CS1110 2 April 2009

Sorting: insertion sort, selection sort, quick sort

Do exercises on pp. 311-312 to get familiar with concepts and develop skill. Practice in DrJava! Test your methods!

Time spent on A5:

min	2 (11 students)
average	5.7
median	5
max	15

15 students took 9-14 hours
26 students took 7-8 hours

Course website contains more prelims and answers, for Prelim 1, 2, and 3

Comments on A5

Recursion:

Make requirements/descriptions less ambiguous, clearer; give more direction.

Should be a part where you can create your own recursive functions

Need optional problem with more complicated recursive solution would have been an interesting challenge, more recursive functions.

Do more of recursive graphics-type problems. They really make us think!

Make task 5 easier. I could not finish it.

Include a clue that more than one recursive call can be included in a recursive method. I wasted hours because I didn't know this.

Clarify clear(), setPanelSize(). Improve graphic rendering. Explain diff between JFrame and JPanel

Liked not having to write test cases!

Needed too much help, took too long

Add more methods; it did not take long

Give us the option to do the recursive methods with loops rather than recursively.

Sorting: "sorted" means in ascending order

pre: b [0 … n: ?] post: b [0 … n: sorted]

insertion sort inv: b [0 … i: sorted | i … n: ?]

```
for (int i= 0;  i < n;  i= i+1) {
    Push b[i] down into its sorted
        position in b[0..i];
}
```

0 … i	i
2 4 4 6 6 7	5

0 … i	i
2 4 4 5 6 6	7

Iteration i makes up to i swaps.
In worst case, number of swaps needed is
0 + 1 + 2 + 3 + … (n-1) = (n-1)*n / 2.

Called an "n-squared", or n^2, algorithm.

b[0..i-1]: i elements

in worst case:

Iteration 0: 0 swaps
Iteration 1: 1 swap
Iteration 2: 2 swaps
…

pre: b [0 … n: ?] post: b [0 … n: sorted]

insertion sort invariant: b [0 … i: sorted | i … n: ?]

Add property to invariant: first segment contains smaller values.

selection sort invariant: b [0 … i: ≤ b[i..], sorted | i … n: ≥ b[0..i-1], ?]

```
for (int i= 0;  i < n;  i= i+1) {
    int j= index of min of b[i..n-1];
    Swap b[j] and b[i];
}
```

i … n
2 4 4 6 6 | 8 9 9 **7** 8 9

i … n
2 4 4 6 6 | **7** 9 9 **8** 8 9

Also an "n-squared", or n^2, algorithm.

Quicksort

```
/** Sort b[h..k] */
public static void qsort(int[] b, int h, int k) {
    if (b[h..k] has fewer than 2 elements)
        return;
    int j= partition(b, h, k);
    // b[h..j–1] <= b[j] < b[j+1..k]
    // Sort b[h..j–1]  and  b[j+1..k]
    qsort(b, h, j–1);
    qsort(b, j+1, k);
}
```

To sort array of size n. e.g. 2^{15}

Worst case: n^2 e.g. 2^{30}

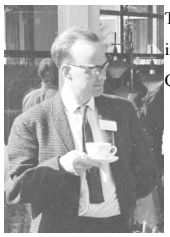
Average case: n log n. e.g. $15 * 2^{15}$

$2^{15} = 32768$

pre: b [h: x | h … k: ?]

j= partition(b, h, k);

post: b [h … j: <= x | j: x | j … k: >= x]



Tony Hoare, in 1968
Quicksort author

Tony Hoare in 2007 in Germany

Thought of Quicksort in ~1958. Tried to explain it to a colleague, but couldn't.
Few months later: he saw a draft of the definition of the language Algol 58 –later turned into Algol 60. It had recursion.
He went and explained Quicksort to his colleague, using recursion, who now understood it.

Viewpoint On teaching programming Reply

I don't like how we are forced to visualize things in Dr. Gries' way. … Entire point of programming is to be able to look at things in different ways and come up with different solutions for one problem. Forcing us to think of things in his way and testing us on it has been detrimental to my learning because in my opinion it wastes time and confuses me. This course should focus more on solving problems rather than drawing folders to represent objects.

1. A model of execution of Java programs is needed in order to bring understanding.
2. Problem solving *is* the focus. The programs you wrote for A5, the algorithms we are now studying, and the way we develop them, could not have been possible without the basics that we have given you.
3. We are giving you tools for coming up with *good* solutions, not just different ones.

7

The NATO Software Engineering Conferences
homepages.cs.ncl.ac.uk/brian.randell/NATO/

7-11 Oct 1968, Garmisch, Germany
27-31 Oct 1969, Rome, Italy

Download Proceedings, which have transcripts of discussions.
See photographs.

Software crisis:
Academic and industrial people.
Admitted for first time that they did not know how to develop software efficiently and effectively.

Software Engineering, 1968

Next 10-15 years: intense period of research of software engineering, language design, proving programs correct, etc.

9

Software Engineering, 1968

10

During 1970s, 1980s, intense research on
How to prove programs correct,
How to make it practical,
Methodology for developing algorithms

The way we understand recursive methods is based on that methodology.
Our understanding of and development of loops is based on that methodology.

Throughout, we try to give you thought habits to help you solve programming problems for effectively

Mark Twain: Nothing needs changing so much as the habits of **others**.

11

The way we understand recursive methods is based on that methodology.
Our understanding of and development of loops is based on that methodology.

Throughout, we try to give you thought habits to help you solve programming problems for effectively

Simplicity is key:
Learn not only to simplify, learn not to complify.

Separate concerns, and focus on one at a time.

Develop and test incrementally.

Don't solve a problem until you know what the problem is (give precise and thorough specs).

Learn to read a program at different levels of abstraction.

12