Lab 6: Basic Comparison Sorts

A comparison sort is a type of sorting algorithm that compares elements in a list (array, file, etc) using a comparison operation that determines which of two elements should occur first in the final sorted list. The operator is almost always a **total order**:

1. a ≤ a for all a in the set

2. if a ≤ b and b ≤ c then a ≤ c (transitivity)

3. if a ≤ b and b ≤ a then a = b

4. for all a and b, a ≤ b or b ≤ a // any two items can be compared (makes it a total order)

In situations where three does not strictly hold then, it is possible that a and b are in some way different and both a ≤ b and b ≤ a; in this case either may come first in the sorted list. In a **stable sort**, the input order determines the sorted order in this case.

The following link shows visualization of some common sorting algorithms:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Your goal for this lab is to implement simple versions of Insertion Sort - **insertion_sort(alist)**, and Selection Sort - **selection_sort(alist),** that will sort a list of items and **count the number of comparisons**. Each function takes as input a list, sorts the list counting the comparisons at the same time, **and returns the number of comparisons**. After the function completes, the "**alist**" should be sorted.

The worst-case runtime complexity is $\Theta(n^2)$ for selection and insertion sort. Why? Write out the summation that represents the number of comparisons (we did this in class).

Note: There is a fundamental limit on how fast (on average) a comparison sort can be, namely $\Omega(n*\log n)$.

Fill out and **submit a paper copy to your instructor**, the table below as well as answers to the questions below.

- When counting comparisons, only count the number of comparisons between elements in the list. Don't count the comparison of the index of the list (the j > 0 part).
- Don't worry about your actual numbers, or which sort turns out to be faster in this experiment.
- Use the largest observed numbers for computing your estimated times.

Lab 6

Selection Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)		
2,000 (observed)		
4,000 (observed)		
8,000 (observed)		
16,000 (observed)		
32,000 (observed)		
100,000 (estimated)		
500,000 (estimated)		
1,000,000 (estimated)		
10,000,000 (estimated)		

Insertion Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)		
2,000 (observed)		
4,000 (observed)		
8,000 (observed)		
16,000 (observed)		
32,000 (observed)		
100,000 (estimated)		
500,000 (estimated)		
1,000,000 (estimated)		
10,000,000 (estimated)		

1. Which sort do you think is better? Why?

2. Which sort is better when sorting a list that is already sorted (or mostly sorted)? Why?

3. You probably found that insertion sort had about half as many comparisons as selection sort. Why? Why are the times for insertion sort **not** half what they are for selection sort? (For part of the answer, think about what insertion sort has to do more of compared to selection sort.)