CSE 373 Practice Final Exam #2
ANSWER KEY

1. Big-Oh Analysis

a) $O((\log N)^2)$
b) $O(1)$
c) $O(N \log N)$
d) $O(N^2)$

2. Java / Guava Collection Programming

```
public static String commonFirstName(List<String> first, List<String> last) {
    if (first.size() != last.size() || first.isEmpty()) {
        throw new IllegalArgumentException();
    }
    Multimap<String, String> names = HashMultimap.create();

    // Note that you MUST use an implementation that implements SetMultimap interface
    // (ArrayListMultimap or LinkedListMultimap would not work)
    for (int i = 0; i < first.size(); i++) {
        names.put(first.get(i), last.get(i));
    }
    int maxNum = 0;
    String maxName = "";
    for (String firstName : names.keySet()) {
        int num = names.get(firstName).size();
        if (num > maxNum) {
            // Since you're guaranteed there was at least one name in the list,
            // this will be true at least once.  >= would also work.
            maxNum = num;
            maxName = firstName;
        }
    }
    return maxName;
}
```

3. Heaps

```
a) after adds

              10
            /      \
       19             21
      /    \        /    \
   75       32    107     64
  /  \     /  \
138  96 209  53

array: [/, 10, 19, 21, 75, 32, 107, 64, 138, 96, 209, 53]


b) after two remove-mins

              21
            /      \
       32             64
      /    \        /    \
   75       53    107    209
  /  \
138  96

array: [/, 21, 32, 64, 75, 53, 107, 209, 138, 96]
```

4. Sort Tracing

```
a) merge sort
 0    1   2   3   4   5   6   7   8   9
[26,  7, 63, 42, 12, 34,  1, 10, 14, 30]

[26,  7, 63, 12, 42]
[26,  7]
[26] [7]
[ 7, 26]
       [63, 42, 12]
       [63]
           [42, 12]
           [42][12]
           [12, 42]
       [12, 42, 63]
[ 7, 12, 26, 42, 63]
                    [34,  1, 10, 14, 30]
                    [34,  1]
                    [34] [1]
                    [ 1, 34]
                            [10, 14, 30]
                            [10]
                                [14, 30]
                                [14][30]
                                [14, 30]
                            [10, 14, 30]
                    [ 1, 10, 14, 30, 34]
[ 1,  7, 10, 12, 14, 26, 30, 34, 42, 63]

b) heap sort
 0   1   2   3   4   5   6   7   8   9
[11, 37, 99, 77, 60, 68, 53, 10, 70, 56]
turn into max heap:
[11, 37, 99, 77, 60, 68, 53, 10, 70, 56]
[11, 37, 99, 77, 70, 68, 53, 10, 60, 56]
[11, 37, 99, 77, 70, 68, 53, 10, 60, 56]
[11, 37, 99, 77, 70, 68, 53, 10, 60, 56]
[11, 99, 70, 77, 60, 68, 53, 10, 37, 56]
[99, 77, 70, 53, 60, 68, 11, 10, 37, 56]
remove-max, move to end:
[77, 70, 68, 53, 60, 56, 11, 10, 37, 99]
[70, 68, 60, 53, 37, 56, 11, 10, 77, 99]
[68, 60, 56, 53, 37, 10, 11, 70, 77, 99]
[60, 56, 37, 53, 11, 10, 68, 70, 77, 99]
[56, 53, 37, 10, 11, 60, 68, 70, 77, 99]
[53, 37, 11, 10, 56, 60, 68, 70, 77, 99]
[37, 11, 10, 53, 56, 60, 68, 70, 77, 99]
[11, 10, 37, 53, 56, 60, 68, 70, 77, 99]
[10, 11, 37, 53, 56, 60, 68, 70, 77, 99]

c) bucket sort
 0  1  2  3  4  5  6  7  8  9
[6, 0, 9, 3, 6, 5, 2, 3, 1, 1]
create counts:
 0  1  2  3  4  5  6  7  8  9
[1, 2, 1, 2, 0, 1, 2, 0, 0, 1]
use to sort:
[1x0, 2x1, 1x2, 2x3, 1x5, 2x6, 1x9]
[0, 1, 1, 2, 3, 3, 5, 6, 6, 9]
```

