S. Schwoon / D. Suwimonteerabuth | Winter Term 2008/09

Fundamental Algorithms

Solution Keys 2

1. Let m_k, n_k, and p_k be the values of m, n, and p, respectively, after entering the loop k times. To prove that $mn + p = MN$ is a loop invariant of the algorithm, we instead prove that $m_k n_k + p_k = MN$ for all $k \geq 0$. We proceed by induction.

   Basis Obviously, $m_0 n_0 + p_0 = MN$, since initially $p_0 = 0$, $M = m_0$, and $N = n_0$.

   Inductive step We prove that for all $k \geq 0$:

   $$m_{k+1} n_{k+1} + p_{k+1} = MN \; ,$$

   which, from the algorithm, can be rewritten to

   $$\left\lfloor \frac{m_k}{2} \right\rfloor 2n_k + p_k + x = MN \; , \tag{1}$$

   where $x = 0$, if m_k is even and $x = n_k$ if m_k is odd. We consider two cases depending on the value of m_k:

   (i) If m_k is even, Equation (1) becomes $m_k n_k + p_k = MN$, which is true by the induction hypothesis.

   (ii) If m_k is odd, Equation (1) becomes

   $$\begin{aligned} \left(\frac{m_k - 1}{2}\right) 2n_k + p_k + n_k &= MN \\ m_k n_k + p_k &= MN \; , \end{aligned}$$

   which is again true by the induction hypothesis.

   The loop invariant can be used to prove that the algorithm works correctly. With m and n are positive integers as the precondition, it can be readily seen that after the loop $m = 0$, thus $p = MN$.

2. The following algorithm computes f_n in $\Theta(n)$ time under the assumption that arithmetic operations take constant time.

   Input: Non-negative integer n
   Output: f_n

   $i := 1$; $j := 0$;
   for $k = 1$ **to** n **do**
       $j := i + j$;
       $i := j - i$;
   end
   return j;

Let j_k and i_k be the values of j and i, respectively, after entering the loop k times. We prove the following loop invariant: $j_k = f_k$ for all $k \geq 0$ and $i_k = f_{k-1}$ for all $k > 0$. Again, we proceed by induction.

Basis Obviously, $j_0 = 0 = f_0$, $j_1 = 1 = f_1$, and $i_1 = 0 = f_0$.

Inductive step

$$\begin{array}{rcll} j_{k+1} & = & i_k + j_k & \text{(Algorithm)} \\ & = & f_{k-1} + f_k & \text{(Induction hypothesis)} \\ & = & f_{k+1} & \end{array}$$

$$\begin{array}{rclcll} i_{k+1} & = & j_{k+1} - i_k & & & \text{(Algorithm)} \\ & = & i_k + j_k - i_k & = & j_k & \text{(Algorithm)} \\ & = & f_k & & & \text{(Induction hypothesis)} \end{array}$$

The loop invariant can be used to prove that the algorithm works correctly. With $j = 0$ and $i = 1$ as the precondition, it can be readily seen that after the loop $j = f_n$.

3. (a) The procedure `percolate`:

   Input: an array a, index i
   Output: if $1 \ldots i-1$ is a heap, then afterwards $1 \ldots i$ is a heap.

   $k := i$;
   repeat
       $j := k$;
       if $j > 1$ **and** $a[j/2] < a[k]$ **then**
           $k := j/2$;
       fi
       `swap`$(a[j], a[k])$;
   until $j = k$;

   (b) We recall from the lecture the procedure that constructs heaps by means of `heapify`:

   Input: an array a with indices $1 \ldots n$
   Output: a heap with elements from a

   for $i = \lfloor n/2 \rfloor$ **downto** 1 **do**
       `heapify`(a, n, i);
   end

   If a heap contains n nodes, we say that the root is at *level* $\lfloor \lg n \rfloor$, and children of a node at level j are at level $j-1$. It can be seen that there is always 1 node at level $k = \lfloor \lg n \rfloor$ (the root), 2 nodes at level $k-1$, ..., and 2^{k-1} nodes at level 1.

   Let $t(n)$ be the number of trips around the loop required to construct a heap of n elements. Since to sift down a node at level r, we make at most $r+1$ trips around the loop. Hence,

$$\begin{array}{rcl} t(n) & \leq & 2 \cdot 2^{k-1} + 3 \cdot 2^{k-2} + \ldots + (k+1) \cdot 2^0 \\ & < & -2^k + 2^{k+1}(2^{-1} + 2 \cdot 2^{-2} + 3 \cdot 2^{-3} + \ldots) \;=\; -2^k + 2^{k+2} \;<\; 4n \end{array}$$

   Therefore, the procedure in the lecture requires $\mathcal{O}(n)$ to construct a heap.

   Now, we consider another procedure that constructs a heap by means of `percolate`:

Input: an array a with indices $1 \ldots n$
Output: a heap with elements from a

for $i = 2$ **to** n **do**
    `percolate`(a, i);
end

Again, let $t(n)$ be the number of trips around the loop required to construct a heap of n elements. Since to percolate node i, we make at most $\lfloor \lg i \rfloor + 1$ trips around the loop. Hence,

$$t(n) \leq \sum_{i=2}^{n} \lfloor \lg i \rfloor + 1 \leq \sum_{i=1}^{n} \lg i + n$$

However, $n! \leq n^n$, thus $\lg n! \leq n \lg n$ for all $n \geq 1$. Therefore, $t \in \mathcal{O}(n \log n)$.

Similarly, one can show that $t \in \Omega(n \log n)$, which implies that new algorithm is asymptotically slower than the one presented in the lecture.