**CIS 55**
**Exercise 1, Due no later than Friday, 17-Sep-2021, 11:59pm Pacific Time**

Below is your exercise for unsorted arrays. For submitting your answers, you can either send me an updated Word document with your responses or directly in an email. You can also do the work on a paper and take a screen shot of your results and email to me if you like. In all cases, email me your answers when you feel ready.

True or False
For each statement below, indicate whether you think it is True or False

1) If you have an array with no "holes", the append function performs at O(1)
2) Inserting an element at the front and shifting all elements performs significantly worse than appending an element at the end of the list
3) For the insert function, if the array is empty, there are no comparison operations that need to be performed and you can immediately add the new element
4) Binary search can be used on an unsorted array to significantly improve its performance from O(n) to O(1)
5) Because the update algorithm depends on using linear search, its performance is O(1) in the worst case scenario
6) If you search for and delete an element in an unsorted array and then shift the rest of the elements to fill the hole, the worst case performance is O(n)
7) If you search for and delete an element in an unsorted array and then move the last element to fill the hole, the worst case performance is O(n)

Short answer (5 points): The insert, update, delete, and search functions discussed in the slides are designed such that there are no holes assumed to be in the unsorted array. Your friend thinks that they can be coded to be equally as good in the worst case situations even if there are holes while still only using just the array and a "currentSize" variable. Is he right?

Address this by doing/answering the following:

1) Describe how the search algorithm changes if there are holes in the array. What is its Big O() and is it significantly worse, better, or the same as the no hole version?
2) Describe how the update algorithm changes (if any) if there are holes in the array. What is its Big O() and is it significantly worse, better, or the same as the no hole version?
3) Describe how the delete algorithm changes since it no longer needs to "fill the hole". What is its Big O() and is it significantly worse, better, or the same as the no hole version?
4) Now describe an insert algorithm that has to account for possible holes in the array. What is its Big O() and is it significantly worse, better, or the same as the no hole version?
5) Summarize your conclusion about your friend's assertion in a couple of sentences or a short paragraph

For your algorithm solutions, you may use pseudocode or actual code to describe your work. If you choose to write any code, be sure to also include a description of the steps (i.e. do not **only** submit code for your answers)