5. Sorting Algorithm Implementation

```
// Sorts the characters in a using the bucket sort algorithm.
// Assumes that a contains only 'a' - 'z'.

public static void charBucketSort(char[] a) {
    int[] counters = new int[26];
    for (char c : a) {
        counters[(int) c - 'a']++;
    }
    int i = 0;
    for (int j = 0; j < counters.length; j++) {
        for (int k = 0; k < counters[j]; k++) {
            a[i] = (char) (j + 'a');
            i++;
        }
    }
}

// Big-Oh is O(N).
```

6. Graph Properties

a) unconnected (example: A cannot reach B)

If the graph were undirected, then it would be connected because every vertex would be able to reach every other vertex. (Such a graph is actually called a "weakly connected" graph.)

b) acyclic

c) C has in-degree 3 (B, D, and F have edges that point to C)

d) edge list:

```
[(A,E:2), (B,A:1), (B,C:13), (B,E:6), (D,C:3), (E,F:4), (F,C:8), (F,D:2)]
```

adjacency list:

```
 +---+   +---+
A|   |-->|E:2|
 +---+   +---+   +---+   +---+
B|   |-->|A:1|-->|C:1|-->|E:6|
 +---+   +---+   +---+   +---+
C| / |
 +---+   +---+
D|   |-->|C:3|
 +---+   +---+
E|   |-->|F:4|
 +---+   +---+   +---+
F|   |-->|C:8|-->|D:2|
 +---+   +---+   +---+
```

7. Graph Paths

```
a) DFS:
   B -> A -> E -> F -> D

b) Dijkstra's:
         Visited?    Cost    Previous
     +-------------------------------
   A |      X         0      /
   B |      X        inf     /
   C |      X        11      D
   D |      X         8      F
   E |      X         2      A
   F |      X         6      E

   path from A to C: [A, E, F, D, C], weight 11

c) topological sort:
   B, A, E, F, D, C
```

8. Graph Implementation

```
public static Set<String> popular(Graph<String, String> graph) {
    Set<String> results = new TreeSet<String>();
    for (String v : graph.vertices()) {
        int in = graph.inDegree(v);
        int out = graph.outDegree(v);
        if (in < 2 || in <= out) { continue; }

        int edgeWeightIn = 0;
        for (String v2 : graph.vertices()) {
            if (graph.containsEdge(v2, v)) {
                edgeWeightIn += graph.edgeWeight(v2, v);
            }
        }

        int edgeWeightOut = 0;
        for (String v2 : graph.neighbors(v)) {
            edgeWeightOut += graph.edgeWeight(v, v2);
        }

        if (edgeWeightIn > edgeWeightOut) {
            results.add(v);
        }
    }

    return results;
}
```

9. Parallel and/or Concurrent Programming

Here is an example order of execution for 2 threads that causes a deadlock. The key problem is when two threads make opposite trades, that is, where Thread 1 trades from team A to B, and Thread 2 trades from team B to A. In such a case, certain execution orders cause deadlock. Here is an example:

```
Set<Player> dodgers  = ...;
Set<Player> mariners = ...;

Thread 1: trade("Joey",  mariners, "Dan",   dodgers);
Thread 2: trade("Randy", dodgers,  "Edgar", mariners);
```

```
 1  // Moves player1 from team1 to team2, and moves player2 from team2 to team1.
 2  // If player1 is not on team1, or if player2 is not on team2,
 3  // throws an IllegalArgumentException.
 4
 5  public void trade(Player player1, Set<Player> team1,
 6                    Player player2, Set<Player> team2) {
 7      if (!team1.contains(player1) || !team2.contains(player2)) {
 8          throw new IllegalArgumentException();
 9      }
10      synchronized (team1) {
11          synchronized (team2) {
12              team1.remove(player1);
13              team2.remove(player2);
14              team1.add(player2);
15              team2.add(player1);
16          }
17      }
18  }
```

Here is an execution order that causes deadlock:

- Thread 1 runs lines 1-10. It grabs the lock for its `team1`, which is `mariners`.
- Thread 2 runs lines 1-10. It grabs the lock for its `team1`, which is `dodgers`.
- Thread 1 runs line 11. It tries to grab the lock for its `team2`, which is `dodgers`. This lock is already held by Thread 2, so Thread 1 blocks and waits.
- Thread 2 runs line 11. It tries to grab the lock for its `team2`, which is `mariners`. This lock is already held by Thread 1, so Thread 2 blocks and waits.
- Neither thread will ever release its lock to free up the other thread, so both threads are deadlocked.