Please enter your name and uID below.

Name:

uID:

Collaborators, if any, and how you collaborated:

Submission notes

- **(PS0 specific)** Significant points be rewarded for answers that give a clear legitimate effort. The goal of this assignment is to learn the course expectations for written proofs.
- **(PS0 specific)** Due at 11:59 pm (midnight) on Thursday, Sep 1, 2021. *No late submissions will be accepted.*
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

Sample problems and solutions; do not submit on Gradescope.

Please note that in 4150, proofs should be written in English in paragraph format (they should not be in bulleted lists), and equations should be used sparingly. Code/pseudocode should not be included in proofs. If the problem provides code/pseudocode, you may (and should) reference it by line number in arguments about correctness or complexity (e.g. "line 7 is executed n times"). If you have questions about what statements may be assumed versus need justification in a proof, please post to Instructors on Piazza and we will be happy to clarify. The following two short examples are meant to help you understand the level of detail we expect in problem set solutions in this course.

A. Prove by induction that every finite, nonempty set of real numbers has a largest element.

   Proof. Let S be a finite, non-empty subset of $\mathbb{R}$. We induct on the size (cardinality) of S. If S has size 1, then $S = \{a\}$ for some $a \in \mathbb{R}$ and its largest element is a. Assume that for some $k \geq 1$, if a set has size k, then it has a largest element. Consider a set S of size $k + 1$. Since $k \geq 1$, S has at least two elements. Choose $s \in S$ and let $S' = S \setminus \{s\}$. Then S' has $k \geq 1$ elements. By the inductive hypothesis S' must have a largest element, which we call a. If $a > s$, then s is the largest element of S. Otherwise a is the largest element of S. □

B. Describe (in English) an algorithm for finding a query element in a sorted array using binary search. You do not need to argue correctness or analyze its runtime.

   Start with two "pointers" (which we call left and right), initialized to 0 and length-1 respectively. Then, repeat the following process:

   Find the element "halfway between" where left and right are "pointing" in the array; specifically, set middle to be the average of left and right, rounded down to the nearest integer. Consider the item in the array at index middle, and compare it to our query element: If right == left and the item at middle (which we note is equal to right) does not equal our query, return false because our query does not exist in the array. If our query is greater than the item at index middle, set left to middle + 1 and repeat. This eliminates the left half of our subarray. If our query is less than the item at index middle, set right to middle - 1 and repeat. This eliminates the right half of our subarray. If our query is equal to the item at middle, we have found our item. Return true.

1. (Total Induction)

Algorithm 1: SimpleTotal(n)

```
total ← 0
for $i \leftarrow 5n$ to $7n$ do
    for $j \leftarrow 1$ to $i$ do
        total ← total +1
    end
end
```

The number of times `total` is updated in SimpleTotal above is exactly $1 + \sum_{k=5n}^{7n} k$. Prove that this summation is $O(n^2)$.

Prove this in three steps. First *guess* what polynomial you think the sum above should be equal to by using common summation rules, or by trying some cases yourself and extrapolating, or by some other way. It's OK if your initial guess is wrong! Second, *prove* the sum is equal to your conjectured polynomial using induction. Finally, show that your polynomial is $O(n^2)$ using the definition of big-O.

2. (Vaccination) You are in charge of determining the best order in which to vaccinate hospital employees. You're given a priority score for each employee, where employees with higher priorities should be vaccinated *sooner*.

   However, this score is currently flawed, because it doesn't account for whether or not employees are working remotely. You decide that it doesn't make sense to vaccinate any at-home workers before the in-hospital employees, regardless of their original priority and designed the following algorithm that you believe accomplishes this goal.

   Algorithm 2: Vaccination Scheduler

   ```
   Input: Arrays of employee work locations ($loc$) and priority scores ($ranks$)
   1  $n \leftarrow$ length of $loc$
   2  for $i \leftarrow 0$ to $n-2$ do
   3      for $j \leftarrow 0$ to $n-i-2$ do
   4          if $loc[j+1]$ is in-hospital, and $loc[j]$ is at-home then
   5              swap $ranks[j+1] \leftrightarrow ranks[j]$
   6              swap $locs[j+1] \leftrightarrow locs[j]$
   7          else if $locs[j] == locs[j+1]$ then
   8              if $ranks[j] < ranks[j+1]$ then
   9                  swap $ranks[j+1] \leftrightarrow ranks[j]$
   10                 swap $locs[j+1] \leftrightarrow locs[j]$
   11     end
   12 end
   ```

   Prove *by contradiction* that this algorithm works. That is, prove that in the order produced by this algorithm, all in-hospital workers are vaccinated before all at-home workers (or, equivalently, no at-home worker is vaccinated before any in-hospital worker). Your proof should start with a clear statement of what it is assuming, and should argue about the behavior of the algorithm by referring to specific line numbers.

3. (CattleSorting) Suppose you are given an arbitrarily ordered array of n cows; each cow is one of k different breeds. As a cattle rancher, you want to group the cows based upon their breeds.

   For example, given an array of cows with breeds [Longhorn, Hereford, Longhorn, Hereford, Angus] (here $k = 3$) potential solutions include: [Longhorn, Longhorn, Hereford, Hereford, Angus], or [Longhorn, Longhorn, Angus, Hereford, Hereford]. You would not be happy with [Longhorn, Angus, Longhorn, Hereford, Hereford].

   Normally, we would simply sort the cows – and this would be an easy problem to solve. However, cows are sensitive creatures and don't like being compared to each other. For example, Herefords shouldn't be told that Longhorns are superior, because this will make the Herefords sad. We can only say that two cows are the same breed, or that they are not. Therefore, we can only use equality comparison (not $\geq$, $>$, $\leq$, $<$, or hashing, which feature in most sorting algorithms).

   Give an English description (no code/pseudocode) of an $O(nk)$ algorithm for grouping the cows by breed using only equality for cow comparison. Argue why your runtime is $O(nk)$ (a formal proof isn't needed).