QUEENS COLLEGE Department of Computer Science
CSCI 313 Midterm Exam Fall 2018 11.07.18

Instructor: Alex Ryba

07.45am – 09.00am, Wednesday, November 07, 2018

Problem 1 *(10 points)* (a) Write complete code for the `interface Comparable<K>`.

Answer:

```
interface Comparable<K> {
   int compareTo(K k);
}
```

(b) Give a Θ-estimate for a function $t(n)$ that satisfies:

$$t(n) = 3(t(n/2) + \sqrt{n} + 2n \log n)$$

Answer: $t(n) = \Theta(n^{\log_2 3})$

(c) How many nodes belong to the smallest *proper binary tree* that has height n?

Answer: $2n + 1$.

(d) The following code to compute the height of a tree node has errors. Add a comment after any line(s) that need to be corrected to indicate the correct line that is needed instead. Some of the lines may be unusual but not incorrect — you will lose credit for marking these as errors.

```
public int height(TNode n) {

  if (isLeaf(n)) return 1;                      // return 0;

  int answer = 0;

  Iterator<TNode> c = n.children();

  while (c.hasNext()) {

    TNode child = c.next();

    if (height(c.next()) >= answer)      // if (height(child) >= answer)

      answer = height(child);

  }

  return answer;                          // return answer + 1;

}
```

(e) A heap is implemented so that its elements are stored as entries of an array called `data`. An instance variable `size` gives the number of elements in the heap. Suppose that the implementation leaves the array entry with index 0 permanently empty. This means that the minimum element occupies entry `data[1]`. (Note that this is a little different from the implementation we considered in class where the minimum entry occupied entry `data[0]`.)

The following is an implementation of the method bubbleUp.

```
private void bubbleUp(int n) {
  if (CONDITION) return; // at root
  int p = PARENT;
  K dn = data[n];
  K dp = data[p]; // parent data
  if (dn.compareTo(dp) >= 0) return; // no problems
  swapData(n, p);
  bubbleUp(p);
}
```

Write Java code as replacements for `PARENT` and `CONDITION` to complete this to a working method.

Answer:

`PARENT` is `p = n / 2`

`CONDITION` is `n <= 1`

Problem 2 *(10 points)* The `class ItQueue<T>` implements the interfaces Queue and Iterable.

Write a method with title

`ItQueue<String> deFred(ItQueue<String> q)`

that returns a new queue containing all those entries of `q` that are not *Freddy*. The original queue `q` must remain unchanged — none of its nodes should be either removed or replaced. (Solutions with more than 10 lines of code will lose credit.)

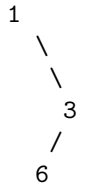
Answer:

```
public static ItQueue<String> deFred(ItQueue<String> q) {
   ItQueue<String> ans = new ItQueue<>();
   for (String x:q)
     if (!x.equals("Freddy")) ans.enqueue(x);
   return ans;
}
```

Problem 3 *(10 points)* The class `BinaryNode<T>` represents a node of a binary tree. It has instance variables: `BinaryNode<T> left, BinaryNode<T> right, T data`.

An `eOrder` traversal of a node `n` begins with the data at `n`. Then, if the left child of `n` is not null the traversal continues with the eOrder traversal of that left child followed by the data at `n` again. Finally if the right child of `n` is not null it continues with the eOrder traversal of that right child followed by the data at `n` again.

For the following tree, the traversal of the root would be: 1,2,4,2,5,2,1,3,6,3,1.



Write code for a `BinaryNode<T>` method with title line:

`public static void eOrder(BinaryNode<T> n, ArrayList<T> answer).`

The goal of the method is write the traversal of the node `n` onto the end of the `ArrayList answer`. (Do not use any instance variables or methods other than those given above. Solutions with more than 12 lines of code will lose credit.)

Answer:

```
public static void eOrder(BinaryNode<T> n, ArrayList<T> answer) {
   if (n == null) return;
   answer.add(n.data);
   if (n.left != null) {
       eOrder(n.left, answer);
       answer.add(n.data);
   }
   if (n.right!= null) {
       eOrder(n.right, answer);
       answer.add(n.data);
   }
}
```