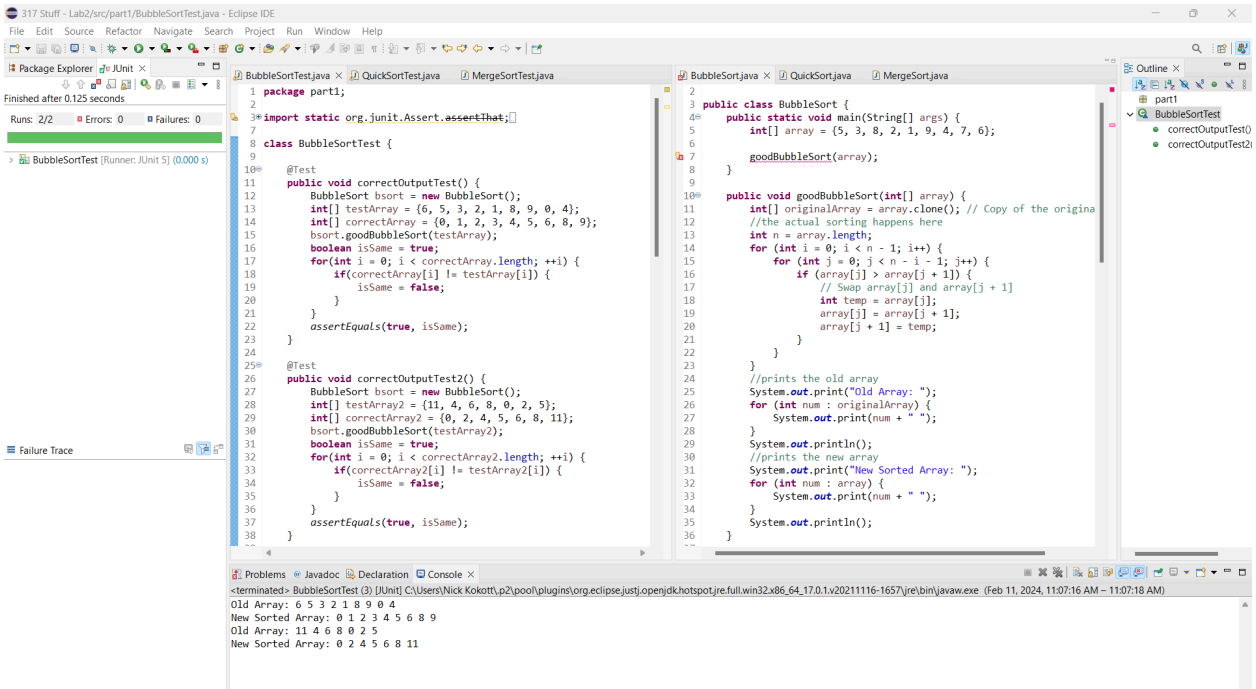
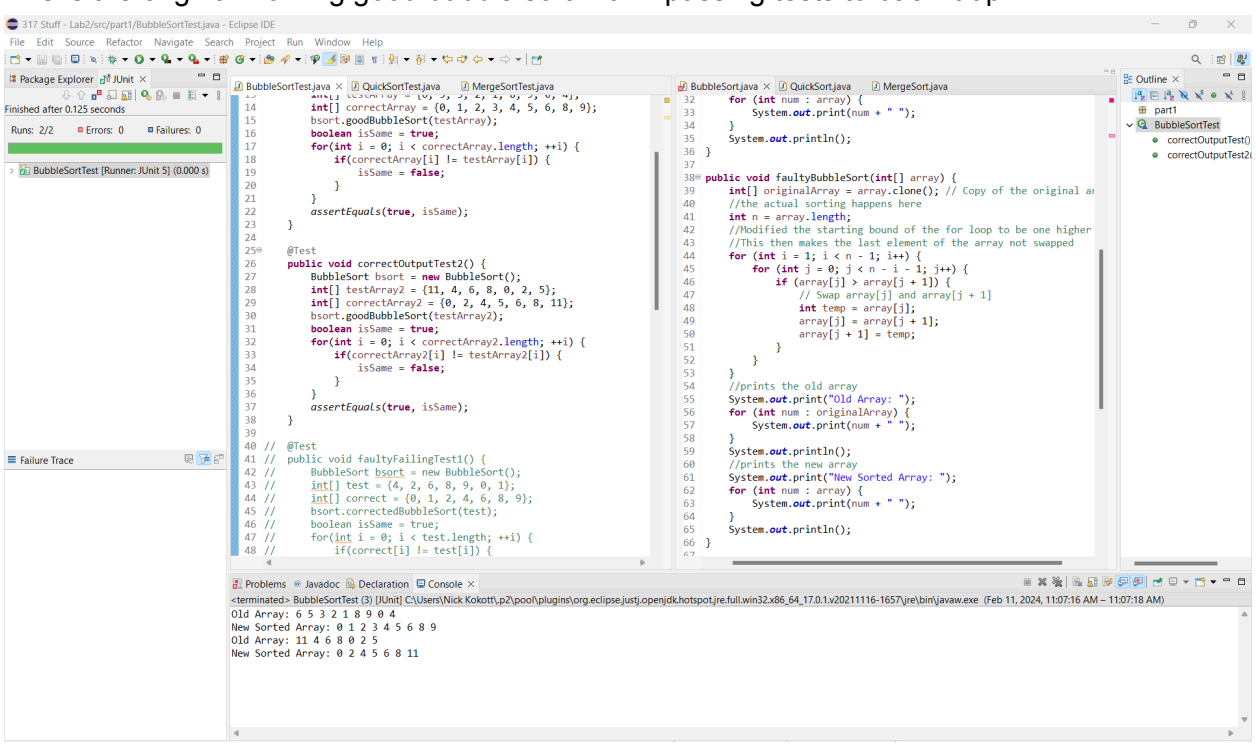
Nicholas Kokott - kokottni

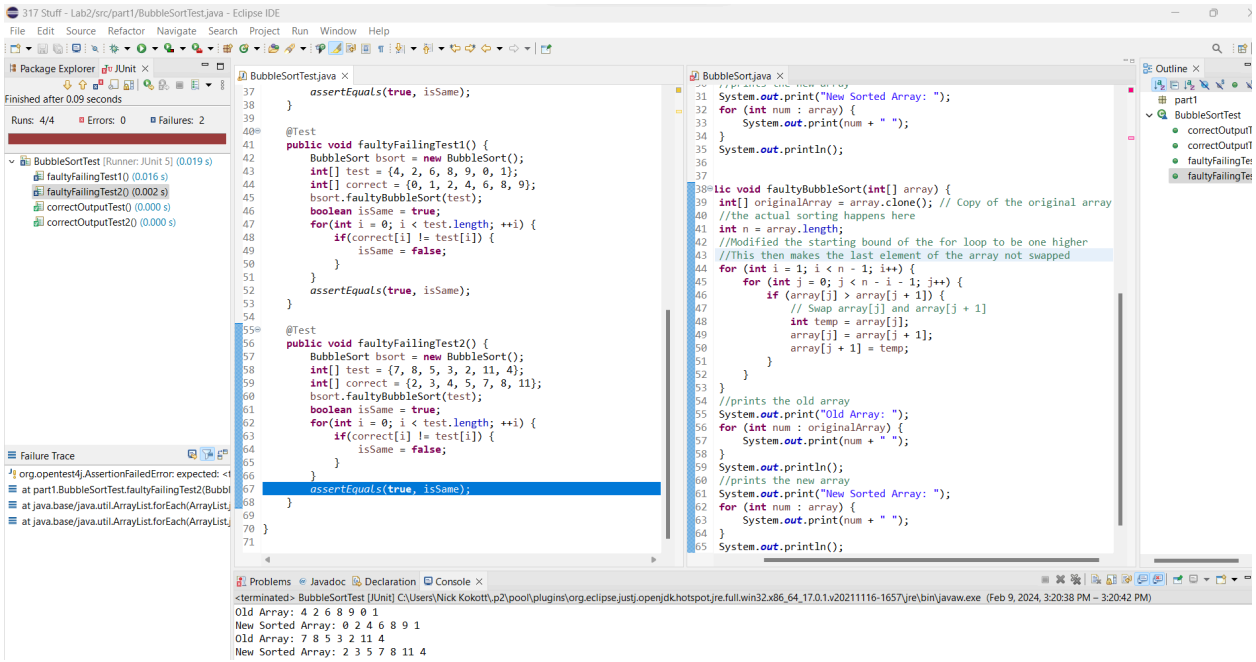
Bubble Sort pictures



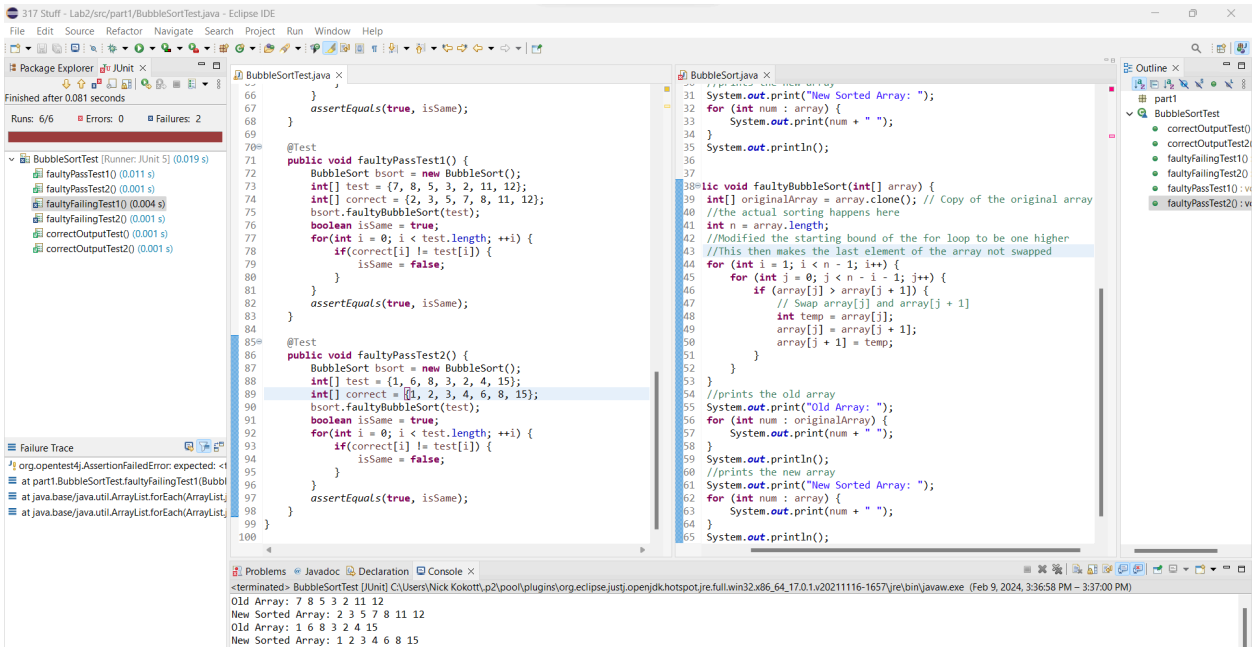
