Binary Heap (Min-heap, Max-heap implementation)

Summary: In this lab you will implement an array-based `MinHeap` class and an array-based `MaxHeap` class. Both of these classes will be of type `int`. The methods for each class are listed below. You will then write test cases to check that your implementations have correct functionality. Each method should have a test case. Check for ordinary and exceptional behavior. Include exceptions in your heap classes. Use `IndexOutOfBoundsException` from the `java.lang`.

Note: You must include the methods, constructors, and fields as specified below and use exceptions in your heap classes, to indicate when an insert is being attempted on a full heap, or an operation is being attempted on an empty heap. Your test cases should include using try/catch blocks that check for correct functionality with exception handling.

`MinHeap class`

- `int currentSize` – number of items currently in the heap
- `int[] h` – internal array to hold heap items
- `MinHeap()` – Default constructor - Constructs an empty binary min heap.
- `MinHeap(int[] A)` – Constructor - Constructs a binary min heap, via `buildMinHeap()`. You can use the built-in `length` property to determine the size of the array.
- `void buildMinHeap()` – Builds a binary min heap in linear time from an unsorted array. (See Lecture 6 slides)
- `int heapMin()` – Returns the minimum key in the heap. If empty, throw exception.
- `void heapExtractMin()` – Removes minimum key from heap. If empty, throw exception.
- `void minHeapInsert(int key)` – Insert key into heap. If full, throw exception.
- `void trickleDown(int index)` – `private` helper function - Maintains the heap property between a parent node located at `index` in array, and its children. Used in `buildMinHeap()`. (See Lecture 6 slides)
- `void trickleUp(int index)` – `private` helper function - Maintains the heap property between a node located at `index` in array, and its parent. Use in `minHeapInsert(int key)`. (See Lecture 6 slides)

`MaxHeap class`

- `int currentSize` – number of items currently in the heap
- `int[] h` – internal array to hold heap items
- `MaxHeap()` – Default constructor - Constructs an empty binary max heap.
- `MaxHeap(int[] A)` – Constructor – Constructs binary max heap, via `buildMaxHeap()`. You can use the built-in `length` property to determine the size of the array.
- `void buildMaxHeap()` – Builds a binary max heap in linear time from an unsorted array. (See Lecture 6 slides)
- `int heapMax()` – Returns the maximum key in the heap. If empty, throw exception.
- `void heapExtractMax()` – Removes maximum key from heap. If empty, throw exception.
- `void maxHeapInsert(int key)` – Insert key into heap. If full, throw exception.

- `void trickleDown(int index)` – `private` helper function - Maintains the heap property between a parent node located at `index` in array, and its children. Used in `buildMaxHeap()`. (See Lecture 6 slides)
- `void trickleUp(int index)` – `private` helper function - Maintains the heap property between a node located at `index` in array, and its parent. Use in `maxHeapInsert(int key)`. (See Lecture 6 slides)

Here is an example class stub for `MaxHeap`.

```
public class MaxHeap {
  private int h[];
  private int heapSize; // the number of items stored in h
  public static final CAPACITY = 100;
  public MaxHeap() { //Initialize h of size CAPACITY }}
  public MaxHeap(int[] A) {
    //Initialize h of size CAPACITY
    //Copy A into h
    buildMaxHeap();
  }
  public void buildMaxHeap() {
    for (int i = (heapSize/2)-1; i >= 0; i--)
      trickleDown(i);
  }
  // Rest of MaxHeap class implementation
}
```

File I/O

Create 2 input files for testing: both of type `int`. The files will be named `input_int_small.txt` and `input_int_large.txt`. The content of `input_int_small.txt` is listed below. The second file should be a file of random integers of size N=10000. When using the larger input file set `CAPACITY = 10000`. For testing insert into a full heap (which should throw an exception), add one additional `int` to `input_int_large.txt`.

`input_int_small.txt`:
20 1 11 15 6 9 3 5 12 4

For testing, your program must write the contents of the heap instances to the console in the following format.

```
input_int_small.txt:
max heap
20
15 11
12 6 9 3
5 1 4
```

```
min heap
1
4 3
5 6 9 11
15 12 20
```

You are required to demo for TA the following:

1. Demo your handling of exceptions thrown by `heapExtractMax()`, `heapExtractMin()`, `heapMax()`, and `heapMin()` on an empty heap.
2. Demo your handling of exceptions thrown by `maxHeapInsert(int key)` and `minHeapInsert(int key)` on a full heap.
3. Demo that your heap classes correctly build heaps.
4. Demo that your heap classes correctly implement inserts.
5. Demo by a sorted successive sequence of outputs from `heapMax()/heapExtractMax()`, and `heapMin()/heapExtractMin()`, your heap classes correctly maintain a heap after removals.

Note: You must reset CAPACITY back to 100 for both classes before submitting your work.

For additional testing of your classes, you are welcome to use random.org to generate input files.

Demo: Demo your working code for your TA.
Submission: Submit your work as `lab4.tgz` via turn-in link on Piazza. The tar archive `lab4.tgz` must contain the following files only: `Main.java`, `MinHeap.java`, `MaxHeap.java`, `input_int_small.txt`, `input_int_large.txt`. If you do not finish and demo in your lab session, also include a `README` text file that clearly states what functionality your program successfully implements and what is missing or not working properly.

Note: The name of the archive, main, and source files for the heap classes must be as listed above. No other files names for these items will be accepted.

Rubric:
20 pts Attendance (On-time, attend entire lab)
80 pts Perfect functionality
-15 pts Minor errors/bugs in code such as partially incomplete methods and/or incorrect output.
-15 pts Major missing functionality or significant errors/bugs. Examples are incorrect error checking when opening a file or inserting and removing from the heap classes.