# COMP 410 - Fall 2014
# Programming Assignment 3

## PROJECT DESCRIPTION

A ***min-max heap*** is a generalization of the binary heap that we have studied in class, that supports the additional operations **max** and **deleteMax**. For this assignment, you are to implement a min-max heap in an array by completing the methods in the following class declaration:

```
public class MinMaxHeap {
    private int currentSize;
    private int[] arr;

    public MinMaxHeap(int capacity){//Constructor
        arr = new int[capacity + 1];
        currentSize = 0;
    }

    public boolean isFull(){return currentSize == arr.length - 1;}
    public boolean isEmpty(){return currentSize == 0;}

    // COMPLETE THE FOLLOWING METHODS
    public void insert(int x){...}//PRE: The heap is not full

    public int min(){...}//PRE: The heap is not empty
    public int max(){... }//PRE: The heap is not empty

    public int deleteMin(){...}//PRE: The heap is not empty
    public int deleteMax(){...}//PRE: The heap is not empty

    //Private methods go here.
}
```

The methods `isFull()`, `isEmpty()`, `min()`, and `max()` should each have $\Theta(1)$ run-time; the remaining methods – `insert()`, `deleteMin()` and `deleteMax()` should each have $\Theta(\log N)$ runtime.

## NOTES

You will find it helpful to first read Question 6.18 of your text (appended to this document), and work out the answers to parts (a)–(c). (You do not need to turn in your answers to these questions.) You may also find it helpful to read the paper

> *Atkinson, Michael D., J-R. Sack, Nicola Santoro, and Thomas Strothotte. "Min-max heaps and generalized priority queues." Communications of the ACM 29, no. 10 (1986): 996-1000*

You can find the paper at the following URL:
http://www.akira.ruc.dk/~keld/teaching/algoritmedesign_f03/Artikler/02/Atkinson86.pdf

**Do *not* use any in-built Java implementation (e.g., Stack, Queue, Lists, ArrayList, etc) from the java.util.* API. You *must* implement the `MinMaxHeap` class from scratch.**

## IMPLEMENTATION REQUIREMENT

Please note that in the constructor, the array `arr` has a length of `capacity` + 1. You are expected to store the root of the heap at index 1, not 0. This is because it makes the math simpler: to find the depth level of a node, you would compute $\log_2(idx)$, where $idx$ is the index of the node in the array. If the root was at 0 then this math wouldn't work. By not following this instruction and storing your root at index 0, you will lose points.

## GRADING RUBRIC

- (20 points) `insert` method correctly implemented
- (15 points) `min` method correctly implemented
- (15 points) `max` method correctly implemented
- (20 points) `deleteMin` method correctly implemented
- (20 points) `deleteMax` method correctly implemented
- (10 points) Clean code

## SUBMISSION INSTRUCTIONS

Upload all your source code in a `.zip` file to Sakai. You are responsible for ensuring that your program compiles and functions properly. Any non-functioning program will receive a zero.

You are welcome to use any method you would like for testing your code, but please do not include any code that is dependent on external libraries, such as JUnit tests.

## HONOR CODE

Please review the honor code description from the course syllabus. No collaboration (with anyone) is permitted in assignments. Collaboration in assignments, or the use of code not the students own, constitutes an honor code violation. Any violation will be reported to the Student Attorney General.

## APPENDIX – QN 6.18 OF THE TEXT

6.18 A **min-max heap** is a data structure that supports both `deleteMin` and `deleteMax` in $O(\log N)$ per operation. The structure is identical to a binary heap, but the heap-order property is that for any node, $X$, at even depth, the element stored at $X$ is smaller than the parent but larger than the grandparent (where this makes sense), and for any node $X$ at odd depth, the element stored at $X$ is larger than the parent but smaller than the grandparent. See Figure 6.57.

a. How do we find the minimum and maximum elements?

*b. Give an algorithm to insert a new node into the min-max heap.

*c. Give an algorithm to perform `deleteMin` and `deleteMax`.

*d. Can you build a min-max heap in linear time?

**e. Suppose we would like to support `deleteMin`, `deleteMax`, and `merge`. Propose a data structure to support all operations in $O(\log N)$ time.