with 50 bars and 100 bars. *Write down in each case the* **# of Bars***,* **# of Swaps***, and* **# of Comparisons***. What do you conclude about Quick Sort's running time? Is $\Theta(n \log n)$ believable?* (Note that Θ notation is discussed on pages 192–193 of the textbook. Informally, it means in this case that for large n, the running time is close to some constant times $n \log n$.) *Try to make an estimate as to how long Quick Sort would take with 10,000 bars.*

4. Change the **Arrangement** of the bars to **Ascending**, so your initial data starts out sorted, instead of random. Try running Quick Sort with 25 bars. *How does this compare with Selection Sort? With Insertion Sort? Explain your results.*

5. (The next two problems are independent of the previous ones.) Do problem 6 on pages 197–198 of the textbook. What precondition and loop invariant should hold? Note that the comma in the the test for the *while* means less than.

6. Consider the following problem: There are three politicians, A, B, and C. You know that one of them always tells the truth, one of them always lies, and one of them sometimes tells the truth and sometimes lies. You are allowed to ask these three politicians any three true/false questions you like, and you may choose which politician is asked which question. How would you determine how to order the politicians by how often they tell the truth? This problem is quite difficult. Try your problem solving abilities, but do not be disappointed if you fail.

Assignment due 8:15, October 21

Late assignments will not be accepted. Working together is not allowed. (You may write this either in English or Danish, but write clearly if you do it by hand.)

1. How many comparisons would Selection Sort do if there were 150 numbers to be sorted? How many swaps?

2. Do problem 2 on page 189.

3. Do problem 5 on page 197. What precondition and loop invariant should hold? Note that the comma in the the test for the *while* means less than.