CS/Math 240: Intro to Discrete Math 3/8/2011

Homework 6

Instructor: Dieter van Melkebeek

Guidelines:

This assignment covers recurrences and asymptotic analysis, and reviews program correctness and analysis. It is due on 3/22 at the beginning of class. You *only need to turn in problems 1, 2, and 3.* Good luck!

Questions:

1. Relate the functions defined by the following expressions as tightly as possible using "=", "$\sim$", "Θ", and "O", where all logarithms involved have base 2:
$2^{(2^{\sqrt{\log n}})}$, $n^{\log n}$, $2^{\sqrt{n}}$, $2^{5\log n + \log\log n}\log(n^5)$, $4^{3\log n}$, $n^5 2^{2\log\log n} + 2^{\sqrt{n}}$, $2^{(\log n)^2}$, $n^5 \log^2 n$.

2. Consider the domain of all functions from the natural numbers to the positive reals. Prove of disprove the following statements.
   (a) $(\forall f, g)\, f = \Omega(g) \Rightarrow f^2 = \Omega(g)$
   (b) $(\forall f, g)\, f = O(g) \Rightarrow f + g = \Theta(g)$

3. (a) Prove that for any real $r \neq 1$ and any integer $k \geq 0$, $\sum_{i=0}^{k} r^i = \frac{r^{k+1}-1}{r-1}$.
   (b) Consider the function $f : \mathbb{N} \to \mathbb{N}$ defined by the following conditions.

$$f(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2f(\lfloor n/3 \rfloor) + \lfloor \sqrt{n} \rfloor & \text{otherwise} \end{cases}$$

   Derive that $f(n) = \Theta(n^c)$ for some constant c, and determine c.

4. Consider the following program specification:

   Input: Integer b and e with $b \leq e + 1$ and an array $A[b..e]$ of integers.

   Output: The array A sorted from smallest to largest.

   Consider the implementation on the next page, known as quiksort – the algorithm uses the last element of A as a pivot to rearrange the array such that all elements less than or equal to the pivot come first, followed by the elements larger than the pivot, and then recursively sorts those two parts of the array.

   (a) State and prove adequate loop invariants for the while loop.
   (b) Use the invariant to fill in the missing statement in line (9), and to prove partial correctness and termination.
   (c) Determine exactly the maximum number of times the test in line (3) is executed as a function of $n = e - b + 1$.

```
QuickSort($A, b, e$)
(1)     if $b \geq e$ then return
(2)     $p \leftarrow A[e]$; $i \leftarrow b$; $j \leftarrow e$
(3)     while $i < j$
(4)         if $A[i] > p$
(5)             $j \leftarrow j - 1$
(6)             swap $A[i]$ and $A[j]$
(7)         else
(8)             $i \leftarrow i + 1$
(9)     ???
(10)    QuickSort($A, b, i - 1$)
(11)    Quicksort($A, j + 1, e$)
```

5. Recall that for an array $A[b..e]$ of integers, a pair of positions (i, j) is in error if $b \leq i < j \leq e$ and $A[i] > A[j]$.

   Consider the following program to compute the number of pairs in error in a given array. The intuition for the program is similar to the explanation we gave at the end of Lecture 11 why every strategy for the unstacking game of Lecture 7 yields the same score.

```
CountPairsInError($A, b, e$)
(1)     if $b \geq e$ then return 0
(2)     $m \leftarrow \lfloor (b + e)/2 \rfloor$
(3)     $c \leftarrow$ CountPairsInError($A, b, m$) + CountPairsInError($A, m + 1, e$)
(4)     $B[b..e] \leftarrow A[b..e]$
(5)     $i \leftarrow b$; $j \leftarrow m + 1$; $k \leftarrow b$
(6)     while $i \leq m$ and $j \leq e$
(7)         if $B[i] \leq B[j]$ then $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$; $c \leftarrow c + j - m - 1$
(8)                          else $A[k] \leftarrow B[j]$; $j \leftarrow j + 1$
(9)         $k \leftarrow k + 1$
(10)    if $i \leq m$
(11)        $A[k..e] \leftarrow B[i..m]$
(12)        $c \leftarrow c + (m - i + 1) \cdot (e - m)$
(13)    return $c$
```

   (a) Show that CountPairsInError correctly implements the following specification:

   Input: Integers b and e with $b \leq e + 1$ and an array $A[b..e]$ of integers.

   Output: The number of pairs in error in the input array.

   Side Effect: At the end, A is the sorted version of the original A.

   As usual, this involves establishing partial correctness and termination.

   (b) Show that the total number of elementary steps is $O(n \log n)$, where $n = e - b + 1$.