Computer Science 112 - Fundamentals of Programming II
Lab Project 4
Due on Github: 11:59 PM Monday 11 Octrober

In this project, you will complete three implementations of the bag collection called ArrayBag , LinkedBag , and ArraySortedBag . Partial implementations of ArrayBag , LinkedBag , and a tester program for all bag implementations can be found in the lab4.zip.

Here's how to complete the current lab:

1. Unzip the Lab4 folder to your account and put your name in each file, as usual. Open the file **baginterface.py** and review the interface for the bag collections. Note that this code can be loaded into the shell, but the methods don't actually do anything.

2. Open the file **testbag.py** and examine the code for testing a bag type. Note that the test function creates some bag objects and exercises the methods on them. Note that this function can be called with any bag type as an argument. Now press F5 to run the program and note the output. Everything goes as expected until the end, when the bag's array runs out of room to accommodate items.

3. Now open the file **arraybag.py** and locate the add method. Here is the code that caused the exception. Note that the array needs to be resized if there is not room to add an item. Complete the code for that procedure now and run the **testbag.py** program again. At this point the test program shouldn't crash, but will report an incorrect test. You will handle that incorrect test in the last exercise (8) below.

4. The code for the method **`remove`** in **ArrayBag** runs correctly, but leaves wasted space when the array becomes 3⁄4 empty. Find the spot where this problem can be fixed and complete the code to resize the array when necessary. (Caution: the size of the array should not go below its default capacity; otherwise, its size should be shrunk by half if the number of items is less than or equal to 1⁄4 of the array's current size.) Run both the test function and the **testResize** function to verify that the remove method is working as expected; i.e., you should see the length of the array decreasing in the report.

5. Test your **LinkedBag** class with the same tester program. You should be able to run test with **LinkedBag** as an argument. Note that the clear method does not work as expected. However, the program does not crash when all those items are added at the end. Your job in this exercise is to complete the code for the clear method and test the program again.

6. A sorted bag behaves just like a regular bag, but allows the user to visit its items in ascending order with a for loop. Therefore, the items added to this type of bag must have a natural ordering and recognize the comparison operators. Some examples of such items are strings and integers.

Define a new class named **ArraySortedBag** that supports this capability. Like **ArrayBag** , this new class is array-based, but its **`in`** operation can now run in logarithmic time. To accomplish this, **ArraySortedBag** must place new items into its array in sorted order. The most efficient way to do this is to modify the **`add`** method to insert new items in their proper places. You will also have to include a __**`contains`**__ method, to implement the new, more efficient search. *Hints*: (1) copy **arraysbag.py** to a new file named **arraysortedbag.py** and make your changes in that file. The most obvious, immediate change you should make in your new file is to search-and replace **ArrayBag** with **ArraySortedBag**. (2) Before modifying the **`add`** method, use a pencil-and-paper or text editor to work out some examples of inserting an item into an already-sorted list. This will help you devise an algorithm; (3) A nice implementation of binary search is already in the lecture notes; feel free to use it!

You can test **ArraySortedBag** with the test function in **testbag.py**. Just import your new class and run the test function with it. Be sure to randomize data that are added to a sorted bag so you can verify that it's behaving correctly.

7. Add a method named **`count`** to the **BagInterface** . This method expects an item as an argument and returns the number of instances of this item in the bag. Then include and test this method in each of your bag implementations.

8. Think about the logic of the **`__eq__`** method, which compares a bag and another object for equality. If the other object is the same type of bag and its length is the same as self , the method returns True if all of the objects in self are also in the other bag. While this would be a satisfactory equality test for sets, is it so for bags? For example, suppose bag A contains three 1s and two 2s, and bag B contains two 1s and three 2s. These two bags will pass the current equality test, but are they really equal? Revise this method to handle all of these false positives. (Hint: you can use a method that you recently added to help solve this problem.)