Name: _____

1. (4 points) The procedure MUL computes the product of two positive integers. Use the following loop invariant to prove that the call MUL(A, B) correctly computes the product $A \cdot B$:

At the beginning of an iteration, $xy + p = A \cdot B$

```
MUL(x, y)
1  p = 0
2  while y ≠ 0
3        if (y mod 2) == 0
4              x = 2 · x
5              y = y div 2
6        else p = p + x
7              y = y − 1
8  return p
```

Solution

Initialization. Before the first execution of the while loop, $p = 0$, $x = A$, and $y = B$, so

$$xy + p = A \cdot B + 0 = A \cdot B.$$

Maintenance. Suppose the invariant holds at the start of an iteration; that is, $xy + p = A \cdot B$. First consider the case that $y \bmod 2 = 0$. Then y is even and $y = 2z$ for some integer z. Let x' and y' be the updated values of x and y, so $x' = 2x$ and $y' = y \text{ div } 2 = z$. Then

$$x'y' + p = (2x)z + p = x(2z) + p = xy + p = A \cdot B,$$

so the invariant still holds. In the case that y is odd, let $p' = p + x$ and $y' = y - 1$ be the updated values of p and y. Then

$$xy' + p' = x(y - 1) + (p + x) = xy - x + p + x = xy + p = A \cdot B,$$

so the invariant holds in this case as well. We've shown that in either case, if the invariant is true at the start of an iteration, it is true at the start of the next iteration.

Termination. Initially, $y = B$, a positive integer. At each iteration, y is halved if it is even and reduced by one if it is odd; thus, the updated value of y is always a non-negative integer. Since y is reduced at every iteration but remains non-negative, we must eventually have $y = 0$, at which point the while loop terminates. The loop invariant gives us that, at termination, $xy + p = A \cdot B$, but since $y = 0$, this reduces to $y = A \cdot B$, so p is the product of A and B and the algorithm functions correctly.

(continued on other side)

2. (4 points) BINARY-SEARCH implements the standard algorithm to search for a value v in a sorted array A. The parameters p and r indicate that the subarray $A[p\,..\,r]$ is to be searched; initially, the function would be called with $p = 1$ and $r = A.length$.

```
BINARY-SEARCH(A, v, p, r)
 1  // Assume A is sorted in ascending order
 2  if p < r
 3       q = ⌊(p + r)/2⌋
 4       if v ≤ A[q]
 5            return BINARY-SEARCH(A, v, p, q)
 6       else
 7            return BINARY-SEARCH(A, v, q + 1, r)
 8  if A[p] == v
 9       return p
10  else
11       return NIL
```

(*a*) Derive a recursion for the running-time of BINARY-MULTIPLY.

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

(b) The tree consists of just a trunk with $\lg n$ levels:

$$c \to c \to c \to \cdots \to c.$$

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

(c) It is sufficient to prove the Big-Oh bound; proof of the Big-Omega bound is similar. Let $n > 2$ and suppose that for all $2 \le k < n$ we have that $T(k) \le c \lg k$ for some constant $c > 0$ [this is the *inductive hypothesis*]. We need to show that $T(n) \le c \lg n$.

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

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*) $T(n) = 2T(n/2) + \Theta(n)$ (MERGE-SORT)

(*b*) $T(n) = 3T(n/2) + \Theta(n)$ (KARATSUBA)

Solution

(a) $a = 2$, $b = 2$, 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, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n)$.

(b) $a = 3$, $b = 2$, so $\log_b a = \lg 3$ and $n^{\log_b a} = n^{\lg 3}$. Since $\lg 3 \approx 1.58$, $f(n) = \Theta(n)$ is asymptotically smaller than $n^{\lg 3 - \epsilon}$ for some positive epsilon (for example, $\epsilon = 0.1$). Therefore, by case (a) of the Master Theorem, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\lg 3})$.

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 $T(n) = \Theta(f(n))$.*