CSC-361
Lab 8

A Comparison of Heap Implementations (Part II)

In this lab you will repeat all of the steps from Lab 7 with the array-based, complete tree implementation of a min-heap: `ClassicMinHeap`. That is, you will implement the linear time `build` operation (with a `sink`), an `insert` operation that calls a `swim` procedure, and `extractMin` that uses the `sink` procedure.

Your implementation must follow the same `ExpandableBaseHeap` described in the previous lab.

What You Need To Do: Timing in Main and Reporting Timing Operations

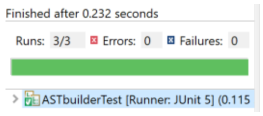
Just as with the last lab, you must time the `build` and `extractMin` operations using a randomly permuted list of values. Report the results of these timings in the header comment of the file. Again, we expect a table and efficiency in O-notation similar to what is shown below.

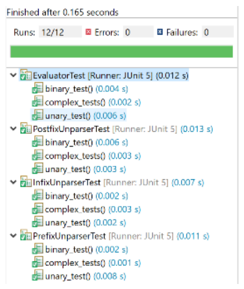
	Build	**ExtractMin**
5000	1	3
10000	2	6
50000	3	9
100000	4	12
200000	5	15
…		
	O(?)	O(?)

No code has been provided since we are building on the prior lab.

JUnit Testing

No output should be produced by your tests; we are seeking only a 'green' output indication in Eclipse. Make sure your screenshot shows success of all junit testing methods and not just the overall summary. For example, the image on the left is bad (and on the right is good).


Finished after 0.232 seconds
Runs: 3/3 Errors: 0 Failures: 0
ASTbuilderTest [Runner: JUnit 5] (0.115


Finished after 0.165 seconds
Runs: 12/12 Errors: 0 Failures: 0
EvaluatorTest [Runner: JUnit 5] (0.012 s)
binary_test() (0.004 s)
complex_tests() (0.002 s)
unary_test() (0.006 s)
PostfixUnparserTest [Runner: JUnit 5] (0.013 s)
binary_test() (0.006 s)
complex_tests() (0.003 s)
unary_test() (0.003 s)
InfixUnparserTest [Runner: JUnit 5] (0.007 s)
binary_test() (0.002 s)
complex_tests() (0.003 s)
unary_test() (0.002 s)
PrefixUnparserTest [Runner: JUnit 5] (0.011 s)
binary_test() (0.002 s)
complex_tests() (0.001 s)
unary_test() (0.008 s)

Submitting: Source Code

Your code should be well documented, including docstring comments of methods, blocks of code, and header comments in each file. Junit tests should have reasonable String messages output, if failure occurs.

Header Comments

Your program must use the following standard comment at the top of *each source code file*. Copy and paste this comment and modify it accordingly into these files.

```
/**
 * Write a succinct, meaningful description of the class here. You should avoid wordiness
 * and redundancy. If necessary, additional paragraphs should be preceded by <p>,
 * the html tag for a new paragraph.
 *
 * <p>Bugs: (a list of bugs and / or other problems)
 *
 * @author <your name>
 * @date   <date of completion>
 */
```

Inline Comments

Comment your code with a *reasonable amount of comments* throughout the program. Each method should have a comment that includes information about input, output, overall operation of the function, as well as any limitations that might raise exceptions; Javadoc comments are ideal. Each *block* of code (3-4 or more lines in sequence) in a function should be commented.

It is ***prohibited*** to use *long* comments to the right of lines of source code; attempt 80 character-wide text in source code files.

Submitting: Proof of Program Execution

Execute your code and take a screenshot of the associated junit window and output console (with no output). Place these screenshots into a word processing document (Word, OpenOffice, GoogleDocs, etc.). If multiple screenshots are necessary, label each clearly. Please make sure to crop and enlarge the screenshots so that the picture and / or text is clear (and doesn't strain my old eyes).

Create a PDF of this document and call it `evidence.pdf`.

Source Code Files

Please submit only `ClassicMinHeap`.

Final Submission File

In a folder named `lab`, place (1) the source code files and (2) `evidence.pdf`. Zip folder `lab` and label that zip file as `lab.zip`. This zip file is to be submitted via Moodle.

Please be reminded that following instructions explicitly and submitting well-formatted documents (with consistent fonts and typeset equations) is an important part of professionalism.

Only one person in your pair needs to submit the final product.