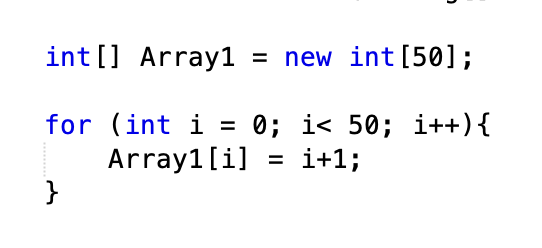
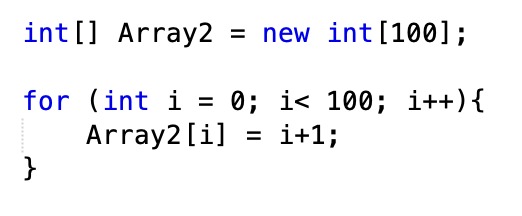
COM 210 Assignment 2 - Joseph Giglio and Darius Freeman

For this lab, we created three sorted arrays. One array with fifty values, one array with 100 values, and one array with 1000 values. On each array we ran a sorted fetch, insert, and delete algorithm, as well as an unsorted fetch, insert, and delete algorithm. This lab helps us understand the importance of Big O and how it is important in execution time.

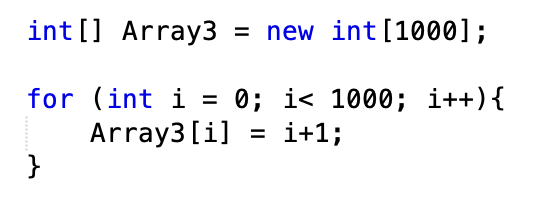
Code to make sorted array of 50 items:



Code to make sorted array of 100 items:



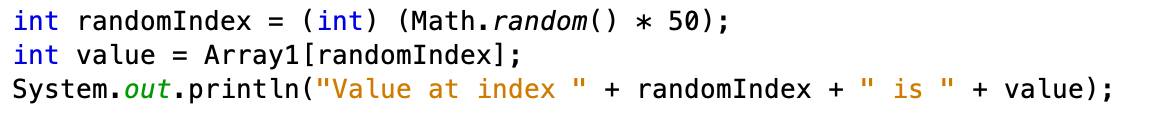
Code to make array of 1000 items:



As you can see, the code for creating each array is very similar. The value in the brackets after “new int” was just changed to the number of items we want in each array. Then, we made a simple for loop after, adding every element to the array. Just like in the first line of code, we just changed the value after “i<” to change the number of values being added to each array.

In order to save time, we just ran the unsorted and sorted version of each algorithm on the sorted array. Even though an array is sorted, you can still run an unsorted algorithm on it.

Code for Unsorted Fetch:



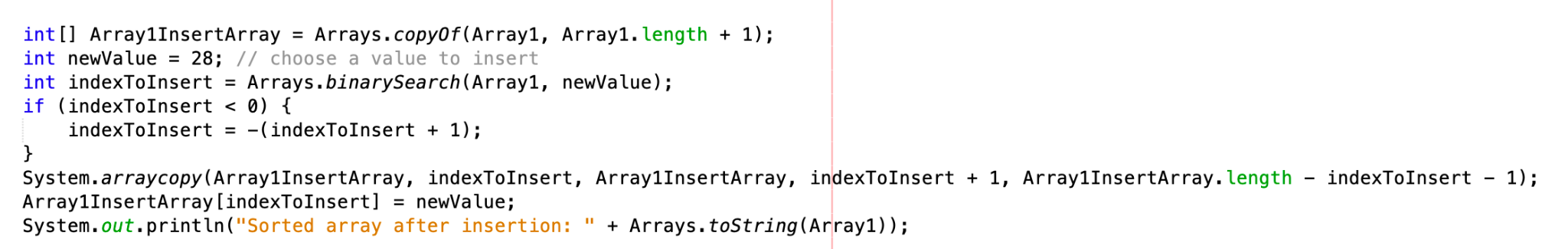
This code basically takes a random integer value in the array and fetches it. It then outputs the value and the index in a random location.

Code for Sorted Fetch:



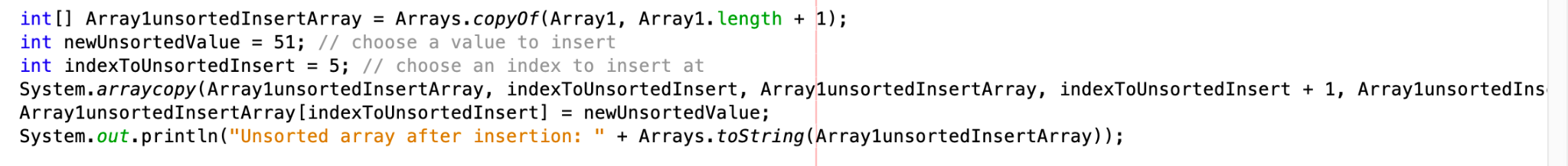
For the sorted fetch, you choose the index where you want the value to be fetched from. Then it outputs the value at that specific index.

