```cpp
//************************************************************
// Project #1 Practice with Vectors
// Name: Ben Diekhoff
// Data Structures and Algorithms
// Date: 01/28/2019
//************************************************************
// This program takes several numbers as input. The first line determines the
// number of cases to be solved. Each case has 3 lines, one for each friend.
// The first number of each line states how many problems each friend solved,
// and thus how many numbers will be read in for that friend. The following
// numbers are the problems that particular friend solved.
//
// The program reads the relevant data  into 3 vectors, compares the data in
// each vector to the data in another, and only keeps the data that is unique
// to a single friend. Then it checks each vector to make sure that there are
// no duplicate entries (i.e. friend 1 solved problem 3 twice, etc.). Then,
// for each case, it prints the friend who solved the most problems, the
// number of problems he solved, and his unique problems. In the case of a
// tie, it prints this information for every friend who tied.
//
//                  COMPLEXITY
// The most complex function of this program is O(N^2) complexity.
// However, every function is called every time a new test case is read in.
// Since reading in the test cases is O (N), The worst case complexity is
// O (T * (N^2)), where T is the number of test cases.

//************************************************************
/* I have written the enitre program as turned in and have not copied this
code, or parts of this code from the internet or another student.

Signature_____________________________________________*/

//************************************************************

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void loadVects(vector<int> &, vector<int> &, vector<int> &);
void removeDupes(vector<int> &, vector<int> &, vector<int> &);
void printVects(vector<int> &, vector<int> &, vector<int> &);



//************************************************************
// main()
// Parameters: none
// Complexity: O(N)
//
// Reads in the total number of cases and loops that amount of times, calling
// functions to read in the information, process it, and print it out. This
```

```
53  // function has a single for loop, so its complexity is O(N).
54  //*************************************************************
55  int main() {
56      vector<int> f1, f2, f3;  //f1, f2, f3 holds the problems for friends, 1,
57                               // 2, and 3, respectively
58
59      int numCases;            //numCases holds the number of Cases to be read in
60
61      cin >> numCases;
62
63      for (int i = 0; i < numCases; i++) {
64          cout << "Case #" << i + 1 << ":" << endl;
65
66          loadVects(f1, f2, f3);
67          removeDupes(f1, f2, f3);
68          printVects(f1, f2, f3);
69      }
70
71      return 0;
72  }
73
74
75  //*************************************************************
76  // loadVects()
77  // Parameters: 3 vectors of ints, passed by reference.
78  // Complexity: O(N)
79  //
80  // This function reads in the number of problems for a particular friend,
81  // and pushes the problems the friends solved to their own vectors. After all
82  // the vectors are loaded, it sorts the contents of each vector in numerical
83  // order.
84  // The parameters in this function are passed by reference, so they are
85  // modified directly and it doesn't need to return anything.
86  //
87  // This function has multiple for loops that are not nested, so its complexity
88  // is O(N).
89  //*************************************************************
90  void loadVects(vector<int> &v1, vector<int> &v2, vector<int> &v3) {
91
92      int tmp, numProbs;  // numProbs holds the number of problems to read in
93                          // for each friend
94
95      cin >> numProbs;
96      for (int i = 0; i < numProbs; i++) {
97          cin >> tmp;
98          v1.push_back(tmp);
99      }
100
101
102     cin >> numProbs;
103     for (int i = 0; i < numProbs; i++) {
104         cin >> tmp;
```

```
105         v2.push_back(tmp);
106     }
107
108
109     cin >> numProbs;
110     for (int i = 0; i < numProbs; i++) {
111         cin >> tmp;
112         v3.push_back(tmp);
113     }
114
115     // Sorts the vectors
116     sort(v1.begin(), v1.end());
117     sort(v2.begin(), v2.end());
118     sort(v3.begin(), v3.end());
119 }
120
121
122 //*************************************************************
123 // removeDupes()
124 // Parameters: 3 vectors of ints, passed by reference
125 // Complexity: O(N^2)
126 //
127 // This function creates 3 vectors to hold the unique problems each
128 // friend solved. First, it makes sure that the sorted vectors it has been
129 // passed are not holding any duplicate data, for example, if friend 1 has
130 // solved problem 3 five times, it will remove all but one instance of problem
131 // 3. Then, it resizes the vectors based on the distance from the first
132 // element of the vector to the element pointed to by the iterator after using
133 // unique() function.
134 // Next, it compares each element in a vector to the elements in both other
135 // vectors. Any unique elements are pushed to a vector of ints whose purpose
136 // is to hold the unique numbers.
137 // After each friend has all their uniqe numbers stored, it clears the
138 // original vectors that were passed to it by reference. Then it copies the
139 // contents from the vectors that hold the unique numbers to the originals,
140 // and clears the unique vectors.
141 //
142 // The parameters in this function are passed by reference, so they are
143 // modified directly and it doesn't need to return anything.
144 //
145 // The most complex operation in this function compares two vectors. Since
146 // I need to compare EVERY element in a vector against each element in another
147 // vector, I nested the comparison inside a for loop, which raises the
148 // complexity to O (N^2).