This is the original working good bubble sort with 2 passing tests to back it up



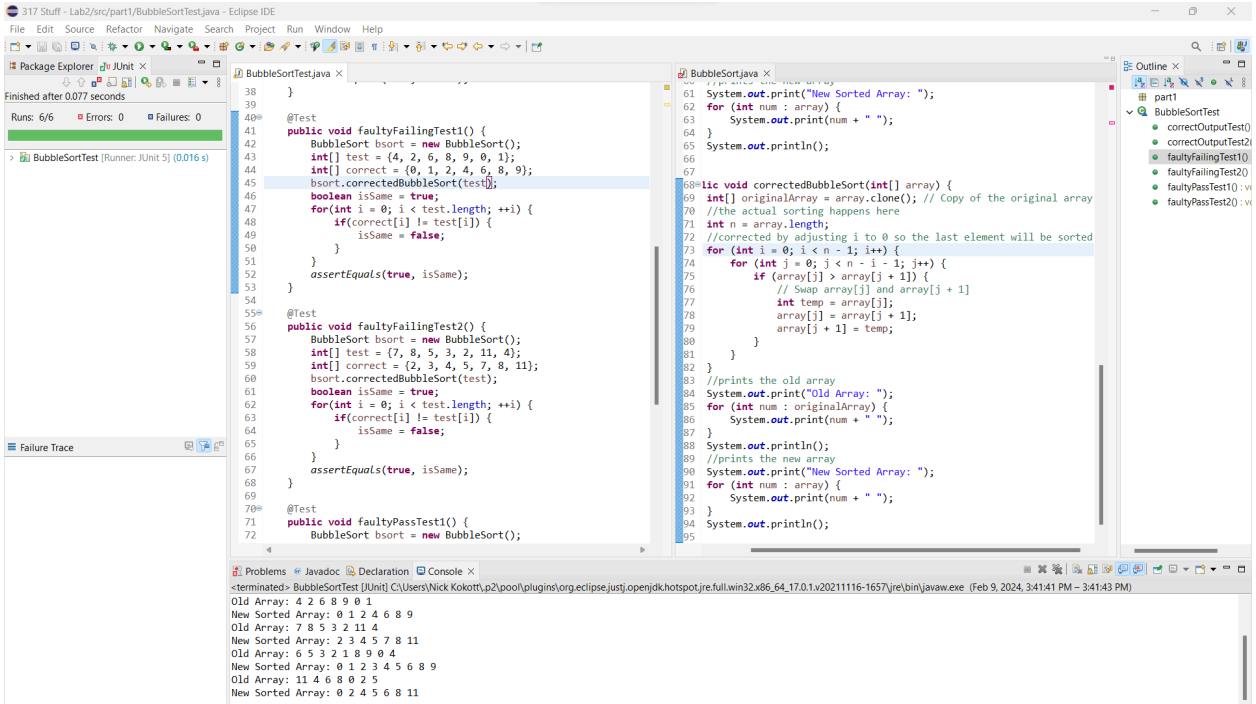
This is a screenshot of the fault being injected. This fault is an off by one error in the outermost loop in the loop structure. Instead of starting i at 