CSC172 LAB 12

HEAPS

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_programming.

Every student must hand in their own work, but every student must list the name of their lab partner if any on all labs.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

The textbook should present examples of the code necessary to complete this lab. However, collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

2 The Heap Data Structure

The goal of this lab it to gain familiarity with the basic operations of the Heap data structure.

1. Begin this lab by defining a simple `MyHeap` interface as shown below.

```
public interface Heap {
     public void insert(Comparable item);
     public boolean isEmpty();
     public int size();
     public Comparable deleteMin();
}
```

2. Write your own class that implements the `Heap` interface. You will need to declare an array of `Comparable` objects (i.e. `Comparable[]`) to implement the heap as well as integers for the current size and the default capacity. Once you have done this you can quickly implement the `size()`, `isEmpty()` and constructor methods. You should have two constructor methods, one that takes in an integer for the default capacity of the heap, and a parameterless constructor that assigns a default capacity of 10. Write a small test class with a simple `main()` method to test these methods.
3. Implement the `insert()` method in your heap class. Of course, in order to do this you will have to write the private `bubbleUp()` method since bubbling up is part of insertion on any heap. Write an additional helper method `printHeap()` that prints your heap's contents to the console. Do not print null elements in the heap. In the main method of your test class, insert some `Integer` objects into your heap and then print the heap to display the contents.
4. One problem arises with the array implementation of any heap. If more items are inserted on the heap than the array can accommodate, we need to expand the size of the heap. Implement a method that expands the array of objects by copying existing objects into a new, larger array. Modify your insert method to test for potential array overflow and make a call to the expand method when necessary. Test your method by having your constructor start with a very small array and make sufficient insertions to require at least two expansions of the array.
5. Implement the `deleteMin()` method. In order to do this you will have to implement a `bubbleDown()` method. Modify your test program to demonstrate the workings of your `deleteMin()` method.
6. Sometimes we need to start with a random array and form it into a heap. It would be inefficient to do this by successive insertions. Implement a third constructor to your heap class that can take an array of comparable types and turn them into a heap by re-arranging the elements. Write a `heapify()` method that performs the operation by swapping array elements. Add code to your test program to fill an array with random `Integer` objects, create a heap using the new constructor, and finally print your heap using `printHeap()` to demonstrate your `heapify()` method.

3 Hand In

Hand in the source code from this lab at the appropriate location on the BlackBoard system at my.rochester.edu. You should hand in a single zip file (compressed archive) containing your source code, README, and OUTPUT files, as described below.

1. A plain text file named README that includes your contact information, your partner's name, a brief explanation of the lab (A one paragraph synopsis. Include information identifying what class and lab number your files represent.), and one sentence explaining the contents of all the other files you hand in.
2. Several Java source code files representing the work accomplished for this lab. All source code files should contain author and partner identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

4 Grading

90% Functionality and Testing

- 15% constructors (5% each)
- 5% size() method
- 5% isEmpty() method
- 20% insert() and bubbleUp() methods
- 20% delete() and bubbleDown() methods
- 10% printHeap() method
- 15% heapify() method

10% README and OUTPUT files