CSE 373 Practice Midterm Exam #1
ANSWER KEY

1. Big-Oh Analysis

a) $O(N^3)$
b) $O(N^2)$
c) $O(N)$
d) $O(N \log N)$
e) $O(N^2)$
f) $O(N)$

2. Java / Guava Collection Programming

```
public static int countDuplicateValues(
        Map<String, Integer> map1, Map<String, Integer> map2) {
    Set<Integer> values = new HashSet<Integer>();
    int dupes = 0;
    for (int n : map1.values()) {
        if (values.contains(n)) {
            dupes++;
        } else {
            values.add(n);
        }
    }
    for (int n : map2.values()) {
        if (values.contains(n)) {
            dupes++;
        } else {
            values.add(n);
        }
    }
    return dupes;
}

public static int countDuplicateValues(
        Map<String, Integer> map1, Map<String, Integer> map2) {
    Multiset<Integer> values = HashMultiset.create();
    for (int n : map1.values()) {
        values.add(n);
    }
    for (int n : map2.values()) {
        values.add(n);
    }
    int dupes = 0;
    for (int n : values.elementSet()) {
        int count = values.count(n);
        if (count > 1) {
            dupes += count - 1;
        }
    }
    return dupes;
}
public int countDuplicateValues(Map<String, Integer> m1, Map<String, Integer> m2) {
    Collection<Integer> v1 = m1.values();
    Collection<Integer> v2 = m2.values();
    Set<Integer> unique = new HashSet<Integer>(v1);
    unique.addAll(v2);
    return v1.size() + v2.size() - unique.size();
}
```

3. Java Class Programming for Collections

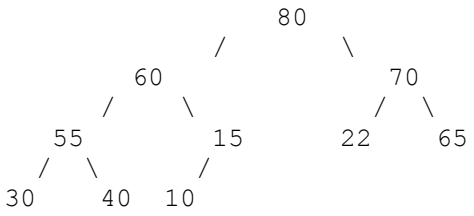
```
public class Appointment implements Comparable<Appointment> {
    private Date date;
    private Time startTime;
    private int duration;

    ...

    public int compareTo(Appointment other) {
        if (!date.equals(other.date)) {
            return date.compareTo(other.date);
        } else if (!startTime.equals(other.startTime)) {
            return startTime.compareTo(other.startTime);
        } else {
            return duration - other.duration;
        }
    }

    public boolean equals(Object o) {
        if (o instanceof Appointment) {
            Appointment other = (Appointment) o;
            return date.equals(other.date) && startTime.equals(other.startTime)
                && duration == other.duration;
        } else {
            return false;
        }
    }

    public int hashCode() {
        return 37 * date.hashCode() +
              131 * startTime.hashCode() +
              417 * duration;
    }
}

---

    // Guava solution
    public int compareTo(Appointment other) {
        return ComparisonChain.start()
            .compare(date, other.date)
            .compare(startTime, other.startTime)
            .compare(duration, other.duration)
            .result();
    }

    public boolean equals(Object o) {
        return o instanceof Appointment && compareTo((Appointment) o) == 0;
    }

    public int hashCode() {
        return Objects.hashCode(date, startTime, duration);
    }
```

4. Hashing

```
   key            value
   +--------------+--------------+
0  |      /       |     /        |    size: 4
   +--------------+--------------+    capacity: 10
1  |      /       |     /        |    load factor: 0.4
   +--------------+--------------+
2  |      /       |     /        |
   +--------------+--------------+
3  |     33       |    Kona      |
   +--------------+--------------+
4  |     XX       |    XX        |
   +--------------+--------------+
5  |     15       |    Daisy     |
   +--------------+--------------+
6  |      6       |    Tina      |
   +--------------+--------------+
7  |      7       |    Meghan    |
   +--------------+--------------+
8  |     XX       |    XX        |
   +--------------+--------------+
9  |      /       |     /        |
   +--------------+--------------+
```

5. Heaps

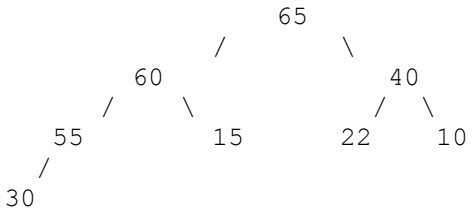
a) after all adds, final max-heap tree:



array:

```
 0   1   2   3   4   5   6   7   8   9  10  11  12
[/, 80, 60, 70, 55, 15, 22, 65, 30, 40, 10,  /, ...]
```

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

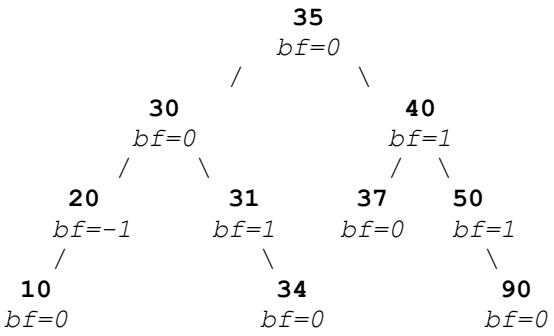


array:

```
 0   1   2   3   4   5   6   7   8   9
[/, 65, 60, 40, 55, 15, 22, 10, 30,  /, ...]
```

6. AVL Trees

after all adds, AVL tree, with balance factor of each node:



7. Heap Priority Queue Implementation

```
public int nodesAtLevel(int level) {
    if (level <= 0) {
        throw new IllegalArgumentException();
    }
    int start = 1;
    for (int i = 1; i <= level - 1; i++) {
        start = start * 2;  // go to leftmost child at the given level
    }
    if (start > size) {
        throw new IllegalArgumentException();
    }

    int end = Math.min(size + 1, 2 * start);
    return end - start;
}

public int nodesAtLevel(int n) {
    int start = (int) Math.pow(2, n - 1);
    if (n < 1 || size < start) {
        throw new IllegalArgumentException();
    }
    int end = Math.min(start * 2, size + 1);
    return end - start;
}
```