Recitation 12

Warmup: How to deal with repeated elements in BucketSort and MergeSort?

```
1   /** Merge contents of arrays S1 and S2 into properly sized array S. */
2   public static <K> void merge(K[ ] S1, K[ ] S2, K[ ] S, Comparator<K> comp) {
3     int i = 0, j = 0;
4     while (i + j < S.length) {
5       if (j == S2.length || (i < S1.length && comp.compare(S1[i], S2[j]) < 0))
6         S[i+j] = S1[i++];                  // copy ith element of S1 and increment i
7       else
8         S[i+j] = S2[j++];                  // copy jth element of S2 and increment j
9     }
10  }
```

Code Fragment 12.1: An implementation of the merge operation for a Java array.

```
1   /** Merge-sort contents of array S. */
2   public static <K> void mergeSort(K[ ] S, Comparator<K> comp) {
3     int n = S.length;
4     if (n < 2) return;                                  // array is trivially sorted
5     // divide
6     int mid = n/2;
7     K[ ] S1 = Arrays.copyOfRange(S, 0, mid);            // copy of first half
8     K[ ] S2 = Arrays.copyOfRange(S, mid, n);            // copy of second half
9     // conquer (with recursion)
10    mergeSort(S1, comp);                                // sort copy of first half
11    mergeSort(S2, comp);                                // sort copy of second half
12    // merge results
13    merge(S1, S2, S, comp);                 // merge sorted halves back into original
14  }
```

Code Fragment 12.2: An implementation of the recursive merge-sort algorithm for a Java array (using the merge method defined in Code Fragment 12.1).

R-12.4 Is our array-based implementation of merge-sort given in Section 12.1.2 stable? Explain why or why not.

C-12.47 Given two sets A and B represented as sorted sequences, describe an efficient algorithm for computing $A \oplus B$, which is the set of elements that are in A or B, but not in both.

C-12.35 Suppose we are given an n-element sequence S such that each element in S represents a different vote for president, where each vote is given as an integer representing a particular candidate, yet the integers may be arbitrarily large (even if the number of candidates is not). Design an $O(n \log n)$-time algorithm to see who wins the election S represents, assuming the candidate with the most votes wins.

C-12.37 Consider the voting problem from Exercise C-12.35, but now suppose the integers 1 to k are used to identify $k < n$ candidates. Design an $O(n)$-time algorithm to determine who wins the election.