1. S is a structure of type MysteriousStructure. All that you know about S is that it has an iterator method S.iterator() that returns a value of type Iterator<Integer>, and this iterator goes through all of the values stored in S. Write function

\[
\text{Integer largest(MysteriousStructure S)}
\]

that returns the largest value in S, or null if S is empty.

\[
\text{Integer largest(MysteriousStructure S) \{} \\
\quad \text{Iterator<Integer> it = S.iterator();} \\
\quad \text{Integer big;} \\
\quad \text{if (!it.hasNext() )} \\
\quad \quad \text{return null;} \\
\quad \text{else} \\
\quad \quad \text{big = it.next();} \\
\quad \text{while (it.hasNext() ) \{} \\
\quad \quad \text{Integer x = it.next();} \\
\quad \quad \text{if (x > big)} \\
\quad \quad \quad \text{big = x;} \\
\quad \quad \text{\}\} \\
\quad \text{return big;} \\
\text{\}}
\]
2. Here is a picture of a binary tree with an integer stored in each node.

List the node values in pre-order, in-order and post-order traversals of this tree. Label your traversals so I can tell which is which.

pre-order:  50 20 10 100 85 80 90 200
in-order:  10 20 50 80 85 90 100 200
post-order:  10 20 80 90 85 200 100 50
3. Here is an AVL tree

and here is a “level-by-level” listing of its node values: 50 20 100 10 85 200 80 90 (just taking the node at level 0, then the nodes at level 1, etc, each level listed from left to right). Find the AVL tree that results from adding value 95 to this tree and give its level-by-level listing. [Note: I am asking for this level-by-level listing so you don’t need to draw the tree; if you find it easier to draw the tree, do that rather than give the listing.

First, here is the tree with the added node. Z is the first node up from the addition that fails the AVL property; Y is Z’s tallest child, X is Y’s tallest child. a, b, and c are X, Y and Z in increasing order. h1 through h4 are the four children of X, Y, and Z (who are not X, Y, and Z) from left to right.
Then we build the following tree:

Here is the result:

Level-by-level this is 50 20 90 10 85 100 80 95 200
4. Here is an ArrayList that represents a heap

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\_ & 10 & 20 & 30 & 40 & 50 & 30 & 40 & 85 & 95 \\
\end{array}
\]

a. Give the ArrayList for the result of adding value 15 to this heap. You don’t need to write the index numbers for your ArrayList.

Start with this tree with 15 added as the next leaf:

Then percolate up the 15;

This gives the ArrayList \([ \_ 10 15 30 40 20 30 40 85 95 50 ]\)
b. Go back to the original heap, before you did the addition in part (a). Give the ArrayList that results from removing the smallest value from this heap.

Start here:

Remove the 10 and put the value of the last leaf there:

Finally, percolate the 95 down:

This gives the ArrayList \[
\text{[ _ 20 40 30 85 50 30 40 95]}
\]
5. **Give an algorithm in English for removing the value at a specific index from a heap.**
   [Note: this is not something we did in class. I am asking you to figure out an algorithm. Think about what a heap is, and how we did other things like inserting into one or removing its smallest value. If you find they help you can make use of percolateUp( ) and/or percolateDown( ) without rewriting them.]

   Put the value of the last leaf (the value at index $size$) at the specified index.
   Decrement the size.
   If the value we just inserted is less than its parent, percolate it up. If instead it is greater than either of its children, percolate it down. It can’t be both less than its parent and greater than one of its children.
6. I want to store the following key-value pairs in a HashMap `<String, Integer>` built on a
table of size 9:

<table>
<thead>
<tr>
<th>Key</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>20</td>
<td>50</td>
<td>30</td>
<td>30</td>
<td>40</td>
<td>25</td>
<td>10</td>
</tr>
</tbody>
</table>

Here are the values of the `hashValue()` function for the keys:

<table>
<thead>
<tr>
<th>String</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashValue</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

I am going to add the data to the table in alphabetical order of the strings (i.e., first A,
then B, then C, etc.). **Give the value in each entry of the table** (it will either be a key-
value pair such as (A 20) or NULL) **after all of these additions**:

<table>
<thead>
<tr>
<th>Index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(B 50)</td>
</tr>
<tr>
<td>1</td>
<td>(D 30)</td>
</tr>
<tr>
<td>2</td>
<td>(E 40)</td>
</tr>
<tr>
<td>3</td>
<td>(G 10)</td>
</tr>
<tr>
<td>4</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>(A 20)</td>
</tr>
<tr>
<td>7</td>
<td>(F 25)</td>
</tr>
<tr>
<td>8</td>
<td>(C 30)</td>
</tr>
</tbody>
</table>