Assignment 1: Sorting
CS 758/858, Fall 2019
Source code due at **11:30pm on Mon, Sep 2**
Written work due **11:10am on Tue Sep 3**

Implementation

The skeleton code on the course web page is the start of a program to sort integers quickly. Complete the program by implementing counting sort and radix sort, and then measure its performance.

Your program will take three command line arguments: 1) the name of the sorting algorithm to use (`counting` or `radix`, plus the provided `insertion` and `system_quick`), 2) an input file name, or "-" to indicate that standard input should be used, and 3) an output file name, or "-" to indicate that standard output should be used. These things will have been taken care of for you in the skeleton code.

Your program will read a list of white-space delimited unsigned integers in plain text format from an input file. The first line of the file will contain a small header with two numbers: the number of values to sort and the maximum number of bits needed to represent a value — it will be ≤ 64. Your counting sort doesn't need to work with more than 16 bits, but its guts should be re-used for your radix sort.

Your program will write the numbers to an output file in sorted order, smallest to largest.

Testing

On the course web page and in the course account on `agate`, we supply some utility programs and input files. Most of the programs we distribute in this class will tell you their command-line arguments if you run them with the `--help` option.

`*.dat` files containing different numbers of random integers. These are available on `agate` in `~cs758/data/asn1/`.

`sort-harness` runs your program, validates its output, and optionally displays a plot of the performance. For example:

```
sort-harness -d -a radix -a counting ints-16bit-1mil.dat ints-16bit-10mil.dat
```

will run both radix and counting sort on the two datafiles `ints-16bit-1mil.dat` and `ints-16bit-10mil.dat`. By passing the `-d` option, the resulting performance will be plotted to the screen. If you are running the harness on a system without an X display then you may use the `-o <filename>` option to output the plot to a file instead (give the file a ".pdf",".ps" or ".png" extension). You can supply `--log` to the harness to put the plots in log scale which may make the plots easier to read.

Note: if you write to stderr, you might crash the harness. Also, if the harness crashes, please be sure to delete any temporary files that the harness left lying around, or else they will fill up your account quota.

`sort-ints-reference` is a reference sorting program that the harness uses to check your solution. You may also use `diff` to quickly check if your sorted output matches that of the reference solution.

We suggest that you implement one algorithm at a time (beginning with counting sort because the radix sort relies on it). After implementing each algorithm, you should compare it to the sorted output of the reference solution (possibly using the 'diff' program) and/or run it with the harness to make sure that it is correctly sorting and that it gives a reasonable performance curve. Begin testing with small instances before moving on to larger ones. For example:

```
sort-harness -a counting ints-16bit-1thou.dat
```

will run the counting sort on 1,000 16 bit random values and will compare the result to the result generated by the reference solution.

Finally, once all algorithms have been implemented, run them all an a variety of data sizes to compare the performance profiles. For example:

```
sort-harness -o 16bit-profiles.pdf -a counting -a radix -a
system_quick ints-16bit*.dat
```

will run your sorting program on all of the 16 bit data files with all algorithms. Each algorithm/datafile combo will be compared to the reference solution for correctness and finally a plot will be created and saved in the 16bit-profiles.pdf file.

Note: When testing the counting sort, do not use data more than 20 bits wide because you may run out of memory on agate.

Written Problems

1. Briefly summarize which parts of your program are working or not. Include transcripts or plots showing the successes or failures. Is there anything else that we should know when evaluating your implementation work?
2. What can you conclude about the performance of the various algorithm implementations from your experiments? (Include your plots when you turn in your write-up.)
3. Exercise 2.1–3 in CLRS.
4. Exercise 3.1–1 in CLRS
5. (858 only) Problem 3–1, parts a, b, and c.
6. (858 only) Problem 3–4, parts a, b, f, and g.
7. What suggestions do you have for improving this assignment in the future?

Submission

Please make sure that your code (as submitted) compiles on agate with the makefile that you supply. Please make sure that your code runs with the harness because this will be used to grade your assignment.

Electronically submit your source code using the instructions from the programming tips sheet on the course web page. Use the assignment name `1-undergrad` or `1-grad`, depending on whether you are enrolled in 858. Make sure that all of your source code and the makefile that compiles your code are in a directory. You will then submit this entire directory. Run the submission command one directory up from your code directory.

Hand in a listing of your source code (2 pages per page, as with `a2ps -2`) directly to the TA in class, along with your written work, which should contain plots showing the performance of your sorting algorithms.

Evaluation

In addition to correctness, your work will be evaluated on clarity and efficiency.

Tentative breakdown (10 points total):

3 counting sort (2 for 858)

3 radix sort

4 written problems (5 for 858)