CMSC 451 – HW I

1. Consider the following iterative function:

int pentagonal(int n)

{

int result = 0;

for (int i = 1; i <= n; i++)

result += (3 \* i - 2);

return result;

}

Rewrite the function pentagonal using recursion and add preconditions and postconditions as comments.

Then prove by induction that the recursive function you wrote is correct.

**// precondition: n >= 1**

**int pentagonal(int n)**

**{**

**if (n == 1)**

**return n;**

**return pentagonal(n-1) + (3 \* n - 2);**

**}**

**// postcondition: Returns (3n^2 - n)/2**

**Base case: n = 1**

**(3\*1^2 - 1)/2**

**= (3 \* 1 - 1) / 2**

**= 2 / 2**

**= 1**

**Inductive case:**

**assume true for n = k-1**

**prove true for n = k, where n > 1**

**return value = pentagonal(k-1) + 3\*k-2**

**= ((3(k-1)^2) - (k - 1))/2 + 3k - 2**

**= (3(k-1)(k-1) - k + 1 + 6k - 4) / 2**

**= (3(k^2 - 2k + 1) + 5k - 3) / 2**

**= (3k^2 - 6k + 3 + 5k - 3) / 2**

**= (3k^2 - k) / 2**

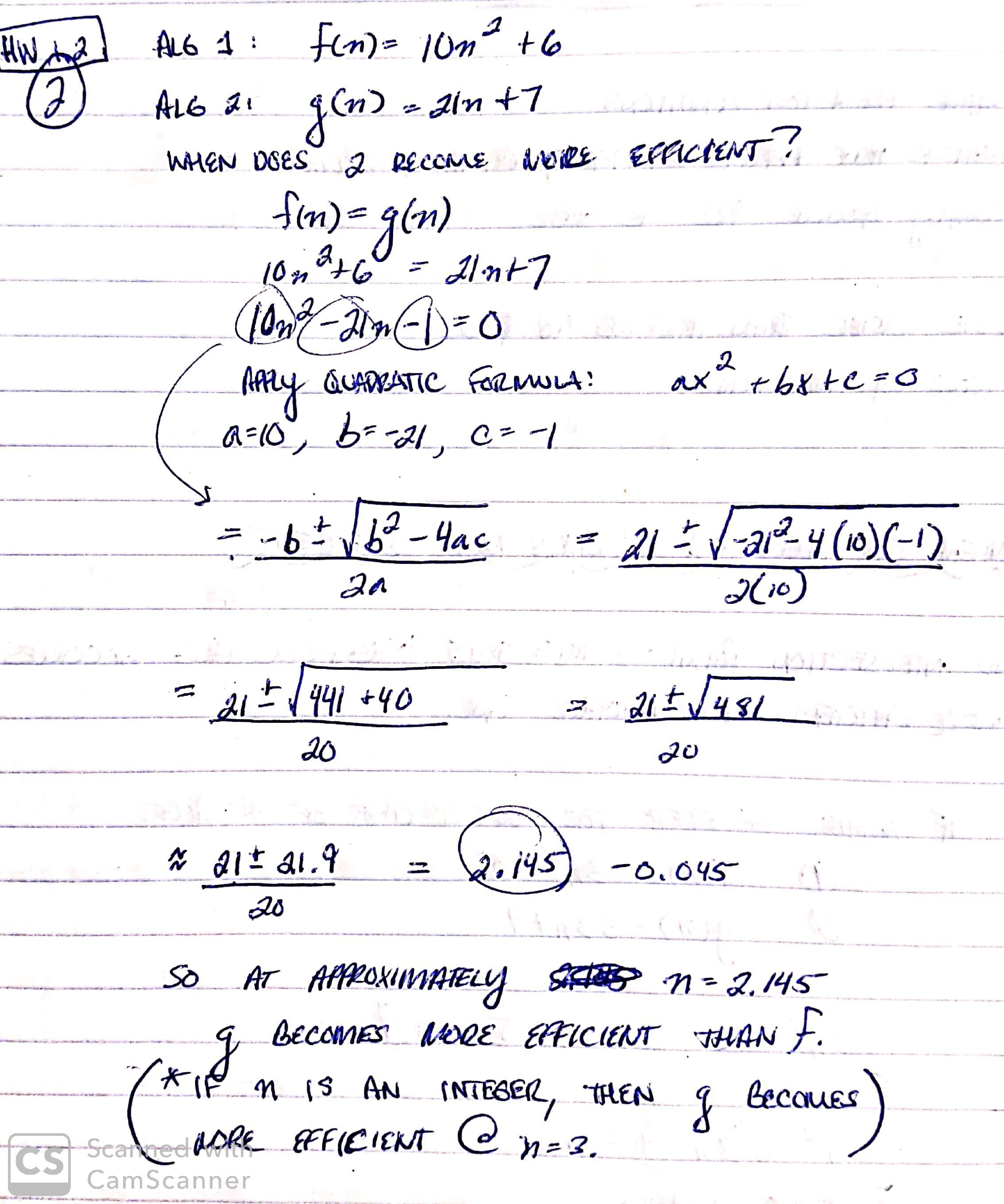
**QED**

2. Suppose the number of steps required in the worst case for two algorithms are as follows:

Algorithm 1: f(n) = 10n^2 + 6

Algorithm 2: g(n) = 21n + 7

Determine at what point algorithm 2 becomes more efficient than algorithm 1.



3. Given the following function that evaluates a polynomial whose coefficients are stored in an array:

double evaluate(double[] coefficients, double x) {

double result = coefficients[0];

double power = 1;

for (int i = 1; i < coefficients.length; i++) {

power = power \* x; **// n - 1 mults**

result = result + coefficients[i] \* power; **// n - 1 mults, n - 1 adds**

}

return result;

}

Let n be the length of the array.

Determine the number of additions and multiplications that are performed in the worst case as a function of n.

**n-1 + n-1 + n-1 = 3n - 3**

4. Given the following recursive binary search algorithm for finding an element in a sorted array of integers:

int recursiveBinarySearch(int[] array, int target, int left, int right) {

if (left > right)

return -1;

int middle = (left + right) / 2;

if (array[middle] == target)

return middle;

if (array[middle] > target)

return recursiveBinarySearch(array, target, left, middle - 1);

return recursiveBinarySearch(array, target, middle + 1, right);

}

Assume n is the length of the array.

Find the initial condition and recurrence equation that expresses the execution time for the worst case of this algorithm and then solve that recurrence.

**INITIAL CONDITION:**

**n = 1**

**t(1) = 1**

**RECURRENCE EQUATION:**

**If n > 1, then the worst-case number of steps taken will be *log n*:**

**t(n) = t(n/2) + 1, for n >= 2**

**t(8) = t(4) + 1 = 4**

**t(4) = t(2) + 1 = 3**

**t(2) = t(1) + 1 = 2**

**t(1) = 1**

**t(n) = t(log(n)) + 1**

**t(n) = O(log(n)) + O(1)**