## Binary Search – Some simple Performance measurements

**The goal for this exercise** is to measure the performance of binary search. The longer-term goal is to compare this performance to other, alternative algorithms (such as linear search) so that will have a clear understanding of why the Big O notation is useful.

Now do the same sort of thing for your Binary Search that you did for your linear search – copy your copy your **FindIntegerBinary** function, and rename the copy to be **FindIntegerBinaryPerfMeasured**. Add to it an **out** (reference) parameter – an integer named **numComparisons**. Within your **FindIntegerBinaryPerfMeasured**, you should initialize (setup) the value of this parameter to be zero. Each time you compare the target to any element of the array, you should increase the parameter’s value by one.

So if you were to search an array for something, and find it in the first slot that it checks, **numComparisons** should be 1 when the method returns. If you have an array of 20 elements, and you don't find it anywhere, **numComparisons** should be about 5 (i.e., 4 or 5, or 6, depending on exactly how you implement this) when the method returns. More precisely, numComparisons should go up by 1 each time you compare the target value (that you’re looking for) to something that’s in the array.

Note that **FindIntegerBinaryPerfMeasured** does NOT have to be recursive. You’re welcomed to make it recursive, but you are neither required nor expected to do so.

Once you’ve done that, you should test the function (once you've written it), by adding whatever you need to add to Main() in order to be confident that your code works correctly.

**What you need to do for this exercise:**

1. Implement the **FindIntegerBinaryPerfMeasured** method, within the SearchingAndSorting class.
	1. Note that you don't (technically) need to complete the "Binary Search" exercise in this same lesson – you can jump straight to this exercise. However, many people find it easier to do that exercise first, then copy-and-paste that code into this exercise.
2. The following is NOT required (which is why it’s crossed out).
I may want to add it back in a future quarter, though, which is why it’s still here. ~~Once you’ve done this, all the tests in the NUnitTests\_FindIntegerBinary\_Measured class should pass. This class is located in the starter solution, in the~~ **~~PCE\_ForTests~~** ~~project, in the file named~~ **~~PCE\_Test.cs.~~**
	1. ~~You may need to set another project as the start up project (such as the PCE\_Test\_Runner project), by right-clicking on that other project, and selecting the “Set As Startup Project” menu option. Within the test runner project, you may (or may not) need to change the code in the RunTests.cs file, particularly at the top of the Main method.~~
	2. ~~Keep an eye on the results of the ‘EqualsFuzzy’ test – if the test is failing because you’re getting a wildly different number of comparisons, and yet you think that the number of comparisons is correct, then talk to the instructor, and/or post a question to the Google Group.~~