Heaps Lab #1

This lab is due at the end of the class period. Even if you do not complete the lab, you must turn in what you have at the end of the class period for grading.

Since following labs require completion of this lab, you will ultimately need to complete it. The only portion that will be graded is what you turn in at the end of class today.

1. Create a project to hold all of your code for this lab and the remaining heap labs.
2. DO NOT PLACE ANY CODE INSIDE OF A PACKAGE. IF YOU HAVE CODE INSIDE A PACKAGE, REMOVE IT FROM ITS PACKAGE.
3. Download the starting code which consists of `TreeDisplay` and `HeapDisplay.` Add this code to your project.
4. From your Binary Trees and Binary Search Trees projects, add `TreeUtil, TreeNode` and `FileUtil` code.
5. Create a `Main` class with a `main` method.
6. Create a class called `HeapUtils.`
7. If you have created any packages or any of your code is contained within a package, stop now and go back to step 1.
8. In the `main` method, create an array of size 12. Place 11 random integer values in the range [1…100) into the array starting at index 1 in the array.
9. In the `main` method, create a `HeapDisplay` object.
10. Run the main method and verify that you see a tree display of a complete tree with 11 nodes.
11. If you have created any packages or any of your code is contained within a package, stop now and go back to step 1.
12. Make sure your code is fully documented.

Throughout this lab and following labs, you must use the algorithms presented in class and no others. You may include a method to swap two array elements, but you may not use any other helper methods.

Exercise: Heapify 1

In this exercise, you will specify the `HeapUtils` method `heapify` which must have the following method signature:

```
public static void heapify(Comparable[] heap, int index, int heapSize)
```

The parameter `index` specifies the root of the tree that is being heapified (which may not be the root of the heap). The parameter `heap` is the array that contains the heap data, and `heapSize` is the size of the heap (which may not be the same as the array size - 1).

Fully document this method, including all algorithms and big o analysis. DO NOT WRITE ANY CODE. You may find it useful to specify a private method that swaps two values in an array. You will not need, nor are you allowed, any other helper methods.

If you are adding a method to swap values, include its method header and fully document it as well.

Exercise: Build Heap 1

In this exercise you will specify the `HeapUtils` method `BuildHeap.` The `BuildHeap` method must have the following method signature:

```
public static void buildHeap(Comparable[] heap, int heapSize)
```

The parameter `heap` is the input array representing a complete binary tree containing `heapSize` nodes. The values in the nodes are arranged in an arbitrary way. The postcondition for the method is that `heap` contains a complete binary tree satisfying the heap condition.

Write complete documentation for this method, including all algorithms used and run time analysis. You do not need, and are not allowed, any helper methods. DO NOT WRITE ANY CODE.

Exercise: Inserting and removing design

The `remove` method must have the following signature:

```
public static Comparable remove(Comparable[] heap, int heapSize)
```

It removes and returns the root value, leaving a complete binary tree that is one element smaller and meets the heap condition.

The `insert` method must have the following signature:

```
public static Comparable[] insert(
     Comparable[] heap, Comparable item, int heapSize)
```

It inserts `item` into the heap, maintaining the heap property, and returns the resulting heap.

Write complete documentation for both of these methods, including all algorithms and runtime analysis. DO NOT WRITE ANY CODE.

What to turn in

Print out your `HeapUtils` class, turn in your printout to your teacher and submit it to Athena2.