CS 1301 - Homework 09

Due before October 24, 3 *pm* on Vista (WebCT). You can resubmit your work up until the deadline.

1. **Winner.java** – Consider a situation where team 1 and team 2 compete against one another in basketball games. You have an array of team 1 scores and a corresponding array of team 2 scores. You want to write a method, ***getWinners*** that will return an array of indices of the games that team 1 won by *x1* to *x2* points, where the two arrays, *x1* and *x2* are input. For example,

```
int[] team1 = {95, 90, 104, 85, 90, 110, 105, 88}
int[] team2 = {90, 97, 102, 77, 100, 90, 109, 80}
int lower = 5, upper = 10

int[] winnerIndices = getWinners( team1, team2, lower, upper )
```

   Then, *winnerIndices* will contain the values: 0, 3, 7

   Write a main that uses array initializers to test your program. Use the result of the method to print out a nice message that shows the scores of the games that team 1 won by *x1* to *x2* points.

2. **DieRoll.java** – You will write several methods that accept an array of integers which represent the faces on die rolls (six-sided die). For instance, the input array may be: 3, 4, 2, 1, 4, 3, 6, 2, 5, 1, 4, 6, 3, 3, 2, 6, 1, 5, 3, 1.

   getFrequencies() – This method will return an array with 6 elements, where each element corresponds to the percentage of times a face occurs. For instance, with the input array above, this method will return: 20.0, 15.0, 25.0, 15.0, 10.0, 15.0. Thus, the first element has value 20.0 because there were four 1's in 20 rolls.

   getLongestSequence() – This method will return an array with two elements in it. The first element will be the length of the longest sequence of consecutive faces which are the same, and the second element will be the corresponding face. For instance, using the input above, this method will return: 2, 3 because the longest sequence was two 3's. Your program should return the first, longest sequence. In other words, if the input is: 3, 4, 4, 1, 5, 6, 6, 2, your method should return: 2, 4 (This is to keep things a bit simpler).

   Write a main that uses array initializers to test your methods. Display the output with descriptive text. For instance, the *getFrequencies* method output should be displayed in a nice table. Similar for the other two. Your output should also show the input array(s) you have used (just list the values, e.g. die rolls: 2, 3, 1, ...)

   EXTRA CREDIT

   getLongestSequences() – Similar to the getLongestSequence method, this method will return an array of the longest length and faces of all the longest sequences. The first element is the length (count), and the subsequent elements correspond to the faces. For example:

Input	Output
2, 2, 2, 3, 1, 4, 5, 5, 5, 6, 4, 4, 4, 2	3, 2, 5, 4
1, 2, 3, 4, 5, 6	1, 1, 2, 3, 4, 5, 6

3. **BinarySearch.java** – You will write several methods and they **must** utilize the binary search algorithm:

   binarySearchBlock() – Which takes an array of doubles, *source* (assume sorted) as input, as well as a *key*, and two ints: *beg* and *end.* Your method will search the array for *key* between the elements with indices *beg* and *end*. The return value should be the usual return from binary search. In addition, the method will return -1 if *beg* and *end* are not valid in any way.

   binarySearch() – Which takes an array of doubles, *source* as input as well as a *key*. Assume that the length of *source* is a multiple of 10 and is sorted in blocks of 10. For instance, *source* may look like this:

   4, 6, 9, 22, 34, 35, 59, 62, 77, 84, 1, 6, 45, 47, 48, 59, 64, 66, 82, 97, ...

   Your method should do a binary search on the array, using consecutive blocks of 10. It should return an array of indices of the result of the binary search on each block of 10. For instance, if *key=45* in the example above, your method will return: -7, 12, ... Your method should utilize the *binarySearchBlock* method. (Note, we don't get the most bang from using binary search on 10 items, so pretend like it is blocks of 1000 or larger.)

4. **GCD.java** – You will write a method **gcd() w**hich takes an array of integers as input and returns the greatest common divisor of the numbers. Write a *main* that prompts the user for integers and then computes and displays the GCD. Hint: see section 4.8.