# CSC 220 Data Structures

## Program #7

## Heaps

Your objective is to implement a max heap and a min heap to store int keys. This means creating a public class for MaxHeap and a public class for MinHeap. You will use an int array as your backing data structure with help from the notes for accessing parent and child keys. The following operations must be implemented for your **MaxHeap** (make sure to put visibility (i.e. public or private) next to every function and field!):

- constructor
  - Should be a "default constructor" (i.e. no parameters) and should initialize the array to a fixed size large enough to hold the contents of our heap. A size of 7 is good enough for the Main class to run, however you may want to increase this value for your own tests.
- insert(key)
  - Adds a new key to the heap, maintaining the max heap property by calling the proper version of maxHeapify
- findMax
  - Returns the maximum key in the heap, but does **not** remove it.
  - If the heap is empty, this should return -1.
- extractMax
  - Returns the maximum key in the heap, and removes it from the heap, maintaining the max heap property by calling the proper version of maxHeapify.
  - If the heap is empty, this should return -1.
- maxHeapifyUp
  - Scans the heap, fixing violations of the max heap property
  - Start from the last inserted value and make your way up the tree, stopping when the max heap property is satisfied
  - You may use iteration or recursion here
  - This function should be marked as private since no code outside of this class will ever call this function directly
- maxHeapifyDown
  - Scans the heap, fixing violations of the max heap property
  - Start from the root and make your way down the tree, stopping when the max heap property is satisfied
  - You may use iteration or recursion here
  - This function should be marked as private since no code outside of this class will ever call this function directly
- isEmpty
  - Very similar to the operations from past abstract data types
  - Simply returns true if the heap has no keys in it, false otherwise

The following operations must be implemented for your **MinHeap**:

- constructor
    - Should be a "default constructor" (i.e. no parameters) and should initialize the array to a fixed size large enough to hold the contents of our heap. A size of 7 is good enough for the Main class, however you may want to increase this value for your own tests.
- insert(key)
    - Adds a new key to the heap, maintaining the min heap property by calling the proper version of minHeapify
- findMin
    - Returns the minimum key in the heap, but does **not** remove it
    - If the heap is empty, this should return -1.
- extractMin
    - Returns the minimum key in the heap, and removes it from the heap, maintaining the min heap property by calling the proper version of minHeapify
    - If the heap is empty, this should return -1.
- minHeapifyUp
    - Scans the heap, fixing violations of the min heap property
    - Start from the last inserted value and make your way up the tree, stopping when the min heap property is satisfied
    - You may use iteration or recursion here
    - This function should be marked as private since no code outside of this class will ever call this function directly
- minHeapifyDown
    - Scans the heap, fixing violations of the min heap property
    - Start from the root and make your way down the tree, stopping when the min heap property is satisfied
    - You may use iteration or recursion here
    - This function should be marked as private since no code outside of this class will ever call this function directly
- isEmpty
    - Very similar to the operations from past abstract data types
    - Simply returns true if the heap has no keys in it, false otherwise

The Main class is provided for you. You are strongly encouraged to test out your heap implementations yourself **before** trying the code out with the given Main.java file. I would suggest creating a file called MyMain.java, putting a public class called MyMain inside of it, and putting your main method inside of that. Use this to test your heaps. Then, when you think everything is working, run the entry point from my Main.java file.

For the Main class to work properly, you will need the given Stuff.class file. If compiling and running from command line, simply having Stuff.class in the same directly is sufficient. If using an IDE, you will need to use heap.jar (which just contains Stuff.class within it). Add this as an external jar to your project.

**Important Note:**
Until you get all methods (at least as method stubs) in both of your heap classes, you will continue to receive errors of the form "cannot find symbol". This is one of the reasons I encourage you to test your heaps out BEFORE using the given Main.java file. If you have everything implemented and working in your own test class but you are still receiving "cannot find symbol" errors when running Main.java, then you don't have the method header/signature done correctly. Look at the error message for help on getting the signature correct.

**For submission:**
Submit your MaxHeap.java and your MinHeap.java files only. I do not need any classes created for testing your heaps. I will test them using my Main.java file which is the same one given to you. Do not submit any .class files. Zip up both MaxHeap.java and MinHeap.java and submit the zip.

**Rubric:**

| # | ITEM | POINTS |
|---|---|---|
| 1 | MaxHeap fully working | 17 |
| 2 | MinHeap fully working | 17 |
| 3 | output is correct | 6 |
| | **TOTAL** | **40** |

| # | PENALTIES | POINTS |
|---|---|---|
| 1 | Doesn't compile | -50% |
| 2 | Doesn't execute once compiled (i.e. it crashes) | -25% |
| 3 | Late up to 1 day | -25% |
| 4 | Late up to 2 days | -50% |
| 5 | Late after 2 days | -100% |