LAB 2 - Max binary Heaps and Young Tableaux

Submission Deadline: Feb. 9, 11:59 pm
Assessment: 5% of the total course mark.

Description:

For this assignment you are required to write the `Java` classes `MaxBinHeap` and `YoungT`. They can be declared in the same package. Class `MaxBinHeap` represents binary max heaps that store integers (they store only the keys). Class `YoungT` represents Young tableaux that store integers. All the operations have to be implemented efficiently (i.e., with the smallest asymptotic time complexity).

You are not allowed to use any predefined `Java` methods other than for output (print, println, printf).

Definitions:

A binary max heap stores the items in a binary tree. The binary tree has all the levels completely filled, except possibly for the last level, which is filled from left to right up to some point (this is the heap structure property). The key at each node is larger than or equal to the key at any child (the max-heap ordering property). The binary tree nodes are stored internally in an array. The first position could be empty, the following array locations are occupied with the tree nodes in level order (i.e. root, children of root, nodes at second level from left to right, etc.). Binary max heaps support the operations `insert` and `deleteMax`.

A $k \times n$ Young tableau is a $k \times n$ matrix such that the entries of each row are in nondecreasing order from left to right and the entries of each column are in nondecreasing order from top to bottom. Some of the entries of a Young tableau may be ∞, which we treat as a nonexistent element. Thus, a $k \times n$ Young tableau can be used to hold r numbers, where $r \leq kn$.
Example:

$$\begin{pmatrix} 4 & 9 & 11 & 29 \\ 5 & 20 & 34 & \infty \\ 25 & 30 & 41 & \infty \\ 28 & 31 & \infty & \infty \end{pmatrix}$$

A $k \times n$ Young tableau y is empty if $y(0,0) = \infty$ and is full if $y(k-1, n-1) < \infty$.

Young tableaux support insertions, searches and `deleteMin` operations that run in $\Theta(k+n)$ in the worst case.

Specifications:

Class `MaxBinHeap` **has only two instance fields**, namely a reference to the array storing the keys (a reference variable of type `int[]`) and an integer representing the size of the heap. Both have to be `private`.

Class `MaxBinHeap` contains the following constructors:

- `public MaxBinHeap( int n)` - constructs an empty `MaxBinHeap`. The parameter indicates the size of the array that is allocated. If $n < 10$, it should be reset to 10.
- `public MaxBinHeap(int[] a)` - constructs a `MaxBinHeap` that stores the integers from array `a`. The constructor should not modify the input array. The running time should be $\Theta(n)$ in the worst case. Hint: you can use the appropriate `buidHeap` operation implemented as a private method.

Class `MaxBinHeap` contains only the following public methods (you may declare additional `private` methods, but not `public` methods):

- `public int getSize()` - returns the size of the heap (the number of items stored in the heap).
- `public void insert(int x)` - inserts the key `x` in the heap. If the array is not big enough to hold the new key, a larger array (of double size) should be allocated.
- `public int readMax() throws RuntimeException` - returns the largest key without removing it from the heap. If the heap is empty, it throws an exception with a message indicating that.
- `public int deleteMax() throws RuntimeException` - returns the largest key and removes it from the heap. If the heap is empty, it throws an exception with a message indicating that.
- `public String toString()` - returns a String that lists the items stored in the heap in level order, separated by a comma and a space.
- `public static void sortArray(int[] a)` - sorts the array `a` in place using HeapSort.

Class `YoungT` **has only three instance fields**, namely a reference to the 2D array storing the integers (a reference variable of type `int[][]`), an integer storing the number of finite integers in the tableau, and one integer representing ∞. All three have to be `private`.

Class `YoungT` contains the following constructors:

- `public YoungT(int k, int n, int infinity)` - constructs an empty $k \times n$ `YoungT`. The last parameter indicates the value for ∞. If $k < 10$, it should be reset to 10. If $n < 10$, it should be reset to 10. If `infinity`< 100, it should be set to 100.
- `public YoungT(int[][] a)` - constructs a `YoungT` object storing the integers in the input 2D array. The Young tableau should have the same dimensions as the input array. The value for ∞ should be reset to 10 times the largest array element.

Class `YoungT` contains only the following public methods (you may declare additional `private` methods, but not `public` methods):

- `public int getNumElem()` - returns the number of finite integers stored in the tableau.
- `public int getInfinity()` - returns the number that represents ∞.
- `public boolean isEmpty()` - returns `true` if the tableau is empty.

- `public boolean isFull()` - returns `true` if the tableau is full.
- `public boolean insert(int x)` - inserts the integer `x` in the Young tableau. If `x` is larger than or equal to the value that represents ∞ or if the tableau is full, no insertion is performed and `false` is returned. Otherwise, the insertion is performed and `true` is returned.
- `public int readMin() throws RuntimeException` - returns the smallest element without removing it from the tableau. If the tableau is empty, it throws an exception with a message indicating that.
- `public int deleteMin() throws RuntimeException` - returns the smallest element and removes it from the tableau. If the tableau is empty, it throws an exception with the message indicating that.
- `public boolean find(int x) throws RuntimeException` - returns `true` if `x` is in the tableau and `false` otherwise. It also prints the sequence of elements (including "infinity") that where compared with `x` (the element in each probed array position should appear only once in the sequence). If the tableau is empty or if `x` is larger than or equal to the value that represents ∞, the method throws an exception with a message indicating that.
- `public String toString()` - returns a String that lists all the numbers in the matrix including "infinity" in raster scan order (each row left to right, starting with the row on top), separated by a comma and a space.

You have to include meaningful comments at key points in your code.

Submission Instructions: Submit the source code for each `Java` class in a separate text file. Include the name of the class, your name and student number in the name of the file. For instance, if your name is "Ellen Davis" and the student number is 12345 then the two files should be named "MaxBinHeapEllenDavis12345.txt" and "YoungTEllenDavis12345.txt"