CSE3081-2: Design and Analysis of Algorithms (Fall 2019)
Machine Problem 2: Master of Sorting

Handed out: October 15, Due: November 4, 11:59PM (KST)

1. Goal

The goal of this MP is to understand performance of different sorting algorithms and design your own efficient sorting algorithm.

2. Problem Description

You are given a list of integers that are not sorted. Your job is to sort the list in non-decreasing order. Simple!

You should implement four different sorting algorithms.

algorithm 1: choose one of the following: insertion sort, bubble sort, or selection sort algorithm 2: quick sort algorithm 3: choose one of the following: merge sort or heap sort algorithm 4: any algorithm or combination of algorithms you think will run the fastest

For algorithms 1-3, don't apply optimization techniques because they are for comparison purposes. For algorithm 4, do your best to optimize the algorithm. :)

3. Your task and requirements (Read Carefully!)

(1) You will write a C/C++ program which takes an input file and produces an output file. The input file has an unsorted list of integers, and your output file should be the sorted list. Your list should be non-decreasing: smaller numbers should appear first.

(2) Similar to mp1, your code should build and run on a **Linux machine**. Make sure you test on the machine before you submit the files.

(3) You should write a **Makefile** this time too. The TA will build your code by running 'make'. It should create the necessary binary file.

(4) Your binary file should be named **mp2_20180001**. The red part should be your student ID. There should be only a single binary file. It is up to you to make a single or multiple source code files.

(5) Your program should take two command-line arguments: The first one is the input file name, and the second one is the algorithm index. An example run is:

```
$ ./mp2_20180001 input00001.txt 2
```

(6) The input file contains a single line of numbers. An example input file is the following:

```
10 766020790 1182770779 1333893513 173226398 1071903604 1702255141
2121871803 2124051570 983886268 1364009855
```

The first number indicates the number of elements in the array. In the above example, there are 10 elements in the list. Note that the first '10' is not a part of the input list.

(7) Your program should produce an output file. The name of the output file must be "**result_algorithm_inputfilename**". In the above example where the input file is "input00001.txt", the corresponding output file should be named "**result_2_input00001.txt**". The output file should have five lines, containing the following items.

1st line: input file name
2nd line: algorithm index
3rd line: input size (the number of elements in the original list)
4th line: running time in seconds
5th line: the sorted list

The 5th line does not need to include the input size, because it is there in the 3rd line.

To measure running time, use the `clock()` function before and after performing the sorting. Your running time does not need to include time used for reading from a file and writing to a file.

In the above example, your output file should be as follows:

```
input00001.txt
2
10
0.000002
173226398 766020790 983886268 1071903604 1182770779 1333893513
1364009855 1702255141 2121871803 2124051570
```

(8) It is pretty obvious, but you must NOT use library functions that does the sorting for you. You should implement the sorting algorithms using basic data structures and primitive functions. Your performance should come from your algorithm, not from using optimized libraries or using special hardware such as GPU.

(9) Do not use parallel programming (multiple processes, multiple threads, CUDA, etc.). Implement a single-thread program.

(10) Do not turn on the optimize options when compiling your code.

4. Report

In addition to code files, you should also submit a report document.

First, you need to compare performance of the four algorithms. Describe how the running time of your algorithms grow as the input size grows.

You should start from a very small list (a few elements in the array), to a very large list (e.g. 2^{20} elements). The TA's test case will not exceed 2^{30} elements.

(IMPORTANT) Conduct two experiments on different types of lists. The first one is a random list, where the elements are totally random. The second one is a list in which the numbers are sorted in non-increasing order. In other words, the list is sorted in an opposite way.

For the random list, you should run algorithm on multiple lists and average their running time, because the running time could depend on the input list.

You can use tables, graphs, or any other visualization methods so that the reader could clearly understand the performance difference of the algorithms.

Second, you should describe how you designed algorithm 4. Describe what kind of techniques you have used, and why you think the technique will reduce the running time.

Your document does not need to be long, but you should definitely include the following:

- Experiment environment: The hardware specification of the machine where you did your experiments, such as CPU speed, memory size, and OS type and version. Obviously, you should do all your experiments in one machine for fair comparison.

- Experiment setup: This is basically how you chose parameters for your experiment. What is the metric that you measured, what is the range of input size, etc.

- Your comments on the experience: This is what you have observed from doing the experiments. (For example, you learned how the algorithms will perform when the input size becomes large. Can you see the trend for yourself? What if the input size is small?) Just write anything you observed, regardless of whether you can explain the phenomena or not.

5. Submission

You should submit the Makefile, the source code(s), and the report document. Make the file into a zip file named cse3081_mp2_**20180001**.zip. The red numbers should be your student ID. You can submit your work on the cyber campus.

6. Evaluation Criteria (Total 40 points)

(1) Correctness of your implementation: 12 points (30%)

- Does your implementation produce correct results for all four algorithms?
- Do you follow the requirements faithfully? (such as Makefile and output format)

(2) Performance of your algorithm 4: 12 points (30%)

If your algorithm produces wrong result, you will get 0 points here.
Otherwise, the points will be given based on the running time of your algorithm 4 (after comparing performance with other students.)

The input files will include multiple random files and may also include some (near) worst-case files.

(3) Comments and coding style: 4 points (10%)

- Is the code organized and easy to read? (indentation, spaces, styles)
- Do variable names reflect what they really are?
- Are there enough comments that explains what the code blocks are doing?

(4) Your report: 12 points. (30%)

- Can the reader understand the object and result of your experiments?
- Can the reader see the result visually?

7. Notes

- You should write your own code. You can discuss ideas with other students, but definitely should not copy their work. For this particular MP, since you can see the example code on the lecture slides, you will lean towards following the code exactly. My advice is that you just catch the idea, and then start writing your own code without looking at the example code.

- As announced in the first class, duplicates will receive zero grade.

- 10% of the evaluation goes to the practice of writing the code. Your code should not just work, but should also look good to other programmers. There are many books and materials on code writing practice. One example is a book called “Code Complete” by Steve McConnell.

- You may write your program in your own environment (Windows, Linux, MAC). But you should test your code on the Linux machine before submitting the file. Each of you will be given an account to the department Linux server.

- Remember that the TA will place your files in a directory, build your code using ‘make’, and run the code with the test inputs. Make sure everything works before submitting your work.

- Do NOT submit binary files. Submit only the files listed in the Submission section.