Homework #2: Divide-and-Conquer Algorithms, Recurrence Relations
Due Date: Wednesday, 27 January 2016

Homework policies

For the complete set of homework policies in CS161 please consult the appropriate section of the class web page `http://cs161.stanford.edu`.

Problem 1. Integer multiplication [30 points]

Let u and v be two n bit numbers where for simplicity n is a power of 2. The traditional multiplication algorithm requires $O(n^2)$ operations. A divide–and–conquer based algorithm splits the numbers into two equal parts, computing the product as

$$uv = (a2^{n/2} + b)(c2^{n/2} + d) = ac2^n + (ad + bc)2^{n/2} + bd$$

The multiplications ac, ad, bc, and bd are done using the algorithm recursively. Determine this algorithm's running time. What is the running time if $ad + bc$ is computed as $(a+b)(c+d) - ac - bd$? Note that multiplication by powers of 2 can be implemented via linear-time shifts for binary numbers.

Problem 2. Give asymptotic upper and lower bounds or $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible and justify your answers. [36 points, 6 points per part]

(a) $T(n) = 3T(n/2) + n \lg n$.

(b) $T(n) = 3T(n/3 + 5) + n/2$.

(c) $T(n) = 2T(n/2) + n/\lg n$.

(d) $T(n) = T(n-1) + 1/n$.

(e) $T(n) = T(n-1) + \lg n$.

(f) $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

Problem 3. Stack depth for QuickSort [30 points, 10 points per part]

The QUICKSORT algorithm contains two recursive calls to itself. After the call to PARTITION, the left subarray is recursively sorted and then the right subarray is recursively sorted. The second recursive call in QUICKSORT is not really necessary; it can be avoided by using an iterative control structure. This technique, called **tail recursion**, is provided automatically by good compilers. Consider the following version of QuickSort, which simulates tail recursion.

QUICKSORT′(A, p, r)

while $p < r$
  do ▷ Partition and sort left subarray
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT′$(A, p, q-1)$
    $p \leftarrow q+1$

(a) Argue that QUICKSORT′$(A, 1, \mathit{length}[A])$ correctly sorts the array A.

Compilers usually execute recursive procedures by using a **stack** that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is invoked, its information is **pushed** onto the stack; when it terminates, its information is **popped**. Since we assume that array parameters are actually represented by pointers, the information for each procedure call on the stack requires $O(1)$ stack space. The **stack depth** is the maximum amount of stack space used at any time during a computation.

(b) Describe a scenario in which the stack depth of QUICKSORT′ is $\Theta(n)$ on an n-element input array.

(c) Modify the code for QUICKSORT′ so that the worst-case stack depth is $\Theta(\lg n)$. Maintain the $O(n \lg n)$ expected running time of the algorithm.