Lab 6: Heaps and heap sort

Week of December 3, 2018

Objective

To compare heap sort to insertion sort, quick sort, and hybrid quick sort.

Exercise

The file `Lab6.java` contains a copy of the solution to Assignment 1, which compares insertion sort, quick sort, and hybrid quick sort, to which we have added comparisons to heap sort and the beginnings of a `Heap` class. Your job is to complete the bodies of the methods in the `Heap` class.

Note that the file `Lab6.java` contains functioning code to run the heap sort, with two notable classes:

- A `Lab6` class, which runs all the sorting methods, printing out the mean and standard deviation of the collected running times for each sort, and the results of the various Z-tests. You must NOT change anything in this class.
- A `Stats` class, which computes the various statistics required in the above class. (You used it in Assignment 1.) You must NOT change anything in this class, either.

The file also includes an incomplete `Heap` class, which is meant to implement a max heap (except it doesn't include an insert method, which isn't needed in this application). When finished, it will include a heap sort method. You will finish this class (details below).

The `Heap` methods to complete: You should write the method bodies for the following methods, and not change anything else in this class:

- `isEmpty()`: Should return `true` if the heap is empty and `false` if it is not. (Look at the instance members and the constructors, which are already provided for you.)
- `parentIndex( int i )`: Should return the index of the parent of the item stored at index `i` in the array `heap`.

  Note that it should do no error checking — it shouldn't care if there is no parent.
- `leftChildIndex( int i )`: Should return the index of the left child of the item stored at index `i` in the array `heap`.

  Note that it should do no error checking — it shouldn't care if there is no left child.
- `rightChildIndex( int i )`: Should return the index of the right child of the item stored at index `i` in the array `heap`.

  Note that it should do no error checking — it shouldn't care if there is no right child.
- `siftDown( int startIndex )`: The item stored at position `startIndex` in array `heap` may not be in heap order with respect to one or both of its children (if it has any), but its descendants are all otherwise heap ordered.

  This method should sift the item at position `startIndex` down until either (a) it is in heap order with respect to its children (or child, if it only has one) or (b) it is in a leaf (has no children). Here, "sifting down" means repeatedly exchanging the item with its highest-priority child.
- `deleteMax()`: Should remove and return the highest-priority item. Of course, the heap should be a proper heap after this method is complete.
- `heapify()`: The array `heap` contains `heapSize` items, but is not heap ordered. Put the items into heap order so that we actually have a proper heap. (The heapify algorithm to use is described in the slides "`Lab6-HeapifyAndHeapSort`", which you can find on UM Learn where you got this lab.)
- `sort()`: Performs a heap sort on the array `heap`, which contains `heapSize` items that are already in heap order. (The heap sort algorithm is also described in the slides "`Lab6-HeapifyAndHeapSort`", which you can find on UM Learn where you got this lab.)

Note that when you are writing this code, you should make proper use of the other `Heap` methods — these methods should call one another as appropriate. (Note in particular that this class contains a swap method that doesn't need an array passed to it, because it always swaps two positions in the array `heap`.)