CS 456 : Advanced Algorithms

Programming Assignment #01

Total Points: 150

Assigned Date : Wednesday, January 24, 2018
Due Date : Wednesday, February 07, 2018 @ 02:59:59 p.m.

Overview

For this **programming assignment**, you will implement Quicksort and Mergesort. For each algorithm, you need to create an implementation to sort an array and one to sort a linked list. This means you will create a total of 4 programs: Quicksort on arrays, Quicksort on linked lists, Mergesort on arrays, and Mergesort on linked lists. Then, empirically validate the asympototic runtime behavior for best case, average case, and worst case of each using computer generated results. More specifically, you are expected to think about and address the following questions:

- At what size n_0 does your implementation start to exhibit asymptotic complexity?
- What is the characteristic of your input required to generate average complexity. How about best case and worst case scenarios? How do you plan to generate the appropriate input?
- How does the measured run time correspond to the abstract complexity analysis using operation counting (as discussed in class)?
- How to measure runtimes? What tools are available in the Linux environment that might help you?
- How to create the sorting class so that it will be extensible and reusable for future projects.

The programs must accept as command line input a filename/filepath to a file of integers, one per line. For output, it must create a file that is sorted. Examples of of both input and output file are at the end of this handout.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- You may use any programming language of your choice. However, you **must** make sure that your code compiles and runs on a typical Linux machine.
- Absolutely **DO NOT** include executables with your submissions.
- You **MUST** submit a makefile. If your program does not need to be compiled, then have the makefile output instructions on how to execute your program instead.
- It is **highly** recommended that you test your program's compilation and execution on the home server, **home.cs.siue.edu**, before submitting.

- The report part of your solution must be produced using a word processor. I highly recommend LaTeX. The report must be in **.PDF** format.
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications.
- Graphs/plots should be properly labeled.
- Your final The report must be in **.PDF** format. No exceptions.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.
- Include only the things necessary to run your program in your tarball. Absolutely do not include executables of any format with your submission.
- Include any additional instructions in a README file if needed.

Deliverables

The due date of this assignment is **Wednesday, February 07, 2018 @ 02:59:59 p.m.** A dropbox will be opened for submission on Moodle before the due date. Your submission should include:

- A **.tar.gz** tarball including your source code, a README file with compilation instructions, and a `makefile` to automate compilation.
  - Your tarball, when decompressed, should create a directory in your `last-name` with your source code in it.
  - Your `Makefile` should be in the base directory created.
  - The executables should end up in the same directory as the `Makefile`
- Your project report in **.pdf** format.

For example, submission to moodle: `lastname.tar.gz lastname.pdf`.

Report [120 points]

Your report should be structured as follows:

- Motivation and background of the experiment [**5 points**].
- For each algorithm, in their own sections, [**15 points**] per algorithm:
  - Pseudocode of the algorithm [**5 points**].
  - Correctness Proof [**10 points**].
- For each program, in their own sections, [**15 points**] per program:
  - Problems Encountered/Key insights [**5 points**].
  - Test results. This should include a graph that shows the execution runtimes of that particular program. You should use input sizes that are big enough to insure that the main factor in the runtime is the problem size, and also use enough inputs that the graph shows a clear pattern. [**5 points**]
  - Observations on and justification of results. If something seems odd, reason about it and offer an explanation about why it is the way it is. [**5 points**]
- Conclusion and performance comparisons. This includes comparing the different sorts against each other as well as how what difference the two data structures made. This section should also explain the graphs that were displayed in the test results section[**25 points**].

Compressed Tarball [30 points]

A compressed tarball of the directory containing your source codes, README and Makefile. Do not include executables in this tarball; we will do a fresh compile of your code using your Makefile. To create a compressed tarball of the directory `source`, use the following command: `tar -zcvf name.tar.gz source/`. Change name to your last name. The only files it should have is a Makefile and source code files.

- Program correct accepts command line input [**2.5 x 4 points**]
- Program produces correct output [**2.5 x 4 points**]
- Good coding practices e.g. naming conventions, readable code, commenting, etc. [**2.5 x 4 points**]

Extra Credit

Being able to utilize parallelization in program execution is an important ability, especially today when processing speedups are coming from the inclusion of multiple processing cores instead of simply faster processing cores.

For extra credit, create a pseudo code for each algorithm that shows how to successfully execute all or part of each algorithm in parallel. [**5x2 points**]

Execution Sample

quicksortarray 10.txt

Sample input file: `10.txt`

17
63
31
95
51
38
30
22
33
83

Sample output file: `quicksortarray-10.txt`

17
22
30
31
33
38
51
63
83
95