BU CLA CS 530: Analysis of Algorithms
Spring 1999
Selected Solutions for Problem Set 1

Problem 1

(a) There is no difference between the best-case, the average-case, and the worst-case, running times for procedure MINIMUM. Let $T(n)$ be the number of comparisons used by MINIMUM(A, p, r), where $n = r - p + 1$ is the size of the subarray $A[p..r]$ processed. (Each execution of the operation "min" in line 2 counts for one comparison.) The recursion for $T(n)$ is:

$$T(n) \;=\; T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) \;+\; 1$$

for all $n \geq 2$, with boundary condition $T(2) = 1$.

(b) Assuming that $n = 2^k$ for some $k \geq 1$, the recurrence is $T(n) = 2 \cdot T(n/2) + 1$, which can be solved by one of the methods of Chapter 4. Using the iteration method (Section 4.2), for example, we obtain:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + 1 \\ &= 4 \cdot T(n/2^2) + 2 + 1 \\ &\vdots \\ &= 2^{k-1} \cdot T(n/2^{k-1}) + 2^{k-2} + \cdots + 2 + 1 \\ &= 2^{k-1} \cdot T(2) + 2^{k-2} + \cdots + 2 + 1 = \textstyle\sum_{i=0}^{k-1} 2^i \end{aligned}$$

By Chapter 3 (see for example equation (3.3), page 44) $\sum_{i=0}^{k-1} 2^i = 2^k - 1 = n - 1$. Hence $T(n) = n - 1$.

(c) The algorithm at the top of page 186 also requires $n - 1$ comparisons on an input array with $n \geq 1$ entries. If the running time for both algorithms is dominated by the execution of comparisons, the two algorithms are equally good. On the other hand, the implementation of the iterative algorithm in the book is simpler (no recursion, no stacks), and will probably run slightly faster than the recursive algorithm given in this problem.

Problem 2

(a) There is no difference between the best-case, the average-case, and the worst-case, running times for procedure MINIMUM. Let $T(n)$ be the number of comparisons used by MINIMUM(A, p, r), where $n = r - p + 1$ is the size of the subarray $A[p..r]$ processed. (Each execution of the operation "min" in line 2 counts for one comparison.) The recursion for $T(n)$ is:

$$T(n) \;=\; T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) \;+\; 1$$

for all $n \geq 2$, with boundary condition $T(2) = 1$.

(b) Assuming that $n = 2^k$ for some $k \geq 1$, the recurrence is $T(n) = 2 \cdot T(n/2) + 1$, which can be solved by one of the methods of Chapter 4. Using the iteration method (Section 4.2), for example, we obtain:

$$\begin{aligned}
T(n) &= 2 \cdot T(n/2) + 1 \\
&= 4 \cdot T(n/2^2) + 2 + 1 \\
&\vdots \\
&= 2^{k-1} \cdot T(n/2^{k-1}) + 2^{k-2} + \cdots + 2 + 1 \\
&= 2^{k-1} \cdot T(2) + 2^{k-2} + \cdots + 2 + 1 = \sum_{i=0}^{k-1} 2^i
\end{aligned}$$

By Chapter 3 (see for example equation (3.3), page 44) $\sum_{i=0}^{k-1} 2^i = 2^k - 1 = n - 1$. Hence $T(n) = n - 1$.

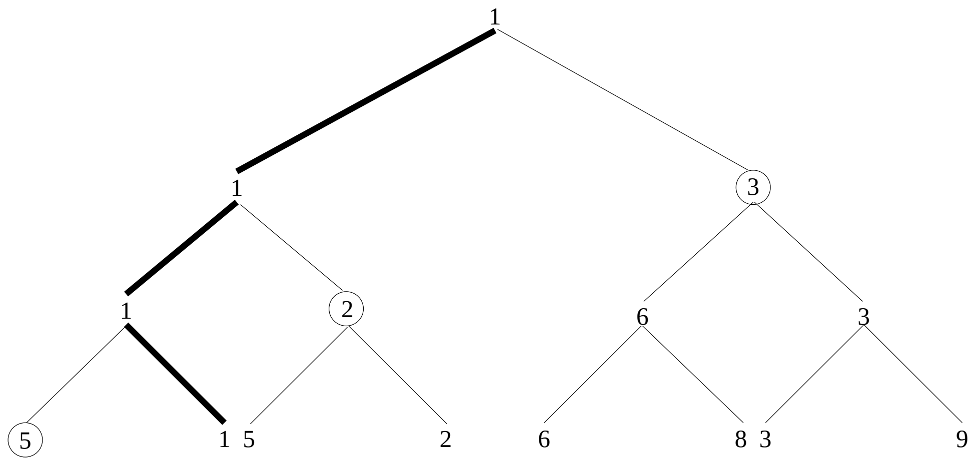
(c) The algorithm at the top of page 186 also requires $n-1$ comparisons on an input array with $n \geq 1$ entries. If the running time for both algorithms is dominated by the execution of comparisons, the two algorithms are equally good. On the other hand, the implementation of the iterative algorithm in the book is simpler (no recursion, no stacks), and will probably run slightly faster than the recursive algorithm given in this problem.

