2210: Assignment 1 - Allison So

1. a) We need to prove that $\frac{1}{2n-1}$ is $O(\frac{1}{n})$. We must find constants c > 0 and integer $n_0 \geq 1$ such that

$\frac{1}{2n-1} \leq c(\frac{1}{n})$ for all $n \geq n_0$

b) Simplify the inequality. Multiply both sides by n.

$\frac{n}{2n-1} \leq c$ for all $n \geq n_0$

c) c = 1

$2n - 1 \leq n$

$1 \leq n$ = n for all $n \geq n_0$, the inequality is valid for all values of n larger than or equal to 1, therefore n_0 can be 1 (n_0 = 1). Since we have found c and n_0 values that are true for this inequality, then we have proven that $\frac{1}{2n-1}$ is $O(\frac{1}{n})$.

2. a) We need to prove that $f(n) + k$ is $O(g(n))$. We must find constants c and k such that

$f(n) \leq c' \bullet g(n)$ for all $n \geq n_0$

b) Simplify.

$c \bullet g(n) + k \leq c' \bullet g(n) + k$

$c \bullet g(n) \leq c' \bullet g(n)$

c) c = 3, k = 4, c' = 4, n_0 = 1

$3 \bullet g(n) \leq 4 \bullet g(n)$ for all $n \geq 1$. Therefore, the inequality is valid for all values of n larger than or equal to 1. Since we have found c, c', k and n_0 values that are true for this inequality, then we have proven that f(n) + k is O(g(n)).

3. We will assume that $\frac{1}{n}$ is $O(\frac{1}{n^2})$ for all $n \geq n_0$ is a true claim.

$\frac{1}{n} \leq c \bullet \frac{1}{n^2}$ for all $n \geq n_0$

$n \leq c \bullet n^2$

$1 \leq cn$

$n \leq \frac{1}{c}$

This simplified inequality is valid only for values of n that are at most $\frac{1}{c}$, thus providing a contradiction to the claim that says, for all $n \geq n_0$. So, with no constant values such that $n \leq \frac{1}{c}$ for all $n \geq n_0$, $\frac{1}{n}$ is not $O(\frac{1}{n^2})$.

4.

The algorithm does not terminate. For example, given array A { 1,2,3,4,5,6,7,8,9,10 }, first = 0, last = 9, x = 12. numValues = 10, third = 3, twoThird = 6. Else return, numValues = Å, third = 8, twoThird = 9. Else return, numValues = 2, third = 9, twoThird = 10. After searching the values the program still does not terminate, because there is no system in place in case there is no match found.

5.

The output of this algorithm is incorrect because using this algorithm, if the value of x is found in the array at index 0, it will be counted twice.

For example, given array A { 3,9,2,6,1,5,4,8 }, first = 0, last = 7, x = 3. Just by skimming we see that there are no other copies of the value 3, so using these values and the given algorithm, I would get c = 1. Then in the loop, an extra 1 is added to the value of c as the loop begins from index 0, but it should only do so if the value of index 0 is not equal to the value of x. And so it follows that, if the value of index 0 is equal to the value of x, then the loop should not begin at 0, but at index 1.

6.

```
duplicateFound ← true                       // 1
i ← 0                                       // 1
while duplicateFound = true do {            // 4n
        if L[i] = L[i + 1] then return true
        else duplicateFound ← false         // 1
        if i < n − 1 then i ← i + 1         // 3
}
return duplicateFound                       // 1
```

f(n) = (4n + 4) + 3 The order of this complexity is O(n).

I computed the time complexity by going through the code, counting the number of primitive operations on every line. Ensuring that when a loop was indicated, the order was increased by one and all loop operations were kept separate from those outside.

7.

n	Linear Search	n	Quadratic Search	n	Factorial Search
5	113 ns	5	352 ns	7	2560944 ns
10	169 ns	10	550 ns	8	14402615 ns
100	1293 ns	100	11787 ns	9	75897496 ns
1000	6177 ns	1000	129602 ns	10	603844635 ns
10000	12773 ns	10000	9940241 ns	11	6742303785 ns
100000	29772 ns			12	84701679657 ns