CSC 225 - SUMMER 2015
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 3
UNIVERSITY OF VICTORIA

Due: Tuesday, June 30th, 2015 at 4:00pm.

1 Programming Assignment

A Java template containing an incomplete heap implementation has been provided. Your assignment is to implement the following methods:

- `removeMin()` - Remove and return the minimum element of the heap.
- `add(element)` - Add the provided element to the heap.
- `getSize()` - Return the number of elements in the heap.
- `getRoot()` - Return the node corresponding to the root of the underlying tree.

The `removeMin` and `add` methods are expected to have a $\Theta(\log n)$ running time, and the `getSize` and `getRoot` methods are expected to have $\Theta(1)$ running times. Your implementation must permit any number of elements to be added to the heap. As a result, to achieve a $\Theta(\log n)$ worst case running time for `add`, it is not possible for your implementation to store the heap in an array (since resizing the array during an `add` operation requires $\Omega(n)$ time). Instead, your implementation should be strictly reference based.

You must use the provided template as the basis for your submission. Other than the four methods above, you are not permitted to change any other aspect of the provided code. You may add new classes and methods as necessary (but all code must be contained in one file). In particular, the `HeapNode` class, which is used to represent nodes of the heap, may not be changed in any way (including adding, removing or modifying members of the class). You are allowed to create a subclass of `HeapNode` if you want to extend or modify its functionality.

1.1 Testing and Verification

The testing code provided in `main()` reads a list of integers and inserts each value into the heap. After all values are inserted, the `removeMin` method is called repeatedly until the heap is empty, to produce a sorted ordering of the input list.

Besides code to test the `removeMin` and `add`, the template also contains functions to automatically verify the correctness of the constructed heap at each step. The testing code in `main()` will verify the heap properties after each insertion and removal. These tests are time consuming (verifying the properties is a $O(n)$ operation), but can be helpful for debugging. The comments in the template explain how to enable each test. Do not use the verification functions directly in your code (let the code in `main()` handle everything for you). The tests will all be disabled automatically before marking (you do not have to disable them yourself before handing in your code).

2 Test Datasets

Since the input format and testing protocol (that is, sorting an integer array) are the same as Assignment 2, no new test data has been posted for this assignment. You should use the posted data for Assignment 1 and Assignment 2, as well as inputs you create yourself, to test your implementation.

3 Evaluation Criteria

The programming assignment will be marked out of 50, based on a combination of automated testing (using large test arrays similar to the ones posted on conneX) and human inspection.

To permit effective testing of the correctness of the `add` and `removeMin` methods, your submission must have a functional `getRoot` method. If the `getRoot` method is incorrect, it will not be possible to properly test your `add` and `removeMin` methods, so you will receive at most 3/24 for each method. Similarly, it will not be possible to test `removeMin` without a correct `add` implementation, so submissions without a functional `add` method will receive at most 3/24 on `removeMin`. For this reason, you are strongly advised to complete the `add` method before starting on `removeMin`.

The 50 marks of this assignment will be divided among the tasks as follows:

`add` method [24 marks]

Score (/24)	Description
$0-3$	Submission does not compile, does not conform to the provided template or contains an incorrect implementation of `getRoot`.
$4-11$	The implemented algorithm is substantially inaccurate on the tested inputs.
$12-18$	The implemented algorithm uses an array to store the entire heap (even temporarily) or has a $\Theta(n)$ running time.
$19-24$	The implemented algorithm has a $\Theta(\log n)$ running time and is correct on all tested inputs.

`removeMin` method [24 marks]

Score (/24)	Description
$0-3$	Submission does not compile, does not conform to the provided template or contains an incorrect implementation of `getRoot` or `add`.
$4-11$	The implemented algorithm is substantially inaccurate on the tested inputs.
$12-16$	The implemented algorithm uses an array to store the entire heap (even temporarily) or has a $\Theta(n)$ running time.
$17-24$	The implemented algorithm has a $\Theta(\log n)$ running time and is correct on all tested inputs.

`getSize` and `getRoot` methods [1 mark each]

Score (/1)	Description
0	Submission does not compile, the method gives incorrect results on any input or the method has a $\omega(1)$ running time.
1	The method is correct on all tested inputs and has a $O(1)$ running time.

To be properly tested, every submission must compile correctly as submitted, and must be based on the provided template. **If your submission does not compile for any reason (even trivial mistakes like typos), or was not based on the template, it will receive at most 6 out of 50**. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.