CSE 373 Winter 2013 Midterm Exam
ANSWER KEY

1. Big-Oh Analysis

a) $O(N)$
b) $O(N \log N)$
c) $O(1)$
d) $O(N^3)$
e) $O(N \log N)$

2. Java / Guava Collection Programming

As with any programming problem, there are many correct solutions. Here are some:

```
// solution 1: longer
public static Set<String> highlyPaidWomen(BiMap<String, String> marriage,
                                          Map<String, Double> salary) {
    Set<String> women = new TreeSet<String>();
    for (String person : salary.keySet()) {
        if (marriage.containsValue(person)) {                // a woman
            String husband = marriage.inverse().get(person);
            if (salary.containsKey(husband)) {               // both employed
                double husbandSalary = salary.get(husband);
                double wifeSalary    = salary.get(person);
                if (wifeSalary > husbandSalary) {
                    women.add(person);
                }
            } else {
                women.add(person);   // she's employed, he is not
            }
        }
    }
    return women;
}


// solution 2: store inverse in variable; shorter
public static Set<String> highlyPaidWomen(BiMap<String, String> marriage,
                                          Map<String, Double> salary) {
    BiMap<String, String> inverse = marriage.inverse();
    Set<String> richWomen = new TreeSet<String>();
    for (String woman : salary.keySet()) {
        if (inverse.containsKey(woman)) {    // marriage.containsValue(woman) OK
            String man = inverse.get(woman);
            if (!salary.containsKey(man) || salary.get(woman) > salary.get(man)) {
                richWomen.add(woman);    // order of tests in || matters!
            }
        }
    }
    return richWomen;
}
```

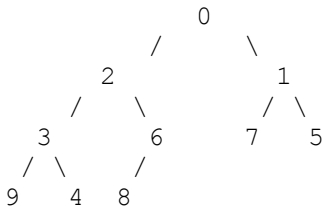
3. Hashing

```
 0   1     2   3    4     5   6   7   8   9   10  11  12    13    14  15  16  17  18  19
+---+-----+---+----+-----+---+---+---+---+---+---+---+-----+-----+---+---+---+---+---+-----+
| / |81=18|XXX|63=9|22=33| / | / | / | / | / | / | / |32=42|72=88| / | / | / | / | / |999=9|
+---+-----+---+----+-----+---+---+---+---+---+---+---+-----+-----+---+---+---+---+---+-----+

size        =  6
capacity    = 20
load factor =  0.3
```

4. Heaps

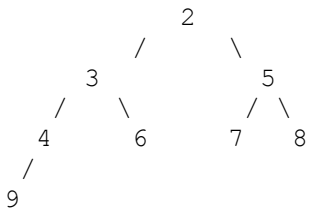
a) after all adds, final min-heap tree:



array:

```
 0  1  2  3  4  5  6  7  8  9 10 11 12
[/, 0, 2, 1, 3, 6, 7, 5, 9, 4, 8, /, ...]
```

b) after 2 removes, final min-heap tree:

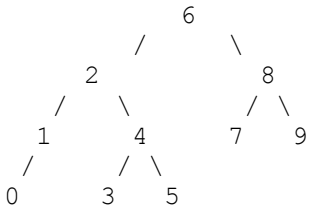


5. AVL Trees

a)

- adding 0 causes case-1 R rotation on 8
- adding 7 causes case-3 RL rotation on 1
- adding 3 causes case-3 RL rotation on 1

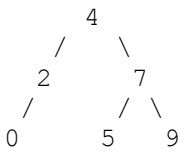
- after all adds, AVL tree:



b)

- removing 8 causes case-5 R rotation on 7
- removing 1 causes case-3 RL rotation on 2

after all removes, AVL tree:



6. Hash Map Implementation

As with any programming problem, there are many correct solutions. Here are some:

```
// solution 1: no special case for k <= 0; while loop to remove
public void trimChains(int k) {
    for (int i = 0; i < elements.length; i++) {
        int length = 0;
        Node current = elements[i];
        while (current != null) {                    // count length of chain
            length++;
            current = current.next;
        }

        while (length > k && elements[i] != null) {   // remove nodes from front
            elements[i] = elements[i].next;
            length--;
            size--;
        }
    }
}


// solution 2: special-case k <= 0; for loop to remove
public void trimChains(int k) {
    if (k <= 0) {
        for (int i = 0; i < elements.length; i++) {
            elements[i] = null;
        }
        size = 0;
    } else {
        for (int i = 0; i < elements.length; i++) {
            int length = 0;
            Node current = elements[i];
            while (current != null) {
                length++;
                current = current.next;
            }

            if (length > k) {
                int numToRemove = length - k;
                current = elements[i];
                for (int j = 0; j < numToRemove; j++) {
                    current = current.next;
                }
                elements[i] = current;
                size -= numToRemove;
            }
        }
    }
}


// solution 3: adjusts list while counting length
public void trimChains(int k) {
    k = Math.max(0, k);   // fix for negatives
    for (int i = 0; i < elements.length; i++) {
        int length = 0;
        Node current = elements[i];
        while (current != null) {
            length++;
            if (length > k) {
                elements[i] = elements[i].next;
                size--;
            }
            current = current.next;
        }
    }
}
```