Computer Science 134C – Fall 2018 **Anonymous ID:**

Duane A. Bailey

Homework 8 – *Due: Monday, November 12*

Please answer the following questions for Monday.
We'll consider the performance of the searching and sorting routines. (There are many others, of course, but these are the most important approaches that you are likely to encounter.)

1. Exam and interview questions frequently involve understanding the performance of a particular sorting algorithm in a particular situation. Assume that we're sorting lists containing integer values with a particular starting order. What is the performance (please use big-O notation; e.g $O(n)$) of each sort in each situation? (Assume bubblesort stops when no swap occurs, and that quicksort always uses the leftmost value as a pivot.)

Initial Order	Sort			
	Bubble	Selection	Insertion	Quicksort
In sorted order				
Mostly in sorted order, then one random value				
Randomly out of order				
In reverse sorted order				

2. Let's look at the action of insertion sort. In the idInitial Setup row of the table below, write the seven digits of your Williams ID. In the six rows that follow, document the state of the list after you've completed each pass of insertion sort. After each pass, draw a circle around the digit just inserted.

Pass	`d[0]`	`d[1]`	`d[2]`	`d[3]`	`d[4]`	`d[5]`	`d[6]`
Initial Setup							
Pass 1							
Pass 2							
Pass 3							
Pass 4							
Pass 5							
Pass 6							

3. Write a method that will detect when a list is already in order. When `key` is provided, it determines the values to be used, as with `sorted`.

```
def inorder(l,key=None):
```

4. As we have learned, `quicksort` can run poorly on data with order, but that can be amelierated by selecting a random pivot value in the low/high partitioning function. Explain how a single call to a `shuffle` method might be used, instead.

⋆