Revised September 14 at 5 PM after mistakes were discovered for problem #2 (during the lecture!). The main changes for #2 are: (1) don't worry about whether "add" is for the end or not, (2) no removing from the front/beginning. See specific changes below in red.

EECS 233 Homework 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.

1. For the following methods from the Permutation class in HW #2, give the Big-O run time complexity and explain your answer in 1 – 3 sentences. Note that *N* is the number of persons. Be sure to use a correct solution for HW #2 (such as the posted solution).
   a. Permutation()
   b. getTrait()
   c. getPerson()
   d. notEqual()

2. Analyze IntArrayBag.java and IntLinkedBag.java, which are described in the textbook, for the operations listed below. Demonstration programs are provided and were shown in class (see BagDemonstration.java and ListDemonstration.java). For each operation, choose any *N* value such that the time is greater than 0 msec, but use the same *N* value for IntArrayBag and IntLinkedBag so they can be compared. In your output, display the *N* value and the run time. Note that IntLinkedBag.java requires IntNode.java.

   a. Add the values 1, 2, 3..., *N* to the data structure using the "add" method. It's OK that each data structure does this differently. (IntArrayBag adds at the end, and IntLinkedBag adds at the front.)

   b. Remove each value from the end of the data structure until all of the values are removed. You will only analyze the run time for the "remove" operations - not the "add" from part (a). For IntArrayBag, you can simply remove values in reverse order: *N*, ...3, 2, 1. For the IntLinkedBag, however, you will need to be careful with the order of the removals because of how values are added and removed. To ensure that you always remove from the end, you must do this: (1) remove 1 because the linked list is built in reverse order, and (2) remove *N*, ...3, 2 to remove the remaining values. The following table shows an example using the values 1, 2, 3, 4:

Command	Ordered Values in IntLinkedBag
add(1)	1
add(2)	2 1
add(3)	3 2 1
add(4)	4 3 2 1
remove(1)	3 2 4
remove(4)	2 3
remove(3)	2
remove(2)	*(empty)*

   As explained in the textbook for the "remove" method (page 223), "data" at end node is replaced with "data" at front, and the "head" is moved to point at the node that was previously #2 in the linked list.

   c. Add the values 1, 2, 3..., *N*. Then use countOccurrences() to count the quantities of all the values. Note: you will only analyze the counting operation, but you must add the values first.

d. Add the value 0 a total of *N* times (size *N*) to the data structure. It's OK that each data structure does this differently.

Include all of your output in your report. Below is an example of how your output might look for operation (a) using one of the data structures:

```
Add to end: N = 10000000, t = 0.116 sec
```

3. For the IntLinkedBag class, the Big-O run time complexity should be the same as similar operations for the IntArrayBag class. For the following methods, state the expected Big-O time complexity, and explain in 1 – 3 sentences why it should be the same as the time complexity for the IntArrayBag class.
   a. add()
   b. remove()
   c. countOccurrences()

4. For the experiments (a) – (d) in #2 above, answer the following:
   a. Are the run times for O(1) operations consistently shorter, as compared to O(*N*) operations? Describe your results in 1 – 3 sentences.
   b. Are the run times for IntArrayBag different from those of the IntLinkedBag class? Describe your results in 1 – 3 sentences.

   Notes: You are not required to explain the reason for your results. You will only be graded on your ability to describe what happened.

Tip (compiling classes): IntNode.java, IntArrayBag.java, and IntLinkedBag.java must all be compiled separately if you use those classes in other programs.

Tip (how many programs for #2?): Use as many (or as few) program files as you wish. One example is to make one program with a main() method to analyze one data structure, and make another program for the other data structure. Alternatively, you could have one program for both. Note that IntNode.java, IntArrayBag.java, and IntLinkedBag.java do not have main() methods, so you can instantiate those classes in your analysis programs (like BagDemonstration.java from the textbook.

Tip (empty arrays vs. full arrays): Be careful to start with an empty array each time you start a new "add" test. To do this, you can just use "x = new ..." to call the constructor again. You shouldn't be adding to a full array from the previous test. In contrast, tests for removing and counting require a full array to begin with. However, you should only test the run time for the "remove" or counting operations. For "remove" and counting, do not include the time required for creating the array.

Tip (is any of this in the textbook?): IntArrayBag.java and IntLinkedBag.java are both discussed in detail in the textbook. Only IntArrayBag.java is analyzed regarding Big-O run time complexity, as was discussed in lecture.

Tip (are we expected to memorize all this front/end add/remove stuff?): No, but it should make sense if you see the Java code. The goal is to look at O(1) and O(*N*) operations. The drastic differences between IntArrayBag and IntLinkedBag make it challenging to use them in the same way.

Rubric is forthcoming...