CS 323 / 723

Analysis of Algorithms
Programming Assignment #1

1. **Comparisons of Clock Time for Sorting Algorithms (large arrays):**
   a) generate an array of 10,000 random integers, with values between 1 and 1,000,000.
   b) "Sort" the array using:
      HeapSort
      Modified Heap Sort (see note at the end of this document)
      QuickSort (basic version of Partition)
      QuickSort (using one of the "protections" we discussed in class

      making sure to use the **original random** array for each algorithm.
   c) Insert a timer into each of the sorting algorithms to measure the exact clock time to run each algorithm, and keep an "accumulator" for each algorithm. (C++ and Java both have functions for time)
   d) Include *counters* for the number of comparisons and the number of "swaps" ("bubbles") used in each algorithm, calculate average values and discuss the results.

2.RUN the sorting algorithms a total of 100 times each, and display the **average clock time** used by each of the algorithms.
a) Answer the following questions:
How do the results of the clock time compare to the expected results based on the known runtimes for these algorithm?

b) Did you encounter any problems when running the algorithms?

c) Run the same programs, but increase the array size to 100,000. How did the results compare to the original arrays of size 10,000? How does this compare to the expected results?

*For one "run" only, run QuickSort - BASIC and HeapSort on a **sorted array.** Did you encounter any problems? How do these compare to the average times from part (d)*

2. **Comparisons of Clock Time for Sorting Algorithms (small arrays)**:

a) generate an arrays of random integers, with values between 1 and 1000, with sizes 16, 32, 64, 128, and 256.

b) "Sort" the arrays using:

InsertionSort
HeapSort
QuickSort (basic version of Partition)
making sure to use the **original random** array for each algorithm.

c) insert a timer into each of the sorting algorithms to measure the exact clock time (ms) to run each algorithm, and keep an "accumulator" for each algorithm.

d) RUN the sorting algorithms a total of 100 times each, and display the **average clock time** used by each of the algorithms. (You will need to use ns to see the differences!)

At what array size is there a change in the "best" clock time for the algorithms? From InsertionSort to an O(n lg n) algorithm.

Modified HeapSort:

The regular *delete* method compares the node "trickling down" against its left and right child, and the swapping if necessary. A proposed improvement is to compare the node to its FOUR GRANDCHILDREN, potentially moving two levels. If the node does not swap with a grandchild, then compare with left and right children. Compare the performance against the standard HeapSort and QuickSort.