CSE250, Spring 2013 **Assignment 10 (the last)** **Due 4/26 & 4/29**

Now with final project question and extra-credit options

The **Final Exam** is **Monday, May. 6**, **11:45am–2:45pm** in **Knox 20**. Unlike the prelims, it will be *open-book, open-notes*. It will cover the text chapters 1–11 up through the domain of this homework—in particular, the last two lectures and/with B-Trees are not covered. Supplementary posted lecture notes and code files *are* in the range of the final, which is *cumulative*—applying to the whole course.

Reading: For the last week, Chapter 11, skimming (but not skipping) AVL Trees. A copy of the Prelim II answer key is available at the non-public link

`http://www.cse.buffalo.edu/∼regan/cse250/CSE250x2key.txt`

The following three questions are *hardcopy only*, due in class on **Friday, April 26**. Sorry for the "extra" during the main project, but they serve as study help for the final, and total only 27 points.

Then follows a final specification of the timing experiment asked for in the Project 2 final client, and the final report question about it. It also spells out extra-credit opportunities.

(1) Consider open-address hashing with the standard quadratic-probing rule that if the initial slot k = `hashFun(item)` is occupied, the i-th retry is in slot $k + i^2$ (modulo the table size). Use the (admittedly poor) hash function that adds up the number values of all letters a = 1, b = 2, etc., as on Prelim II.

(a) For table-size 8, attempt to insert the words `bad bed bid dad fed gag` in that order. Does your table get `fed` up? Or does it gag on `gag`?

(b) Now change the table size to the prime number 7 and try again. (9 pts. total)

(2) Text, section 10.9, "Self-Check Exercise 1" on p614, which asks you to diagram how `quicksort` partitions, recurses on, and sorts the array `[55 50 10 40 80 90 60 100 70 80 20]`. (9 pts.)

(3) Text, section 11.3 of Chapter 11, "Self-Check Exercise 1" on p656. This refers to a particular red-black tree built from the phrase "the quick brown fox jumps over the lazy dog," calling on you to show the changes (and node colors) after further inserting the words "apple," then "cat," then "hat." (Note: this will not be lectured on until Monday. 9 pts., for 27 points total on this last hardcopy set.)

Project 2 Timing Code and Final Report Question

Once you have the Project 2 client working, specifically on the file `allstocks100k-30.data`, use the `HiResTimer.h` file to carry out the following timing runs on `timberlake`. This timer class uses special system files that may not exist on your home system. Besides the files `stringsorts.cpp` and `templatesorts.cpp` shown in lecture from the `Java2C++` directory, the file `Josephus.cpp` in the `/.../PROJECTS/BASE/` directory has an example of using the timer to carry out a run. Compile your code on `timberlake` with `g++ -O5` for highest optimization, and don't forget the needed extra `-lrt` at the end to load the library for real time. Please do the following experiments:

(a) Do 10 runs on `allstocks100k-30.data` with your ChainedHash and your heap class. Average the times and record the result.

(b) Change your project client to use your-choice-of `Valli` or `std::set` in place of the hash table. (This should not require any change to the heap or other classes in the project, and may require changing just a few declarations in your joint client file.) Again do 10 runs, take the average, and compare the results to (a).

Report Question (2), joint answer in the comment below `main`, after your individual-credit answers to questions (1A,1B) about your data-structure classes: State and compare the running times you obtained. Say also whether there was a lot of variation in your answers, which could be an effect of overall system load. Draw a general conclusion about whether you were able to tell a difference between the efficiency of hashing versus using a sorted data structure.

Final Breakdown of 300 Project Points

- 200 individual-credit points for data-structure and client-class components, as specified before (except that the "momentum" comparison is now extra-credit, see below).
- 21 individual "checkpoint points" for completing a debuggable draft of the above by the Monday 4/22 checkpoint deadline, so that we can inspect and test them before labs begin that week. They do not have to run perfectly, but must have everything the design requires (nominally `rehash` in the hash table can wait, but it's good to try to debug it in the context of the Assignment 8 driver by which to test your code).
- 30 points each for the joint client file, not due until 4/29.
- 24 individual points for report question (1A or 1B) as specified before.
- 24 points each for modifying the client to do the timing runs and use an alternate data structure via the same standard-library interface, with a joint answer also for report question (2).
- 1 pt. for not forgetting to put your name in a comment of every file that you were responsible for, including both names in the client file.

Extra Credit Options

These are expected to be joint efforts, leaving it to you to split work fairly. If you each do one, points can count for both of you—the points are chosen with that in mind, Each has a corresponding report-question requirement, so they are numbered beginning with (3).

(3) Implement the `printTopByMomentum?` query, for 12 points extra (each): say in the report section how much the top-30 list you get differs from the top 30 price (percentage) gainer lists.

(4) Implement `printTopByTrendShares?` *without taking time proportional to the number of past trades in the stock*. You must write in your report answer how you implemented it to run in $O(1)$ time, for 30 pts. extra (each).

(5) Modify your client to use a sorted data structure (`Valli` or `BST` or `std::multiset`) to generate the top-k lists, in place of the heap. Time *just the final top-k section* of `allstocks-100k-30.data` 10 times each with your heap and again with your choice of sorted data structure. Write a short report answer similar to (2). Note that this will require more-extensive changes to your client code than the required-credit test in (2), mainly because `make_heap` is being replaced by re-inserting everything and then iterating (perhaps backwards from the end) to list out the top k elements. Describe these changes in your report answer, and also why you did not make your sorted data structure behave like a set. (50 pts. extra, each)

Finally, and to confirm what I said in class, these two deviations from previously stated rules are permitted: (i) The hash function need not be passed in as a function object, but rather coded in the `ChainedHash` class and applied to a `string` argument. No mention of "`Stock---`" is allowed in the hash-table file, but you can write a `REQ` that the client object used for `I` provides `I.hashString()` or some other appropriate general name for a string-valued method, which the `Stock` class will choose to implement as a getter for the ticker symbol. (ii) Rather than have separate templates `I` and `COMP`, filled respectively by pointer/proxy-to-`Stock` and by your various comparison function objects to create the various heaps, you can combine them so that your `Heap` class has just one template parameter which is `REQ`uired to play both roles. The latter was exemplified in this past week's recitations.