CS 2210 Data Structures and Algorithms Assignment 1 (20 marks)
Due: September 28 at 11:55 pm.
Important: No late concept assignments will be accepted

Please submit through OWL a pdf file or an image file with your solution to the assignment. You are encouraged to type your answers. If you decide to submit hand-written answers to the questions please make sure that the TA will be able to read your solutions. If the TA cannot read your answers you will not be given credit for them.
Remember that concept assignments must be submitted by the due date; **no late concept assignments will be accepted** unless you have an academic accommodation.

For questions 1 and 2 proceed as follows:

1. First explain what needs to be proven: "We need to find constants $c > 0$ and integer $n_0 \geq 1$ such that ...".
2. Simplify the above inequality.
3. Determine values for c and n_0 as required. Do not forget the value for n_0.

For question 3, **you must use a proof by contradiction**:

- First give the claim that you will assume true and from which you will derive a contradiction.
- Use the definition of order to write the inequality from which you will derive the contradiction.
- Simplify the inequality and explain how you derive a contradiction from it.

---

1. (3 marks) Use the definition of "big Oh" to prove that $\frac{1}{2n-1}$ is $O(\frac{1}{n})$.
2. (3 marks) Let $f(n)$ and $g(n)$ be positive functions such that $f(n)$ is $O(g(n))$ and $g(n) \geq 1$ for all $n \geq 1$. Using the definition of "big Oh" show that $f(n) + k$ is $O(g(n))$, where $k > 0$ is constant.

   Hints. Since $f(n)$ is $O(g(n))$ then there are constants $c' > 0$ and integer $n_0' \geq 1$ such that $f(n) \leq c'g(n)$ for all $n \geq n_0'$. Hence, $f(n) + k \leq c' \cdots$. Also, recall that $g(n) \geq 1$ for all $n \geq 1$.
3. (2 marks) Use the definition of "big Oh" to prove that $\frac{1}{n}$ is not $O(\frac{1}{n^2})$.
4. (4 marks) Consider the following algorithm for the search problem.

   Algorithm triSearch$(L, first, last, x)$
   Input: Array L storing integer values sorted in non-increasing order, index *first* of the 1st value in L, index *last* of the last value in L, and integer value x.
   Out: Position of x in L if x is in L, or -1 if x is not in L

   if *first* $>$ *last* **then return** -1
   else {
   $numValues = last - first + 1$
   $third \leftarrow first + \lfloor \frac{numValues}{3} \rfloor$
   if $x = L[third]$ **then return** *third*
   else if $x < L[third]$ **then return** triSearch$(L, first, third, x)$
   else {
   $twoThird \leftarrow first + 2 \times \lfloor \frac{numValues}{3} \rfloor$
   if $x = L[twoThird]$ **then return** *twoThird*
   else if $x < L[twoThird]$ **then return** triSearch$(L, third, twoThird, x)$
   else return triSearch$(L, twoThird, last, x)$
   }
   }

Prove that either (i) the algorithm terminates, or (ii) the algorithm does not terminate.

- Note that to prove that the algorithm terminates you cannot just give an example and show that the algorithm terminates on it; instead you must prove that when the algorithm is given as input **any** array L as specified and **any** values *first*, *last*, and x, the algorithm will always end.
- However, to show that the algorithm does not terminate it is enough to show an example for which the algorithm does not finish. In this case **you must explain** why the algorithm does not terminate.

5. (4 marks) Consider the following algorithm.

Algorithm copies(L, n, x)
Input: Array L storing $n > 0$ integer values, and value x
Output: The number of copies of the value x stored in L.
if $x = L[0]$ **then** $c \leftarrow 1$
else $c \leftarrow 0$
for $i \leftarrow 0$ **to** $n - 1$ **do**
  if $x = L[i]$ **then** $c \leftarrow c + 1$
return c

Note that this algorithm terminates, as the **for** loop is repeated only n times.

Prove that either (i) the algorithm outputs the correct answer, or (ii) the algorithm does not output the correct answer.

- To prove that the algorithm is correct you cannot just give an example and show that the algorithm computes the correct output. You must prove that when the algorithm is given as input **any** array L storing $n > 0$ integer values and **any** value x, the algorithm correctly outputs the number of times that the value x appears in L.
- However, to show that the algorithm is incorrect it is enough to show an example for which the algorithm produces the wrong answer. In this case **you must explain** why the output is incorrect.

6. (4 marks) Consider the following algorithm.

Algorithm duplicated(L, n)
Input: Array L storing $n > 1$ integer values
duplicateFound $\leftarrow$ true
$i \leftarrow 0$
while *duplicateFound* = true **do** {
  if $L[i] = L[i + 1]$ **then return** true
  else *duplicateFound* = false
  if $i < n - 1$ **then** $i \leftarrow i + 1$
}
return *duplicateFound*

Compute the time complexity of this algorithm in the worst case. You **must explain** how you computed the time complexity, and you **must give** the order of the complexity.

7. (2 marks) **Optional question**. Download from OWL the java class `Search.java`, which contains implementations of 3 different algorithms for solving the search problem:

- `LinearSearch`, of time complexity $O(n)$.
- `QuadraticSeach`, of time complexity $O(n^2)$.

- `FactorialSearch`, of time complexity $O(n!)$.

Modify the `main` method so that it prints the worst case running times of the above algorithms for the following input sizes:

- `FactorialSearch`, for input sizes $n = 7, 8, 9, 10, 11, 12$.
- `QuadraticSeach`, for input sizes $n = 5, 10, 100, 1000, 10000$.
- `LinearSearch` for, input sizes $n = 5, 10, 100, 1000, 10000, 100000$.

Fill out the following tables indicating the running times of the algorithms for the above input sizes. You do not need to include your code for the Search class.

n	Linear Search
5	
10	
100	
1000	
10000	
100000	

n	Quadratic Search
5	
10	
100	
1000	
10000	

n	Factorial Search
7	
8	
9	
10	
11	
12	