HW 3: Experimental Assignment - Sorting

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 (10 points), which needs to be determined experimentally, by finding the cutoff point between pure insertion sort and pure quicksort (5 points), and try randomized pivoting (5 points) and median-of-three pivoting (5 points) for possible improvements)
2. Radix sort (Use counting sort as the stable sort subroutine (10 points), parameterize the radix sort using the base as an input parameter (10 points), and find the best base to use for fastest sort on your machine.

Your report should include

1. A source code for each algorithm. The code must be adequately documented and formatted.
2. Report:
    1. A brief description about the computer system you employed (if a PC, then state clock rate, and CPU's name), language used (C/C++ recommended), and the random number generator you used (2 points).
    2. Briefly state your optimizations for quicksort and for radix sort (3 points).
3. Plots to submit:
    1. Plot to find cut off point of insertion sort and quicksort (5 points).
    2. Plot to find the best quicksort among its all variations (5 points).
    3. Plot to find the best base to be employed for Radix sort (typical bases are high such as 100 or more) (5 points).
    4. 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.

        a) The first plot has range of input lengths up to 256 to compare the algorithms on small inputs (10 points).
        b) The second plot has range of input from 64 through 2^{14} (or more, in powers of 2's) using $\log_2$ scale to compare the algorithms over a large range of input lengths (10 points).

The integers to be sorted should be in the range 0..20,000 (or larger). For each *n*, you should run an algorithm on 10 to 20 different randomly generated inputs and plot the average execution time. Ensure that all the algorithms are tested on the same set of data.

```
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 */
```

How to time a program?

To accurately time a small fraction of time, put the block to be timed in a loop, time the loop and take average.

The given framework may be helpful.