EECS 233 Homework 8 (Fixed typo in 1(b). Clarification added for #5 on Friday, November 3.)

General requirements:

- Due at 11:00 PM on the posted due date.
- Include your name and network ID as a comment at the top of all of your programs.
- Create a typed document (.docx or .pdf) for the report with your name and network ID at the top.
- Upload your document and all .java files as a .zip file to Canvas. Do not use other formats such as .rar.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus. File sharing is prohibited.

WARNING: There are various implementations of binary search available on the Internet. However, students are required to use only the techniques that have been presented in class and in the following instructions. Additionally, students will be tested on their understanding of these particular techniques. Developing the programs on your own will maximize your understanding.

1. Revise BinarySearcher.java to include the following methods. **Provide example output in your report.**
   a. Add a method that performs a binary search using a loop instead of recursion. It should receive only the array and the target. It should return the same values as the recursive version does. Hint: instead of recursive calls, just change the values of "first" and "size". Test it in the main method and print the result. For example:

```
Found at index [9].
```

   b. Add a new version of the recursive binary search that, before each recursive call, prints the middle value and whether the next call will be for values *before* or *after* the middle value. Test it in the main method and print the result. For example, suppose the array has values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and we search for 10. There will be two recursive calls: after value 8 and before value 12. The output would be like the following:

```
Searching after 8...
Searching before 12...
Found at index [9].
```

2. Binary search requires that the programmer passes a sorted array. Otherwise, it may give an incorrect result. Answer the following. No programming is required.
   a. Make up an unsorted array of at least 6 values where binary search would accidentally find the target.
   b. Suppose we added a loop to the beginning of the binary search algorithm to verify that the array is sorted. What would the Big-O run time be for the entire algorithm (verification + search)? Explain your answer in 1 – 3 sentences.

3. Assume a hash table has a capacity of 13. Complete the analyses below to show the calculation and the complete array after inserting each key value, in the order given. An example is given for the key value 1 (shown in **bold**). Use the following two hash methods:
   a. Linear probing with hash function *h*(*key*) = *key* % 13.

Key	Calculation	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
1	***1 % 13 = 1***		***1***											
2														
12														
13														
14														
130														
1212														
1301														
1300														

b. Double hashing with primary hash function $h_1(key) = key \% 13$ and secondary hash function $h_2(key) = key \% 11$.

Key	Calculation	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
1	***1 % 13 = 1***		***1***											
2														
12														
13														
14														
130														
1212														
1301														

4. Modify Table.java to have the following features.
   a. Modify the findIndex() method to print each index that is attempted. For example, suppose the keys array contains {null, 1, null, null} and we use put(5, "Hi"). The first hash value would be 5 % 4 = 1, which has a collision. The next index would be 1 + 1 = 2, and the output would look like this:

```
Trying index [1]
Trying index [2]
```

   b. Add a "print" method that prints the indexes and key values in a table. The exact formatting does <u>not</u> matter (can be vertical, columns do <u>not</u> need to be aligned, etc.). Below is an example using print() with an array capacity of 4 and using put(1, "Hi") and put(5, "Bye"). Note that printing the "data" values is not required.

```
[0]   [1]  [2]  [3]
null  1    5    null
```

   c. Add a "main" method that demonstrates (a) and (b). **Provide example output in your report.**

5. Make a new version of your program from problem #4 above that uses <u>double hashing</u>. For the 2nd hash function, you may hardcode the divisor (e.g. 11), or you may use (data.length – 2). You are allowed to modify the Table class as necessary by adding new methods, passing additional arguments, etc.

6. Suppose we insert *N* unique keys, in a random order, into a hash table with <u>no</u> collisions. Answer the following, explaining in 1 – 3 sentences:
   a. What is the Big-O run time of the <u>total</u> process of adding all of the keys?
   b. What is the Big-O run time of finding a <u>single</u> key?

Rubric is forthcoming...

Item	**Points**
General programming requirements	5
1(a) General loop design	5
1(a) Computing values for first and size	5
1(b) Adding correct output	5
2(a) Array values	5
2(b) Answer with explanation	5
3(a) 2 points for each insertion (8 total)	16
3(b) 2 points for each insertion (8 total)	16
4(a) Printing correct values	5
4(b) Printing indexes and array	5

4(c) Coding "main" and reporting output	5
5. 2nd hash function	5
5. Using 2nd hash function correctly	8
6(a) Answer with explanation	5
6(b) Answer with explanation	5
Total	100