CS1110 9 November 2010
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!

<table>
<thead>
<tr>
<th>A5 times</th>
<th>hours</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>2</td>
<td>2-3</td>
</tr>
<tr>
<td>median</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>mean</td>
<td>6.5</td>
<td>5</td>
</tr>
<tr>
<td>max</td>
<td>19</td>
<td>6</td>
</tr>
</tbody>
</table>

\[0 + 1 + 2 + 3 + \ldots (n-1) = \frac{n(n-1)}{2}\]

In worst case, number of swaps needed is \(n^2\).

Iteration \(i\) makes up to \(i\) swaps.

In worst case, number of swaps needed is \(0 + 1 + 2 + 3 + \ldots (n-1) = \frac{(n-1)n}{2}\).

Called an “\(n\)-squared”, or \(n^2\), algorithm.

Sorting: “sorted” means in ascending order.

\[
\begin{array}{c|c|c}
\text{pre: } b & x & \text{post: } b \text{ sorted} \\
0 & ? & n \\
\end{array}
\]

insertion sort

\[
\begin{array}{c|c|c}
\text{inv: } b & x & \text{sorted} \\
0 & ? & n \\
\end{array}
\]

for \((\text{int} i=0; \ i<n; \ i=i+1)\) {

Push \(b[i]\) down into its sorted position in \(b[0..i]\);

\[
\begin{array}{c|c|c}
\text{1st iteration: } 0 & ? & i \\
2 & 4 & \text{6} \\
\end{array}
\]

Iteration \(i\) makes up to \(i\) swaps.

In worst case, number of swaps needed is \(0 + 1 + 2 + 3 + \ldots (n-1) = \frac{(n-1)n}{2}\).

Called an “\(n\)-squared”, or \(n^2\), algorithm.

Partition algorithm: Given an array \(b[h..k]\) with some value \(x\) in \(b[h]\)

\[
\begin{array}{c|c|c|c}
P: \ b & x & ? \\
\hline
\text{Q: } b & \lll x & \rrr x & k \\
\hline
\text{change: } b & \lll 3 \ 5 \ 4 \ 1 \ 6 \ 2 \ 3 \ 8 \ 1 & k \\
\text{into: } b & \lll 2 \ 1 \ 3 \ 5 \ 4 \ 6 \ 3 \ 8 & k \\
or b & \lll b \ 1 \ 3 \ 2 \ 3 \ 4 \ 5 \ 8 & k \\
\end{array}
\]

\(x\) is called the pivot value.

\(x\) is not a program variable; \(x\) just denotes the value initially in \(b[h]\).

Comments on A5

Liked not having to write test cases!

Recursion:
Make requirements/descriptions less ambiguous, cleaner; give more direction.

Need optional problem with more complicated recursive solution would have been an interesting challenge, more recursive functions. They make us think!

Make task 5 easier. I could not finish it.

Also an “\(n\)-squared”, or \(n^2\), algorithm.

Quick sort

\[
\begin{array}{c|c|c}
\text{pre: } b & \lll ? & k \\
\hline
\text{Post: } b & \lll x & \rrr x & k \\
\end{array}
\]

/** Sort \(b[h..k]\) */
public static void qsort(int[] b, int h, int k) {
    if (b[h..k] has fewer than 2 elements) return;
    \[
    \begin{array}{c|c|c|c}
    \text{int } j= \text{partition}(b, h, k); \\
    \text{if } b[h..j-1] \lll b[j] \rrr b[j+1..k] \\
    \text{qsort}(b, h, j-1); \\
    \text{qsort}(b, j+1, k); \\
    \end{array}
    \]
}

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

Worst case: \(n!\) e.g. \(2^{10}\)

Average case: \(n \log n\) e.g. \(15 + 2^{15}\)

\(2^{15} = 32768\)

\[
\begin{array}{c|c|c}
\text{pre: } b & \lll k & ? \\
\hline
\text{Post: } b & \lll \lll x & \rrr j & \rrr x & k \\
\end{array}
\]

Quicksort

CS1110 9 November 2010
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!
Tony Hoare, in 1968
Quicksort author

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.

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.

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

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

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

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

Separate concerns, and focus on one at a time.

Learn to read a program at different levels of abstraction.

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