0, I started i at 1, which causes the last element of the array to not be sorted.



This screenshot shows the two tests I wrote that do not pass their tests. What happened is that the last element of the array was not the highest value in the array so when they were compared, they were not equal.

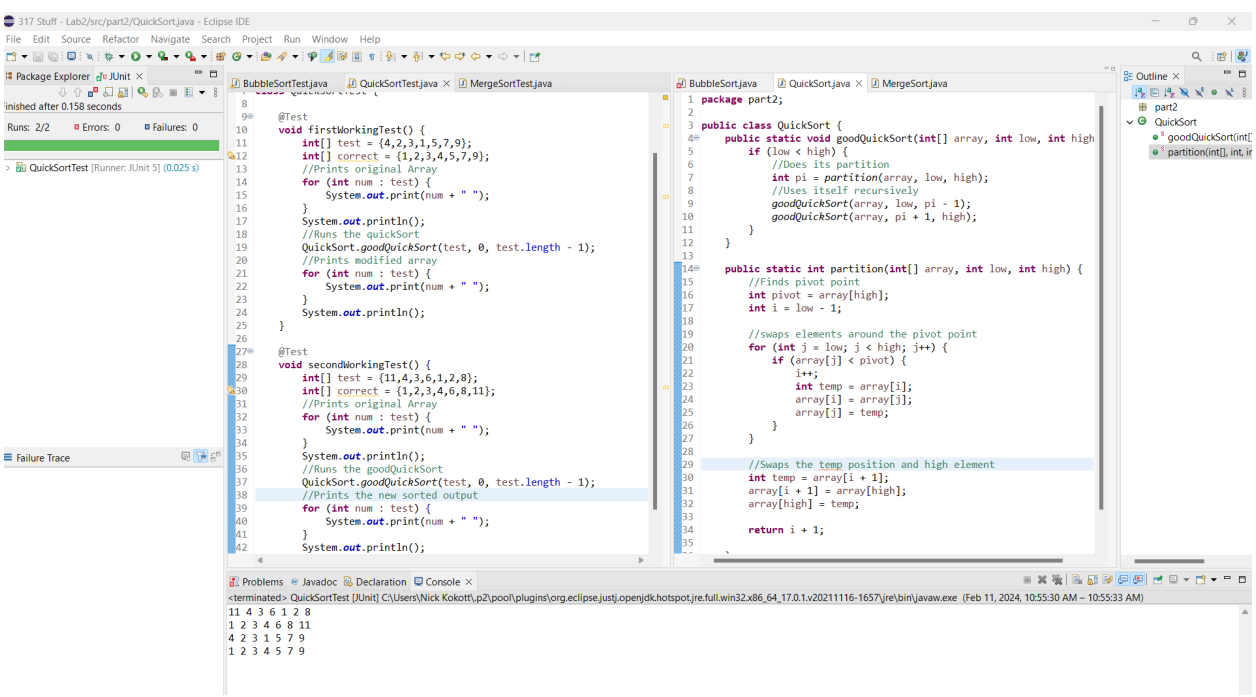


This screenshot shows two faulty tests that are passing. This happens because in these given test arrays the last element is the greatest element in the