1 Experimental Assignment: Due March 22

Compare the performance of the following sorting algorithms in practice. Use array-based data structures only:

1. Quicksort (Use insertion sort for appropriate size small input, which needs to be determined experimentally, by finding the cut off point between pure insertion sort and pure quicksort, and try randomized pivoting and median-of-three pivoting for possible improvements)

2. Radix sort (Use counting sort as the stable sort subroutine, parameterize the radix sort using the base as an input parameter, and find the best base to use for fastest sort on your machine.)

Your report should include

1. A source code listing for each algorithm. The code must be adequately documented and formatted. (tear off the computer sheet into pages)

   Email your source code to GTA at navin_vish@yahoo.com. When sending any email to me, ensure that your NAME variable is properly redefined with your name. Use a meaningful subject header.

2. A brief description about the computer system you employed (if a PC, then state clock rate, and CPU’s name), language used (C recommended), and the random number generator you used. Briefly state your optimizations for quicksort and for radix sort.

3. Plots to submit:
   
   (a) Plot to find cut off point of insertion sort and quicksort.
   (b) Plot to find the best quicksort among its all variations.
   (c) Plot to find the best base to be employed for Radix sort (typical bases are high such as 130).
   (d) Plots to compare best quicksort, insertion sort, and radix sort: Two plots with average execution time on y-axis and input length, n, on x-axis. Use the same plot for all the algorithms so you can make a comparison. Simply connect the data points for an algorithm without performing any curve fitting, and label the curve with the algorithm’s name. Use a plotting program, or simply plot by hand. The first plot has range of input lengths up to 256 to compare the algorithms on small inputs. The second plot has range of input from 64 through $2^{14}$ (or more) using log scale to compare the algorithms over a large range of input lengths.

   The length of the input, n, should be chosen as $2^1, 2^2, \ldots, 2^{14}$ (or, larger). The integers to be sorted should be in the range 0..20000 (or larger). For each n, you should run an algorithm on 20 different randomly generated inputs and plot the average execution time. Ensure that all the algorithms are tested on the same set of data.

1.1 How to time a program?

To accurately measure small fraction of time, put the block to be timed into a loop, time the loop, and then calculate the block’s time by dividing the total loop time by the loop count. To ensure that the block executes on the same data set each time the loop is executed, some reinitialization may be needed. Keep the reinitialization code to a minimum. Since the reinitialization would be the same for all algorithms, it will still allow for a fair comparison.

Program Skeleton: For this assignment, the framework on the next page may be used for timing all the algorithms on the same set of inputs. The framework, however, assumes just two algorithms.

For a sample C program, see the Student’s Guide on C and Unix.
for n = 2^1, ..., 2^14

    Ta = Tb = 0 /* initialize all times to zero */
    for NoOfInputs = 1 to 20
        generate a random list of size n in array X

    /* Figure out how many times to repeat Algorithm */
    loops = Maxloops divided by a function of input size

    /* now run Algorithm A on X several times */
    t = time()
    for loopcount = 1 to loops
        /* repeat an algorithm loops number of times
           to ensure that time can be measured precisely */
        copy X to Y
        process Y using Algorithm A
    endfor
    Ta = Ta + (time() - t)

    /* now run Algorithm B on X several times */
    t = time()
    for loopcount = 1 to loops
        /* repeat an algorithm loops number of times
           to ensure that time can be measured precisely */
        copy X to Y
        process Y using Algorithm B
    endfor
    Tb = Tb + (time() - t)

    endfor /* for each of the 20 different inputs */

    Ta = Ta / (20*loops)
    output: "Average time taken by Algorithm A on an input of size n is Ta"

    Tb = Tb / (20*loops)
    output: "Average time taken by Algorithm B on an input of size n is Tb"

endfor /* for different input size */