Lab 7 (Heaps)

CSC 172 (Data Structures and Algorithms)
Fall 2023
University of Rochester
Due Date: Sunday, November 5th 2023 @ end of day

Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Every student must hand in their own work but also list the name of their lab partner if any on all labs.

In this lab, you will work on the Heap Data Strucure to implement a Priority Queue ADT.

1. Start with the `UR_Heap` interface code below. Implement it in a class. HINT: There are obvious helper methods that can be used.

```
public interface UR_Heap<T extends Comparable<T>> {
       public void insert(T item);
       public boolean isEmpty();
       public int size();
       public T deleteMin();
}
```

2. Write your own class that implements the Heap interface. You will need to declare an array of `T` objects (i.e. `T[]`) 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 `Intege`r 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.

Before you start

Note:

- Be sure to include a Unit Test program that demonstrates how your program works.
- Please write any assumption made. For example: describe what the return value means.
- Your methods must handle all the corner cases gracefully — for example, throwing exceptions with detailed explanations or returning values indicating the error in case the operation is not permitted. The comments should clearly state the issues and the remedies involved. In short, no illegal operation should be permitted and the list and all its parameters should be in a valid state.

Submission

Submit a single zip file `Lab7.zip` containing the class you implement, a unit test file containing a main method, a README file, and your class files. Your README should describe how to run your unit tests. Upload this file at the appropriate location on the Blackboard system at `learn.rochester.edu`.

1. All source code files should contain author and partner identification in the comments at the top of the file.
2. All source code should include links to any internet resources you used outside of course materials.
3. 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 any other files you hand in. You can also share testcases those you ran.

Grading

Total: 100 pts

- 10 ptss for README file, proper commenting, and error handling.
- 5 pts size() method
- 5 pths isEmpty() method
- 10 pts printHeap() method
- 20 pts heapify() method
- 20 pts insert() and bubblup() methods
- 20 pts delete() and bubbledown() methods
- 10 pts for your main method containing your unit testing class (main method, tests, etc)