array. Due to this, the tests pass because the rest of the list was still sorted in the correct order.

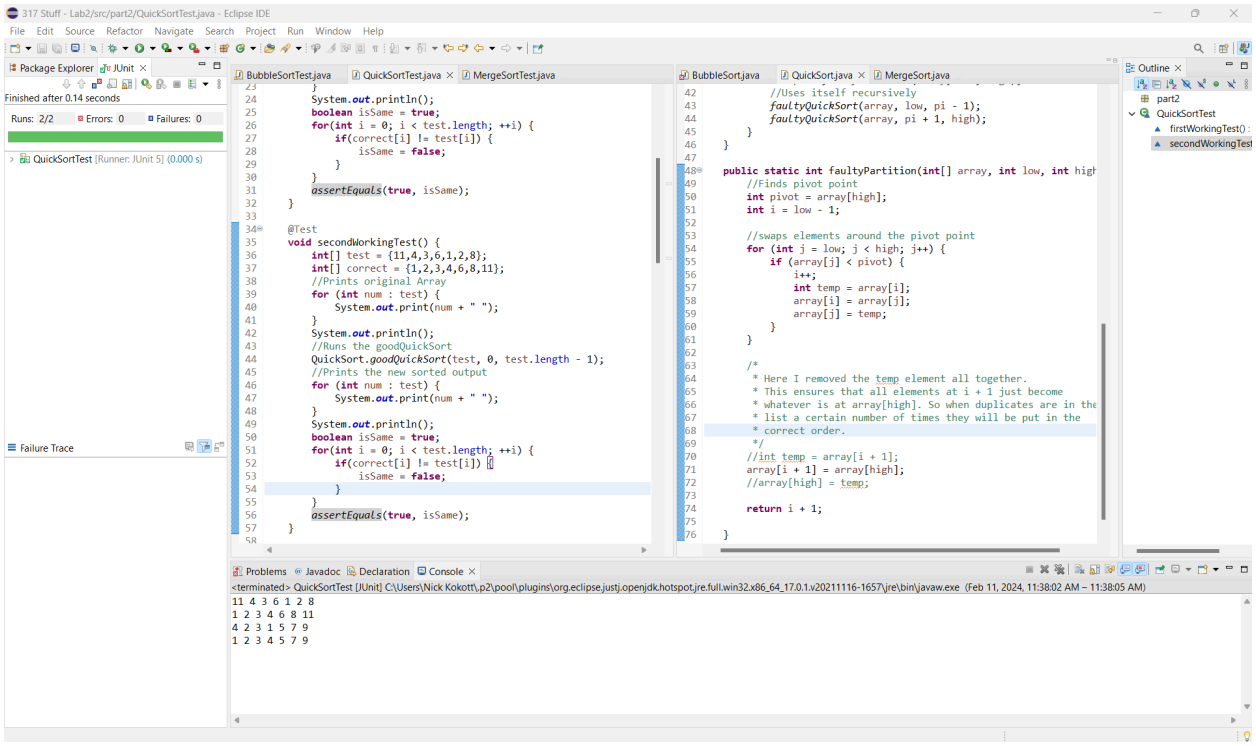


This screenshot shows all of the tests passing now that they are using the correctedBubbleSort method.