Code for Sorted Insert:



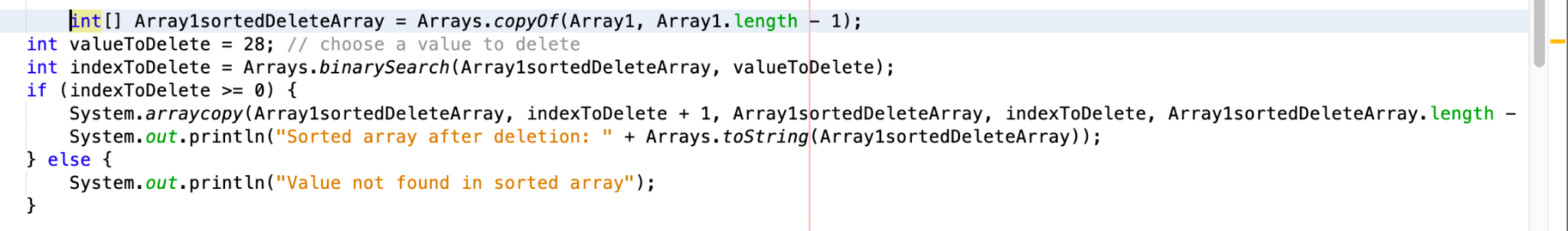
The sorted insert chooses an index where you want the value to be inserted. It then puts the value inside the array at that specific index.

Code for Unsorted Insert:



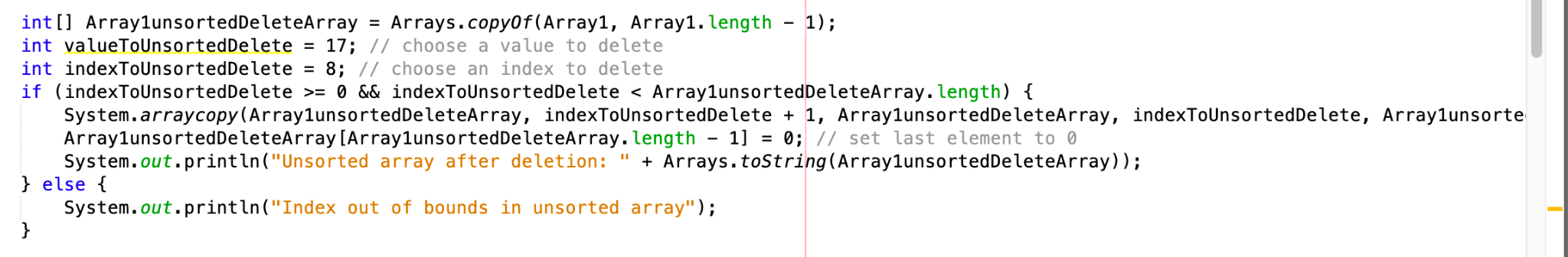
The unsorted insert does the same thing as a sorted insert, except that this time it chooses a random index to insert the new value into.

Code for Sorted Delete:



The code for the sorted delete goes through every single value until it finds the value that it wants to delete. Once it is found, it is deleted from an array. Basically, a fetch and an insert are both performed.

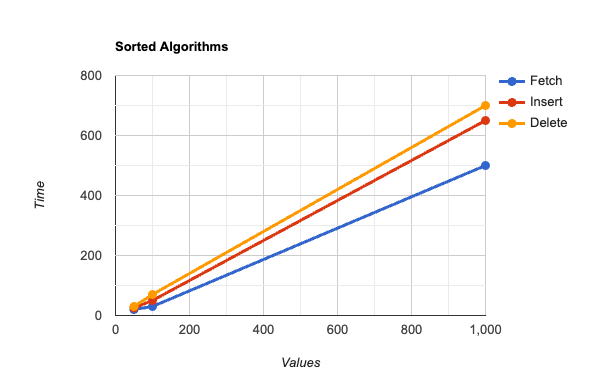
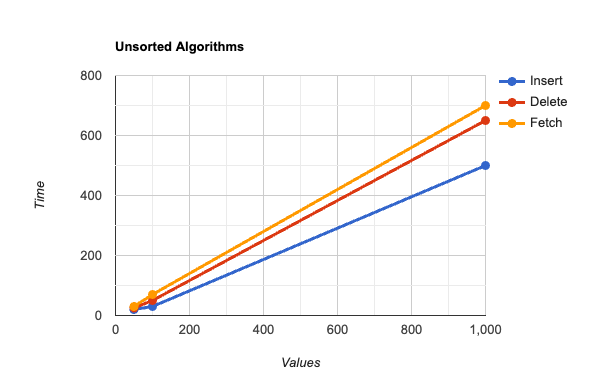
Code for Unsorted Delete:



The code for the unsorted delete is basically the same, except it randomly fetches a value to delete and then deletes it from an array.

**The code for the different types of algorithms for the arrays of 100 and 1000 is exactly the same, except you just change the names of your variables in the code.**

**Graphs Demonstrating Time Complexity for Sorted vs Unsorted:**

****

**Importance of Big-O:**

When approaching Big-O from a theoretical perspective compared to an experimental perspective, they can be the same, but also very different. When you look at something from a theoretical perspective, you can base it off of common knowledge. When it comes to Big-O from a theoretical perspective, it can be based off knowledge, or it can be based off of mathematics performed on paper. When given an algorithm, you can do math on paper to get an estimate of what Big-O can possibly be. This is very good if you are looking to get an idea of what Big-O could be. However, when it comes to Big-O from an experimental perspective, it can be determined by actually experimenting and running your algorithm on a machine. This can give you a better estimate of the value of Big-O is, since it was actually tested on a machine. Also, we think that it is better to use the experimental perspective since it gives you a more accurate time. Overall, getting the value of Big-O is very important to understand the execution time of an algorithm.