Name: ____

1. (4 points) The procedure Euclid computes the greatest common divisor of two positive integers. Use the following loop invariant to prove that the call Euclid(A, B), $0 \leq B < A$, correctly computes the gcd of A and B:

$$\gcd(a, b) = \gcd(A, B) \text{ and } 0 \leq b < a.$$

You may use the following facts:

- $\gcd(b, a \bmod b) = \gcd(a, b)$
- $\gcd(a, 0) = a$

```
EUCLID(a, b)
1  while b ≠ 0
2      r = a mod b
3      a = b
4      b = r
5  return a
```

Solution

Initialization. Before the first execution of the while loop, $a = A$ and $b = B$, so $\gcd(a, b) = \gcd(A, B)$ and $0 \leq b < a$.

Maintenance. Suppose the invariant holds at the start of an iteration; that is, $\gcd(a, b) = \gcd(A, B)$ and $0 \leq b < a$. The variable r is set equal to $a \bmod b$, which is the remainder upon dividing a by b; therefore, $0 \leq r < b$. Lines 3 and 4 update a and b; let a' and b' be the new values after these lines execute. Then since $b' = r$ and $a' = b$, $0 \leq b' = r < b = a'$, or $0 \leq b' < a'$. Also,

$$\gcd(a', b') = \gcd(b, r) = \gcd(b, a \bmod b) = \gcd(a, b) = \gcd(A, B),$$

so $\gcd(a', b') = \gcd(A, B)$, and we conclude that the invariant holds at the start of the next iteration.

Termination. Initially, $b = B$, a non-negative integer. At each iteration, b is reduced but remains non-negative; therefore it must eventually be true that $b = 0$, at which point the invariant gives us

$$a = \gcd(a, 0) = \gcd(A, B),$$

so we can conclude that the algorithm correctly computes $\gcd(A, B)$.

(continued on other side)

2. (4 points) The function Recursive-Max recursively computes the maximum value in a numeric sub-array $A[p \,..\, r]$. Initially, it is called with $p = 1$ and $r = A.\mathit{length}$ to determine the maximum value in the entire array.

Recursive-Max(A, p, r)

if $p < r$
    $q = \lfloor (p + r)/2 \rfloor$
    $x =$ Recursive-Max(A, p, q)
    $y =$ Recursive-Max$(A, q + 1, r)$
    return Max(x, y) // two-argument Max() function
return $A[p]$

(*a*) Derive a recursion for the running-time of Recursive-Min.

(*b*) Use a recursion tree to 'guess' an asymptotic bound for the recursion.

(*c*) Use the substitution method to prove your guess is correct.

Solution

(a) $T(n) = 2T(n/2) + \Theta(1)$ since the algorithm makes two recursive calls of size $n/2$ and the non-recursive work is all constant-time.

(b) The tree is a complete binary tree with $\lg n$ levels. Every node has cost c. Therefore, the cost of level k is $c2^k$, $k = 0, 1, \ldots, \lg n$, and the sum over all the nodes is

$$\sum_{k=0}^{\lg n} c2^k = c(2^{\lg n + 1} - 1) = c(2n - 1).$$

We conjecture that the running time is $\Theta(n)$.

(c) It is sufficient to prove the Big-Oh bound; proof of the Big-Omega bound is similar. Let $n > 1$ and suppose that for all $1 \le k < n$ we have that $T(k) \le ck$ for some constant $c > 0$ [this is the *inductive hypothesis*]. We need to show that $T(n) \le cn$.

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(1) \\ &\le 2T(n/2) + d \text{ (for some positive constant } d \text{ and } n \text{ sufficiently large)} \\ &\le 2c(n/2) + d \\ &= cn + d \end{aligned}$$

Unfortunately, there is no way to make $cn + d \le cn$ since $d > 0$. We start over with a stronger inductive hypothesis: $T(k) \le ck - d$ for positive constants c and d. Then

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(1) \\ &\le 2T(n/2) + e \text{ (for some positive constant } e \text{ and } n \text{ sufficiently large)} \\ &\le 2c(n/2) - 2d + e \\ &\le cn - d \text{ (so long as } d \ge e) \end{aligned}$$

Note: The students are not required to address the base case. We discussed in class that for these sorts of problems, we can always increase the constant c to ensure that the bound is satisfied for any finite number of initial values.

3. (2 points) Use the Master Theorem to determine the running times for each of the following recurrence relations:

(*a*) $T(n) = 4T(n/2) + \Theta(n)$ (Schoolbook-Multiply)

(*b*) $T(n) = 2T(n/2) + \Theta(n)$ (Maximum-Subarray)

Solution

(a) $a = 4$, $b = 2$, so $\log_b a = \lg 4 = 2$ and $n^{\log_b a} = n^2$. Therefore, $f(n) = \Theta(n) = O(n^{\log_b a-\epsilon}) = O(n^2)$, so by case (a) of the Master Theorem, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

(b) $a = 2$, $b = 2$, so $\log_b a = \lg 2 = 1$ and $n^{\log_b a} = n$. Therefore, $f(n) = \Theta(n) = \Theta(n^{\log_b a})$, so by case (b) of the Master Theorem, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n)$.

Theorem (Master Theorem)**.** *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence*

$$T(n) = aT(n/b) + f(n)$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

(a) *If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*

(b) *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.*

(c) *If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*