CS 143 Assignment 1
Begin at the Beginning
DUE on Friday, January 18 at 11:59 PM

In class, we implemented an `IntArrayList` class which `int`ernally had an array of `int`egers but made it easy to add new elements to the end. We saw that after improvements, adding n elements to an `IntArrayList` would only take $O(n)$ time. It accomplished this by doubling the underlying size of the array and copying the old array every time it ran out of space. If n total elements are added to the array, in the worst case, the array might have run out of space and had to copy itself when the very last (nth) element was added, when the $\frac{n}{2}$ nd element was added, $\frac{n}{4}$th, $\frac{n}{8}$th, and so on down to the starting size of the array. Every time the array runs out of space, it needs to copy all the elements from the old array to the new array. However, even as n becomes very large and this series becomes arbitrarily long, $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \cdots \leq 2n$. In other words, the total number of elements we need to copy over the lifetime of an `IntArrayList` is bounded above by a **linear function** of the total number of elements added to it. (Allocating the array each time we copy is additional work which depends on the memory allocation algorithm used, but it will be efficient enough that it won't change the linear bound.)

Your task: Improve `IntArrayList` by creating an `addBefore` method. Instead of adding an element to the end, `addBefore` adds it to the beginning, **at index 0, causing all existing elements to move forward (their indexes increased by 1)**. However, like `add`, your `addBefore` method must also guarantee that only $O(n)$ time is needed to perform n addBefores and adds. To accomplish this, **as with `add`, most calls to `addBefore` must execute very quickly, in O(1) constant time**. The changes you make to `IntArrayList` should preserve the property that the `get` and `set` methods take a constant amount of time (not proportional to the array size) and the above property of `add` that only a linear number of elements is copied.

There are several ways to accomplish this task. Perhaps the easiest: double the array when you hit the beginning, but instead of copying the old array to the beginning of the new array, copy it to the end (leaving all the unused spaces that are still at the end). **You will also need to change the implementation of some other methods (because elements will be indexed differently with this enhancement) and add an additional instance field (to track extra information). The existing methods should continue to have their existing O runtimes as seen in the comment.**

Start with the `IntArrayList` implementation found on Canvas, which will resemble what we did in class. It will have an empty `addBefore` method which you will need to fill in. A tester program will be released which will attempt to empirically test that your program takes only $O(n)$ time to perform n addBefores and adds (after testing that it seems to work, along with verifying that the other methods seem to work). For reference, on the next page is a program listing for `IntArrayList`.

EXTRA CREDIT 3 POINTS: Notice that the above technique will "run out of space" when there is extra room left on the opposite end of the array. Instead, utilize the whole array before triggering any copying (preserving performance as above).

```
public class IntArrayList {
   private int[] a; // Underlying array
   private int length; // Number of added elements in a

   public IntArrayList() {
      length = 0; // Start with no added elements in a
      a = new int[4]; // A little room to grow
   }

   public int get(int i) { // Retrieve an added element, O(1)
      if (i < 0 || i >= length) {
         throw new ArrayIndexOutOfBoundsException(i);
      }
      return a[i];
   }

   public int size() { // Number of added elements, O(1)
      return length; // The number of added elements
   }

   public void set(int i, int x) { // Modify an added element, O(1)
      if (i < 0 || i >= length) {
         throw new ArrayIndexOutOfBoundsException(i);
      }
      a[i] = x;
   }

   public void add(int x) { // Add an element to the end, O(n) for n
      if (length >= a.length) {
         // Create new array of double the length
         int[] b = new int[a.length * 2];
         // Copy the elements of a to the corresponding indexes of b
         for (int i = 0; i < a.length; i++) {
            b[i] = a[i];
         }
         // Reassign a reference to b
         a = b;
      }
      // Place x at the end of the IntArrayList
      a[length] = x;
      // Increase length by 1
      length = length + 1;
   }
   public void addBefore(int x) { /* FILL THIS IN!! */
   }
}
```