Project 1c – Experimental Analysis of searching and sorting algorithms on realistic data

# Project 1C is due on Oct 25. Note that it builds on Project 1a and 1b.

## General Notes

* For this project, by sorting we mean sorting in **ascending** order.
* Submit all source code, program output, and project report electronically on Canvas as a Zip file.
* Allowed programming languages: Python, Java, C++/C, or others *with prior approval from the instructor*.
* Implementations should be as optimized as possible.
* When measuring execution time, do not include the time for reading the input data from disk. Only include the time for the actual algorithm to execute!
* **You are not allowed to use online resources when writing source code solutions.**

## Academic Honesty Policy

For this project, you are not allowed to discuss with any other students currently in the course or with any person outside the course. We use a plagiarism detection tools on all submitted source code. It is capable of finding portions of copied code from other students in the course, other sections of the course, previous offerings of the course, as well as Internet sources.

## Instructions

In project 1a and 1b you implemented several sorting algorithms and then analyzed the performance of those algorithms. The shortcoming of project 1a and 1b was that they used mere *integers* as the input data whereas in a real-life application we need to sort more complicated items, e.g., books in an online library. In project 1c you address this shortcoming by considering book records (of an online library) as the input data, which can have multiple fields like ISBN, author’s name, book title, and more.

A screenshot of a data array

Description automatically generated

The following algorithms **must** be implemented:

1. Linear search (as described in the classroom)
2. Merge sort (as described in the textbook, Page 34) and Binary search (as described in the classroom)
3. [**CS** **5120 ONLY**] Find when Binary search becomes cheaper than Linear search.

The emphasis of the project is on experimental analysis of execution time when run on realistic data. You need to track the time and record them in units that are fine-grained enough to allow meaningful comparison. For example, the unit for time should be micro- or possibly even nano-seconds on fast machines.

Your algorithms will be tested for correctness, so please ensure they work! This includes edge cases like inputs with a single value, odd number of values, even number of values, etc.

## Expected Input Data Format

Each book has the following fields:

1. ISBN (simplified) number (a 6-digit integer)
2. Author’s name (Last name, first name).
3. Title of the book (just *one word* for simplicity).
4. Size of the book in bytes (a 6-digit integer)
5. A pointer to the starting location of the e-book (a 10-digit integer)

These five fields together make a *book record*. We can create a book by choosing *random* values for each field of the book record. No need to make the author’s name realistic and no need to make the title realistic; for instance, “abc, defg” can be an example “last name, first name” whereas “wxyz” can be an example “title of the book”. Create 5,000, 10,000, 20,000, 40,000, and 80,000 *book records* and store them in separate files. Thus, there are a total of 5 input data sets (files). Create these 5 sets of input data once and use the same data for each algorithm.

Files should be text files and contain one *book record* per line. Your algorithm implementations should expect and accept input in this format**. Note that to save RAM we will not reserve memory to hold the full ebook; we will just hold a pointer in the book record, which points to the memory chunk holding the ebook in reality.**

In Project 1a and 1b, we provided a sample Python program to generate integers. You may modify that program to generate the dataset for the current project. The generator program can be called as follows:

* python3 genBooks.py 20000
  + generates a random list of 20000 book records and stores in a file.

If you use a different style of generator program to generate the data sets, that is fine. Your generator must conform to the specified format above.

In any case, you need to submit the data generator program and the smallest data set (a text file with 5000 book records).

## Expected Algorithm Output Format

For each data set (of the five as mentioned above), populate an array with the book records. Then, produce the output as follows.

1. Run the Linear Search program, searching for a book record. Randomly select an ISBN number and search for the corresponding book record. Repeat this step (random selection and search) 5000 times.
2. Run the merge sort program to sort the input array. Run the Binary Search program, searching for a book record. Randomly select an ISBN number and search for the corresponding book record. Repeat this step (random selection and search) 5000 times.

The output of your Linear search or Merge sort or Binary search program should be two values on a **single line, comma separated**: size of the input, and the execution time (in nano/milliseconds). For example, the output might look like the following:

20000, 608379  
20000, 690038  
20000, 690287  
20000, 680785  
. . .

**What to Submit**

See the rubric file for details.

The general guideline is as follows.

You should submit an archive (ZIP, TAR/GZip, etc) of all files, including the following:

1. All source code for all algorithms. Make sure each algorithm can be executed as its own executable/main method and accepts a path to a file to use as input.
2. Makefile (or simple shell scripts) to compile the source code and execute the algorithms.
3. A README.txt file including instructions on how to compile and run your source code, any non-standard libraries used, etc.
4. Your input dataset generator program. The smallest data set (5000 book items) that you generated and used in your analysis and report.
5. Runs showing output (namely the running time) for each algorithm for each data set. This can be saved into either a text file (with file extension .csv) or an Excel worksheet.
6. A project report (Word or PDF) containing (at a minimum) the following:
   1. Expected/theoretical performance (e.g., theta notation) for each algorithm.
   2. A table summarizing the observed results of each algorithm.
   3. A graphical representation of the data (graphs).
   4. Analysis and Discussion of ***your results*** vis-à-vis the expected results.
      1. Are the theoretical and actual performance results consistent?
      2. Any anomalies?
      3. What surprised you?

In Item 7, make sure you tell a story and cover all points.