# Lab – Heaps and HeapSort

APCS with Data Structures.

In this lab, you will investigate the `Heap` data structure and how it can be applied to `Priority Queues` and sorting. Throughout this lab, you will need to pay attention to the style guide and the course policies.

**Throughout this lab, 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.**

As always, English usage counts. Do not submit anything that you would not be proud to show your English teacher.

## *Exercise 1 – Project Setup*

1. Create a project to hold all of your code for this lab.
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. In the `main` method, create an array of `Integer` objects. The array size must be 12. Place 11 random integer values in the range [1…100) into the array starting at index 1 in the array.
8. In the `main` method, create a `HeapDisplay` object. Add code to display your array as a `Heap.`
9. Run the main method and verify that you see a tree display of a complete tree with 11 nodes.
10. Make sure your code is fully documented.

## *Exercise: Heapify Design*

In this exercise, you will design and document 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 using complete sentences and paragraphs. Your documentation should adhere to the same rules as those in effect for your English essays. Your documentation must reflect your understanding of the algorithm and it must document your intended realization. Parroting back the algorithm given in class adds no value and will be scored appropriately. DO NOT WRITE ANY CODE YET. 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: BuildHeap Design**

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. Your documentation should adhere to the same rules as those in effect for your English essays. Your documentation must reflect your understanding of the algorithm and it must document your intended realization. Parroting back the algorithm given in class adds no value and will be scored appropriately. 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. Your documentation should adhere to the same rules as those in effect for your English essays. Your documentation must reflect your understanding of the algorithm and it must document your intended realization. Parroting back the algorithm given in class adds no value and will be scored appropriately. You do not need, and are not allowed, any helper methods. DO NOT WRITE ANY CODE.

**You must have your design checked off prior to proceeding. Failure to complete this step will incur a large grade penalty. Writing *ANY* code (other than method headers) for Heapify, BuildHeap, Insert or Remove prior to having your design checked will similarly result in a substantial penalty.**

### *Exercise: Coding the Methods*

Code and completely test `heapify, buildHeap, insert,` and `remove`. You must use the algorithms given in class. Use of any algorithm other than the one given in class will result in a score of 0 for this portion of the lab and a maximum score of 50% for the entire lab.

Generate a `main` method in the class `Main` and add fully documented code to test all of your methods. Your grade will reflect your test methodology and how well the algorithms are tested.

### *Exercise: HeapSort*

Design, document, code and test a method that accepts a binary heap represented as an array of `Comparable` objects and an integer parameter specifying the size of the heap data structure. The method must then use the algorithm given in class to implement a heap sort of the data in place. Your implementation must be O(NlogN) where N is the size of the heap. Use of any algorithm other than the one given in class will result in a score of 0 for this portion of the lab and a maximum score of 50% for the entire lab.