Problem 3

We explain the idea without implementing it (i.e. without writing the pseudocode of an algorithm). In particular, no attempt is made here to minimize the use of storage space.

First find the smallest of n numbers by a tournament: Compare all the numbers in pairs. Only the smaller of each pair can be a candidate to be the smallest of all n, so that the problem is now reduced to that of finding the smallest of $\lceil n/2 \rceil$ numbers. Compare these $\lceil n/2 \rceil$ numbers in pairs, and repeat the process again and again, until there is just one number left, which is necessarily the smallest of all n numbers.

To find the smallest number according to this method requires exactly $(n-1)$ comparisons, because each number except the smallest loses exactly once. (A number "loses" if it drops out of the race for the first spot, and each comparison eliminates exactly one number – there can't be more comparisons than there are numbers to be eliminated!) Moreover, in the process of winning, the smallest number was compared with at most $\lceil \log n \rceil$ other numbers. This is illustrated by the following example, where the input list is $5, 1, 5, 2, 6, 8, 3, 9$ (at the leaf nodes) and every internal node corresponds to one comparison:



The heavy line is the path followed by the winner, and the circled numbers are those against which the winner was matched.

Now, the second smallest path is among those that were compared with the smallest during the tournament. To find it, conduct a second tournament among these (at most) $\lceil \log n \rceil$ numbers to select a winner for the second spot, which requires at most $(\lceil \log n \rceil - 1)$ comparisons.

The total number of comparisons in the two tournaments is $(n-1) + (\lceil \log n \rceil - 1) = n + \lceil \log n \rceil - 2$ in the worst-case.

Problem 4

Part 1

$$\begin{aligned} Pr\{X=1\} &= \frac{1}{n+1} \\ Pr\{X=2\} &= \frac{1}{n+1} \\ &\vdots \\ Pr\{X=n-1\} &= \frac{1}{n+1} \\ Pr\{X=n\} &= \frac{2}{n+1} \end{aligned}$$

$$\begin{aligned} T_{avg}(n) &= \sum_{i=1}^{n} i \cdot Pr\{X=i\} \\ &= \frac{1}{n+1} + \frac{2}{n+1} + \ldots + \frac{n-1}{n+1} + \frac{2n}{n+1} \\ &= \frac{1+2+\ldots+n}{n+1} + \frac{n}{n+1} \end{aligned}$$

$$
\begin{aligned}
&= \frac{n(n+1)}{2(n+1)} + \frac{n}{n+1} \\
&= \frac{n^2+3n}{2(n+1)}
\end{aligned}
$$

Part 2

RANDOMIZED-LINEAR-SEARCH(A,B)
direction ← RANDOM(0,1)
if direction is 0 then pos ← LEFT-LINEAR-SEARCH(A,B)
else pos ← RIGHT-LINEAR-SEARCH(A,B)
return pos

LEFT-LINEAR-SEARCH(A,B)
n ← length(A)
for i ← 1 to n do if A[i] equals B then return i
return -1

RIGHT-LINEAR-SEARCH(A,B)
n ← length(A)
for i ← n to 1 do if A[i] equals B then return i
return -1

part 3

Given any array of size n where B is an element positioned on either end of the array. Since the random number generator is perfect, exactly half the time the element B will be found with one comparison. The other half of the time element B will be found with exactly n comparisons. Therefore for T^{exp}_{best} and T^{exp}_{worst} the running time will be $\frac{n+1}{2}$.

For T^{exp}_{avg} given an array of size n where B is positioned in the middle of the array. The running time will be $\frac{n+1}{2}$ no matter which direction the algorithm chooses.

Assaf Kfoury, Scott Savarese
Created: 99.01.29