Data Structures and Algorithms Spring 2019
Assignment 5

Due: 11:59 PM on 4/22/2019 in Canvas
(100 points)

PROGRAM DESCRIPTION: In this assignment, you will (1) write a complete C++ program to implement an efficient variable-base radix sort for integers and (2) perform an analysis with respect to the theoretical and experimental running time. We are not providing any code template, but you must follow the input and output formats specified below. Read carefully what you are required to do to receive full credit.

PROGRAM REQUIREMENTS:

- Your program should accept one command-line argument for the supported base (i.e., 2 for binary, 8 for octal, 10 for decimal, and 16 for hexadecimal) that will be used in the radix sort. If the user does not enter a base as the command-line argument, display out a meaningful usage statement and terminate the program.
- Implement as a function a variable-base radix sort (e.g., where the base is passed into a function as a parameter). That is, when the value 10 is passed to the function, the radix sort will use base 10 in the sorting algorithm.
- Randomly generate integral sequences between 0 – 9999, inclusively, and populate a data structure, such as an array, of size n = 10, 100, 1000, and 10000. Be sure to seed your random number.
- For each data structure size, run the radix sort 10 times to take measurements of the time needed for sorting in nanoseconds, excluding the time to generate the random numbers, display the sorting time for each pass, and compute the average sorting time. One possible way to measure the time is here.
- To show the proper working of the radix sort, display the unsorted and then sorted integers for the first pass of ten when the data structure size is the smallest (i.e., 10). Do not include the time to display in the time measurements.
- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.

- Please pay attention to the SAMPLE OUTPUT for how this program is expected to work. If you have any questions about this, please contact your instructor or TA assigned to this course to ensure you understand these directions.

ANALYSIS REQUIREMENTS:

The total running time for a radix sort is $O(nk)$, where n is the number of integers and k is the number of digits for the largest integer. If bucket sort is used as the intermediate sorting algorithm, then the more precise running time for a radix sort is $O\big((n+r)k\big)$, where r is the base. You will measure the impact of changing the base from 2 (i.e., binary) to 16 (i.e., hexadecimal), including base 8 (i.e., octal) and base 10 (i.e., decimal), inclusively, using randomly generated integer sequences of size n = 10, 100, 1000, and 10000. Run every test 10 times and average the results. Be sure only to include the time needed for sorting in nanoseconds, excluding the time to generate the random numbers. Plot the results (input size and running time) for radix sort for the different bases on the same figure, where the input size n is used for the x-axis and the running time (nanoseconds) is used for the y-axis. Since the values are large (in nanoseconds, for example), you may plot using the logarithmic values as needed.

SUBMISSION:

- This is an individual programming assignment that must be the sole work of the individual student.
- You should submit your source code, a README file where you explain all the information required for the grader to grade your program (i.e., how to compile, such as libraries or compile options needed, how to run it, an example, how you did it, etc.), and a document with your results, which include the raw data, a plot of the data, and a one-or-two paragraph analysis of what you've found in your experiment. Be sure to include your name in each submitted file.
- You will electronically submit your source code file(s) and other needed documents in Canvas by the due date.

SAMPLE OUTPUT (user input shown in **bold green**):

```
$ ./radixsort 10
Radix Sort: base = 10 size = 10
Unsorted: 3209 1253 7375 8443 3070  502 1153 2281 2671 7990
Sorted  :  502 1153 1253 2281 2671 3070 3209 7375 7990 8443
Pass  1:    66146   nanoseconds.
```

```
Pass  2:     48026   nanoseconds.
Pass  3:     48427   nanoseconds.
Pass  4:     48017   nanoseconds.
Pass  5:     48193   nanoseconds.
Pass  6:     46184   nanoseconds.
Pass  7:     46181   nanoseconds.
Pass  8:     46033   nanoseconds.
Pass  9:     45368   nanoseconds.
Pass 10:     43238   nanoseconds.
Average:     48581.3 nanoseconds.

Radix Sort: base = 10 size = 100
Pass  1:  120345.0   nanoseconds.
Pass  2:  128186.0   nanoseconds.
Pass  3:  115798.0   nanoseconds.
Pass  4:  113343.0   nanoseconds.
Pass  5:  114320.0   nanoseconds.
Pass  6:  113918.0   nanoseconds.
Pass  7:  113530.0   nanoseconds.
Pass  8:  112404.0   nanoseconds.
Pass  9:  111006.0   nanoseconds.
Pass 10:  121007.0   nanoseconds.
Average:    116385.7 nanoseconds.

Radix Sort: base = 10 size = 1000
Pass  1:  477781.0   nanoseconds.
Pass  2:  475458.0   nanoseconds.
Pass  3:  468680.0   nanoseconds.
Pass  4:  464173.0   nanoseconds.
Pass  5:  450881.0   nanoseconds.
Pass  6:  470459.0   nanoseconds.
Pass  7:  463053.0   nanoseconds.
Pass  8:  464888.0   nanoseconds.
Pass  9:  453539.0   nanoseconds.
Pass 10:  463982.0   nanoseconds.
Average:    465289.4 nanoseconds.

Radix Sort: base = 10 size = 10000
Pass  1: 4032139.0   nanoseconds.
Pass  2: 4000158.0   nanoseconds.
Pass  3: 3986871.0   nanoseconds.
Pass  4: 3936289.0   nanoseconds.
Pass  5: 3914532.0   nanoseconds.
Pass  6: 3946902.0   nanoseconds.
Pass  7: 3936055.0   nanoseconds.
Pass  8: 3957436.0   nanoseconds.
Pass  9: 3959956.0   nanoseconds.
Pass 10: 3873340.0   nanoseconds.
Average:   3954367.8 nanoseconds.
```