Solution to CSE143 Section #8 Problems

1. map: {baz=c, mumble=d, foo=a, bar=b}
   set returned: [a, b, c, d]

   map: {f=z, d=x, e=y, b=y, c=z, a=x}
   set returned: [x, y, z]

   map: {f=2, g=10, d=20, e=1, b=10, c=2, a=1, h=20}
   set returned: [1, 10, 2, 20]

2. One possible solution appears below.

```
public Map<Integer, Integer> counts(List<Integer> list, Set<Integer> set) {
    Map<Integer, Integer> counts = new TreeMap<>();
    for (int value : list) {
        if (set.contains(value)) {
            if (counts.containsKey(value)) {
                counts.put(value, counts.get(value) + 1);
            } else {
                counts.put(value, 1);
            }
        }
    }
    return counts;
}
```

3. One possible solution appears below.

```
public Map<Integer, Set<String>> split(Set<String> words) {
    Map<Integer, Set<String>> buckets = new TreeMap<>();
    for (String word : words) {
        int n = word.length();
        if (!buckets.containsKey(n)) {
            buckets.put(n, new TreeSet<>());
        }
        buckets.get(n).add(word);
    }
    return buckets;
}
```

4. One possible solution appears below.

```
public Map<String, Integer> reverse(Map<Integer, String> map) {
    Map<String, Integer> result = new TreeMap<>();
    for (int key : map.keySet()) {
        String value = map.get(key);
        result.put(value, key);
    }
    return result;
}
```

5. One possible solution appears below.

```
public int maxOccurrences(List<Integer> list) {
     Map<Integer, Integer> counts = new TreeMap<>();
     for (int value : list) {
         if (counts.containsKey(value)) {
             counts.put(value, counts.get(value) + 1);
         } else {
             counts.put(value, 1);
         }
     }
     int max = 0;
     for (int count : counts.values()) {
         max = Math.max(max, count);
     }
     return max;
 }
```

6. One possible solution appears below.

```
public Map<String, Set<String>> convert(Set<String> numbers) {
    Map<String, Set<String>> result = new TreeMap<>();
    for (String s : numbers) {
        String exchange = s.substring(0, 3);
        String digits = s.substring(4);
        if (!result.containsKey(exchange)) {
            result.put(exchange, new TreeSet<>());
        }
        result.get(exchange).add(digits);
    }
    return result;
}
```

7. One possible solution appears below.

```
 public Map<String, Set<List<String>>> acronyms(Set<List<String>> lists) {
     Map<String, Set<List<String>>> result = new TreeMap<>();
     for (List<String> words : lists) {
         String acronym = acronymFor(words);
         if (!result.containsKey(acronym)) {
             result.put(acronym, new HashSet<>());
         }
         result.get(acronym).add(words);
     }
     return result;
 }
```

8. One possible solution appears below.

```
public Map<String, List<Integer>> deepCopy(Map<String, List<Integer>> map) {
    Map<String, List<Integer>> copy = new TreeMap<>();
    for (String key : map.keySet()) {
```

```
            List<Integer> newList = new ArrayList<>();
            for (int n : map.get(key)) {
                newList.add(n);
            }
            copy.put(key, newList);
        }
        return copy;
    }
```

9. Two possible solutions appear below.

```
    public Set<Point> extractEqual(Set<Point> data) {
        Set<Point> result = new HashSet<>();
        for (Point p : data) {
            if (p.getX() == p.getY()) {
                result.add(p);
            }
        }
        return result;
    }

    public Set<Point> extractEqual(Set<Point> data) {
        Set<Point> result = new HashSet<>();
        Iterator<Point> i = data.iterator();
        while (i.hasNext()) {
            Point p = i.next();
            if (p.getX() == p.getY()) {
                result.add(p);
            }
        }
        return result;
    }
```