149 //*************************************************************
150 void removeDupes(vector<int> &v1, vector<int> &v2, vector<int> &v3) {
151
152     vector<int>u1, u2, u3;      // u1, u2, and u3 hold the unique problems
153                                 // solved by friends 1, 2, and 3, respectively
154
155     /* I use a lot of iterators in this function, the naming format is as
156      follows. itervXvY is an iterator used for comparing the vector vX to
```

```
157       vY. For example, iterv1v3 compares the vector v1 to the vector v3. */
158
159       // Removes any duplicate numbers from each vector
160       auto iterv1v1 = unique(v1.begin(), v1.end());
161       auto iterv2v2 = unique(v2.begin(), v2.end());
162       auto iterv3v3 = unique(v3.begin(), v3.end());
163
164       // Resizes the vectors after duplicate entries have been removed
165       v1.resize(distance(v1.begin(), iterv1v1));
166       v2.resize(distance(v2.begin(), iterv2v2));
167       v3.resize(distance(v3.begin(), iterv3v3));
168
169       //Compares v1 to v2 and v3, pushes any unique numbers to u1
170       for (int i = 0; i < v1.size(); i++) {
171           auto iterv1v2 = find(v2.begin(), v2.end(), v1[i]);
172           auto iterv1v3 = find(v3.begin(), v3.end(), v1[i]);
173           if (iterv1v2 == v2.end() && iterv1v3 == v3.end()) {
174               u1.push_back(v1[i]);
175           }
176       }
177
178       //Compares v2 to v1 and v3, pushes any unique numbers to u2
179       for (int j = 0; j < v2.size(); j++) {
180           auto iterv2v1 = find(v1.begin(), v1.end(), v2[j]);
181           auto iterv2v3 = find(v3.begin(), v3.end(), v2[j]);
182           if (iterv2v1 == v1.end() && iterv2v3 == v3.end()) {
183               u2.push_back(v2[j]);
184           }
185       }
186
187       //Compares v3 to v1 and v2, pushes any unique numbers to u3
188       for (int k = 0; k < v3.size(); k++) {
189           auto iterv3v1 = find(v1.begin(), v1.end(), v3[k]);
190           auto iterv3v2 = find(v2.begin(), v2.end(), v3[k]);
191           if (iterv3v1 == v1.end() && iterv3v2 == v2.end()) {
192               u3.push_back(v3[k]);
193           }
194       }
195
196       // Clears the original vectors
197       v1.clear();
198       v2.clear();
199       v3.clear();
200
201       // Copies the unique numbers into the original vectors
202       v1 = u1;
203       v2 = u2;
204       v3 = u3;
205
206       // Clears the unique vectors
207       u1.clear();
208       u2.clear();
```

```
209     u3.clear();
210 }
211
212
213 //*****************************************************************
214 // printVects()
215 // Parameters: 3 vectors of ints, passed by reference.
216 // Complexity: O(N)
217 //
218 // This function determines which of the processed vectors is the largest,
219 // and prints out the number of the friend with the largest vector, the
220 // number of problems he solved, and the unique problems he solved.
221 //
222 // The parameters in this function are passed by reference, so they are
223 // modified directly and it doesn't need to return anything.
224 //
225 // This function has multiple for loops that are not nested, so its complexity
226 // is O(N).
227 //*****************************************************************
228 void printVects(vector<int> &v1, vector<int> &v2, vector<int> &v3) {
229
230     // Friend 1 is the winner
231     if (v1.size() > v2.size() && v1.size() > v3.size()) {
232         cout << "1 " << v1.size() << " ";
233         for (int i : v1) {
234             cout << i << " ";
235         }
236     }
237     // Friend 2 is the winner
238     else if (v2.size() > v1.size() && v2.size() > v3.size()) {
239         cout << "2 " << v2.size() << " ";
240         for (int i : v2) {
241             cout << i << " ";
242         }
243     }
244     // Friend 3 is the winner
245     else if (v3.size() > v1.size() && v3.size() > v2.size()) {
246         cout << "3 " << v3.size() << " ";
247         for (int i : v3) {
248             cout << i << " ";
249         }
250     }
251     // Friends 1, 2, and 3 are tied.
252     else if (v1.size() == v2.size() && v1.size() == v3.size()
253         && v2.size() == v3.size()) {
254         cout << "1 " << v1.size() << " ";
255         for (int i : v1)
256             cout << i << " ";
257         cout << endl;
258         cout << "2 " << v2.size() << " ";
259         for (int j : v2)
260             cout << j << " ";
```

```
261         cout << endl;
262         cout << "3 " << v3.size() << " ";
263         for (int k : v3)
264             cout << k << " ";
265     }
266     // Friends 1 and 2 are tied.
267     else if (v1.size() == v2.size()) {
268         cout << "1 " << v1.size() << " ";
269         for (int i : v1)
270             cout << i << " ";
271         cout << endl;
272         cout << "2 " << v2.size() << " ";
273         for (int j : v2)
274             cout << j << " ";
275     }
276     // Friends 1 and 3 are tied.
277     else if (v1.size() == v3.size()) {
278         cout << "1 " << v1.size() << " ";
279         for (int i : v1)
280             cout << i << " ";
281         cout << endl;
282         cout << "3 " << v3.size() << " ";
283         for (int j : v3)
284             cout << j << " ";
285     }
286     //Friends 2 and 3 are tied.
287     else if (v2.size() == v3.size()) {
288         cout << "2 " << v2.size() << " ";
289         for (int i : v2)
290             cout << i << " ";
291         cout << endl;
292         cout << "3 " << v3.size() << " ";
293         for (int j : v3)
294             cout << j << " ";
295     }
296 
297     cout << endl;
298 
299     // Clears the vectors to prepare for the next case
300     v1.clear();
301     v2.clear();
302     v3.clear();
303 }
304 
```