Name ________________________________

Problem 1 (20 points (10 each))

The following are proposed proofs of the given claims. Below each one, write whether the proof is valid or not. If it is *not* valid, explain why. Only a sentence or so should be needed to explain why a proof does not work.

(a) *Claim.* $n^2 - 56$ is not $O(n)$.

Proof. In order to show that $n^2 - 56$ is not $O(n)$, we need to show that there do not exist real numbers $c > 0$, $n_0 \geq 0$ such that $\forall n \geq n_0 : n^2 - 56 \leq c \cdot n$. Notice that $n^2 - 56$ and n intersect at $n = 8$, so consider $n_0 = 9$ and $c = 1$. But when $n = 10$, $n^2 - 56 = 100 - 56 = 44$, which is greater than 10. So $\forall n \geq 9 : n^2 - 56 \leq 1 \cdot n$ does not hold, so $n^2 - 56$ is not $O(n)$. $\square$

(b) There will be a second claim here on the quiz.

Problem 2 (20 points)

In this problem, you will prove that $2n^2 + 3 = O(n^3)$ using the definition of big O. Follow the three steps carefully.

(5 points) Write down the definition of big O.

(10 points) Give a c and a n_0 that can be used to prove that $2n^2 + 3 = O(n^3)$.

(5 points) Explain what it would mean for this c and n_0 to work in a proof that $2n^2 + 3 = O(n^3)$, and *very briefly* explain why they do (write one sentence, draw a graph, etc).

Problem 3 (20 points)

(a) For the following algorithm give a proposed function representing the number of primitive operations for the algorithm in terms of the input size, addressing each line and/or loop of the algorithm. You do not need to be precise counting constant numbers of primitive operations (e.g., figuring out exactly how many primitive operations a single line does). However, you should try to be precise about how many times a loop runs.

Algorithm 1

```
for i = 1 to 2n do
    j = n;
    while j > 1 do
        j = j/3;
```

Algorithm 1 takes $f(n) =$ primitive operations.

(b) For the $f(n)$ you gave in (a), give the "tightest" (aka asymptotically smallest) $g(n)$ such that $f(n) = O(g(n))$.

$f(n) = O(\qquad\qquad)$

Problem 4 (20 points)

Recall the recursive binary search algorithm from lecture:

Algorithm 2 `binarySearch`(A[1...n], x)

1: **if** $|A| = 0$ **then**
2:     **return** False
3: **else**
4:     $middle = \lfloor \frac{|A|}{2} \rfloor$
5:     **if** $A[middle] = x$ **then**
6:         **return** True
7:     **else if** $A[middle] > x$ **then**
8:         `binarySearch`(A[1..$middle - 1$], x)
9:     **else**
10:         `binarySearch`(A[$middle + 1$...1], x)

Notice that on line 4, `binarySearch` calculates *middle* as $\lfloor \frac{|A|}{2} \rfloor$, meaning that if $\frac{|A|}{2}$ is non-integer, it is rounded down to the nearest integer.. To give a rigorous analysis of the worst-case runtime of `binarySearch`, we should account for the fact that the algorithm behaves slightly differently when n is odd versus when n is even.

In this problem you will give a recurrence relation for the worst-case runtime of `binarySearch`.

(a) Give the base case of the recurrence relation. Make sure you use the smallest possible input to the algorithm.

(b) Give the recursive case of the recurrence relation. Hint: your recurrence relation should address both the case where n is even and n is odd. That is, fill in the blanks below. You don't need to be precise about counting the number of operations if it is constant.

$$T(n) = \begin{cases} \qquad\qquad\qquad\qquad\qquad , \text{if } n \text{ is even} \\ \qquad\qquad\qquad\qquad\qquad , \text{if } n \text{ is odd} \end{cases}$$