Programming assignment 2

Getting started

All your submissions should be implemented in Java. If you do not have a Java development environment, we recommend that you either use the Visual Studio Code Java extensions or Jetbrains IntelliJ. If you use IntelliJ, do not forget to register as a student, so you get access to the full versions. If you run Linux, macOS, or WSL, and want more control over your JDK, we recommend SDKMAN!

Note that you must solve all of the programming problems in Java. You are allowed to use Python (or any other suitable tool) to analyse the results, plot graphs, and so on.

Problems

Problem 1

Implement a randomized queue where each dequeue operation returns a random element from the queue. Your implementation should support the following operations: size, isEmpty, enqueue, dequeue. Note that dequeue should pick one of the inserted elements randomly, remove it and return it. You should also implement an iterator for your randomized queue. Note that the iterator should iterate over the elements in random order, returning each element of the list once. The random iterator must be able to iterate over all elements in linear time. You are allowed to use additional memory.

You should use an array to implement your randomized queue. The queue should grow when full and shrink when enough elements are removed. Discuss your design in the report.

Problem 2

Implement a binary search tree and an AVL tree. Devise a reasonable way to create realistic average case trees (e.g., combinations of inserts and deletes) and then compare the two trees in terms of operation cost/time, height, etc. It is also interesting to compare how they behave when in the worst and best cases.

Describe your experiment setup and your findings in the report. Does the balanced tree improve balance in most cases? At what cost in extra time? Is it worth it when the trees grows and the non-balanced is average or worst case?

Problem 3

You are developing a system for a company that sells tickets to various events. Since there is often more demand than available tickets, those who want to buy tickets must enter a queue. The company has various categories of customers, e.g., member programs that get better or worse access to tickets. These are represented using integers. So, when a person enters to buy a ticket, their name and an integer representing their category are stored. When it is time to release the tickets, the company picks N persons based on time in the queue and priority level. These are removed from the queue and assigned tickets. Since people can join (or leave) the various programs, it should be possible to swap the category of someone in the queue. Note that this should not affect the time in the queue.

Your implementation should at least contain the following methods:

- `insert_person(name, priority)`
- `get_person()` (returns the next person that can buy a ticket)
- `delete_max_prio()`
- `swap_priority(name1, name2)`

Implement the system and discuss your design in the report. Provide an analysis of the time complexity of the various methods and test how well it scales. It is common for high-profile events to have several hundred thousand people interested in tickets.

Problem 4

Implement Quicksort on Java arrays. Use median of three to determine the pivot value. Your implementation should take a parameter, `depth`, which determines at which recursion depth Quicksort should swap to Heap- or Insertsort. You need to implement Heap- and Insertsort.

Conduct an experiment to determine recommended for `depth` and whether Heap- or Insertsort should be used.

Problem 5 (optional, for higher grade)

Huffman coding is commonly used for lossless data compression. Write a program that reads a text file, computes the frequency for each character, and creates a Huffman tree. You do not need to produce the compressed output, but your tree should provide a method to get the Huffman code for a character.

Problem 6 (optional, for higher grade)

Implement Shellsort and experiement with different Gap sequences. Implement at least 2-3 differen seqences. Use Wikipedia's article as a starting point for various gap sequences. Compare how the various gap sequences affects how Shellsoft scales when input sizes increase.

Submission guidelines

Submit your solutions as a single zip-file via Moodle no later than 17:00 on December 15, 2022 (cutoff 08:00 December 18). This is an individual assignment. Your submission should contain well-structured and organized Java code for the problems with a README.txt (or .md) file that describes how to compile and run the Java programs and a report in PDF format that describes the findings from problems 1-4 (and 6 if you solve it).