Homework 4

CSC 140 – Advanced Algorithm Design and Analysis

Follow the "homework procedures" found on Piazza to submit files to DBInbox. If anything in this assignment does not make sense, please ask for help.

Quiz: We will have a closed-note 20-30 minute quiz on this homework in class Mon Mar 11.

Non-submitted work:

Read: CLRS Sections 5.2 and 7.1–7.4.

Written Problems:

There are three steps to follow for the written problems. Step 1: Do them as if they were homework assigned to be graded (ie, take them seriously and try to do a good job). Step 2: Self-assess your work by comparing your solutions to the solutions provided by me. Step 3: Revise your original attempts as little as possible to make them correct. Submit both your original attempt and your revision as separate files.

Submit two files to DBInbox following the procedure documented on Piazza. The first file should be named exactly **hw4.pdf** and should be your homework solutions before looking at my solutions. The second file should be named exactly **hw4revised.pdf** and should be your homework solutions after looking at my solutions and revising your answers.

NOTE: If you wish to handwrite your solutions you may, but only if your handwriting is very *easy to read and the file you submit is less than 1MB in size. Also, part of your homework grade may be based on my subjective opinion of how seriously you take this process.*

1) Do CLRS Exercises 5.2-4, 5.2-5, 7.1-1.

2) Quicksort is known to be a fast sorting algorithm for typical arrays. It is however also known to be pretty slow on small arrays. Some suggest to use quicksort recursively until sub-problem sizes get to a particular threshold, then stop using quicksort and instead use a sort that is more efficient on small arrays. For example, although insertion sort is $O(n^2)$, it performs exceptionally well on nearly-sorted arrays. (Click the "Nearly Sorted" button at `http://www.sorting-algorithms.com` to see just how good it is.) Let's test this hypothesis.

Let's define QuickInsertSort1 as normal quicksort (as defined in Section 7.1 of our textbook) but if $A[p..r]$ is k elements or fewer, it is sorted using insertion sort (as defined in Section 2.1 of our textbook).

Let's define QuickInsertSort2 as normal quicksort but if $A[p..r]$ is k elements or fewer, it is left unsorted. This will leave the array as a sorted sequence of unsorted blocks, each of up to length k. To complete QuickInsertSort2, use insertion sort on the entirety of A.

Code quicksort, QuickInsertSort1 and QuickInsertSort2 and time each for a wide variety of datalengths and k values. Your goal is to determine whether it is worth using insertion sort in either of these two ways and under what circumstance.

Write an analysis including about one page of text and also including graphic visualizations. Your first paragraph should summarize your results and subsequent paragraphs should expand on them. Imagine this is work done for your boss. It should be concise (bosses don't have a lot of time to spare), thorough (answering most questions boss might have), and well-written (it is your boss after all).

Timing a sort is tricky. You cannot repeatedly sort the same array multiple times because after you sort it the first time it becomes a sorted array with different performance characteristics. The way I will do my timing is following this pseudocode:

```
fill A with random integers
t = clock()
for j = 1 to 1000
 copy A to B
overhead = clock() - t
t = clock()
for j = 1 to 1000
 copy A to B
 sort(B)
t = clock() - t - overhead
```

Programming:

Due: The program(s) may be first graded sometime – maybe hours, maybe days – after the quiz.

A) Submit to DBInbox two files. One should be named **hw4_A.c** or **Hw4_A.java** and should implement the three versions of quicksort described above. The second should be named **hw4_main.c** or **Hw4_main.java** and should have the main program used for timing one or more of the three quicksorts. (I imagine your main loops over different sizes and outputs the timing for each, but it would be reasonable to do this for just one of the three quicksorts per run of your program, or to do it for all three at once.)

It should be possible to compile C programs with `gcc -Wextra -Wall -std=c99 hw4_A.c hw4_main.c` or Java programs with `javac Hw4_main.java` and the command should generate an executable without displaying any warnings. Running the executable should output data that you used for your graph in Problem 2.

All submitted code should include a comment at the beginning with your name, 4-digit code, and brief explanation of the what's in the file. Further, any hard-to-understand code should have local comments to help the reader understand.