Analysis of Algorithms (CSCI 323)
Summer 2021
Programming Assignment #2
Due in class - Monday, July 19

(Note: Assignment #2 will likely be assigned before July 19)

Instructions:

- Programs should generally be written in C, C++, Java, or Python. You may use any built-in functions provided by the language or its associated libraries.
- Programs should be submitted before the start of class on the date due through an online Google form that will be provided (similar to lecture notes), but will generally be graded through a peer-review system
- You are permitted to work individually, or in small groups of 2-4 students. However, even if you work in a group, you must have your own copy of the code that runs on your own computer that you can demonstrate as needed.
- At the top of each program source code, write the following
  - Course name and number (Analysis of Algorithms - CSCI 323 or 700)
  - Assignment Number (Assignment 1)
  - Your first and last name
  - If applicable, any students with which you collaborated (allowed, but must be acknowledged)
  - If applicable, any websites/texts from which you got code (allowed, but must be acknowledged)

Problem:

The goal of this assignment is to empirically evaluate Quicksort with two different variations:

(1) Use of a randomized pivot - index randomly chosen between positions low and high - compared to using a fixed pivot such as position low or high.
(2) Switching to Insertion Sort for small sub-arrays, i.e. when high - low < threshold

To empirically measure the performance of Quicksort, you should track both:
1. The clock time taken by the algorithm (subtract clock time at start from clock time at end)
2. The number of comparison operations performed (use a global variable *ncomp* for number of comparisons, and increment it each time you do a comparison)

Run Quicksort for a variety of sizes (say n = 100, 1000, 10000), for several trials of random data for each n, and track both the number of comparisons and clock time for each trial.

Assuming that the switch to Insertion Sort for smaller sizes improves performance, run additional trials to determine the optimal threshold, the point at which the algorithm switches from Quicksort to Insertion Sort.

Document all this in appropriate tables that clearly show the comparisons and clock-times for the various runs of your algorithm. Let the algorithm do the work and build the tables for you. (For Python users, Tabulate and Tabletext are two libraries that can help you achieve professional looking tables that document your effort and results.) You can also write results to a file if you cannot run them all at once.

As a starting point, you may use any publicly available code - in the language of your choice - for Quicksort and Insertion Sort, such as:

https://www.geeksforgeeks.org/quick-sort/
https://www.geeksforgeeks.org/insertion-sort/
https://rosettacode.org/wiki/Sorting_algorithms/Quicksort
https://rosettacode.org/wiki/Sorting_algorithms/Insertion_sort