CS 1723, Data Structures
Fall Semester, 1998
Programming Assignment 9

Sorting Algorithms
Due November 30, 1998

The Assignment: For this assignment, you are to write program(s) that will measure execution (CPU) times of various sort algorithms. You can measure these times using the `clock()` function as shown in the handout that timed the C builtin sort `qsort()`. (Include `<time.h>` for `clock()`. You should call `clock()` just before sorting and just after sorting, so that you don't include the time to generate the random numbers or the time to check that the array is sorted. The difference divided by 1000000 is the CPU time in seconds. Give your values at least in 1/100s of a second. You can also measure times by some other method. Preferred is to do the measurement on runner, but failing that you can use any machine.)

Each sort algorithm should:

1. Generate `M` random real numbers and store them in an array `double a[M]`. (Careful! The array must be an array of `doubles`, not `ints`. If you used an array of `ints` and the random number generator `drand48()`, then all numbers would come out 0. In this case you might have an erroneous sort function that seemed to work OK.)

2. By some method, sort the `M` numbers into increasing order. Note that at the end of this step the numbers would usually still be stored in `a`, but not necessarily.

3. If they are not already there, store the numbers in the array `a`.

4. Run the `check_sorted()` function on the array `a` that will check that the numbers are indeed sorted into increasing order.

Choice of `M`: For high-performance sorts, `M` should be at least 100000. For the low-performance sort, choose `M` so that the sort will complete in a reasonable time (a minute or an hour).

Choice of sort: You are to do one low-performance sort, perhaps insertion or selection sort. Then you are to do each of the various sorts we are studying: heapsort (as discussed in class and in the handout, not from the text) tree sort (can be adapted from the white book, or class notes), and quicksort (presented in class; also in the white book). Thus you should do four (4) sorts altogether, one low-tech sort, and three high- performance sorts. Turn a listing that gives timing results of the sorts. Everything can be combined into one program, with a printed table that identifies the sorts and gives their times.