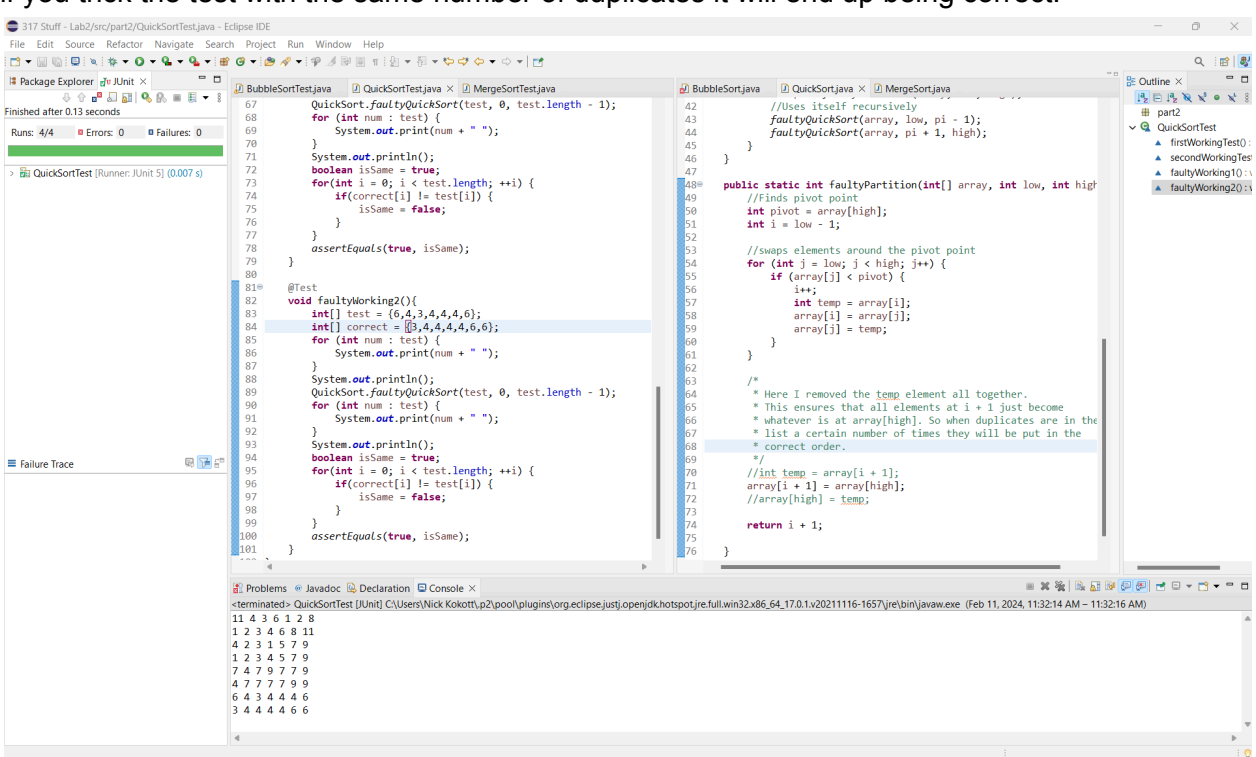
Quick Sort Pictures



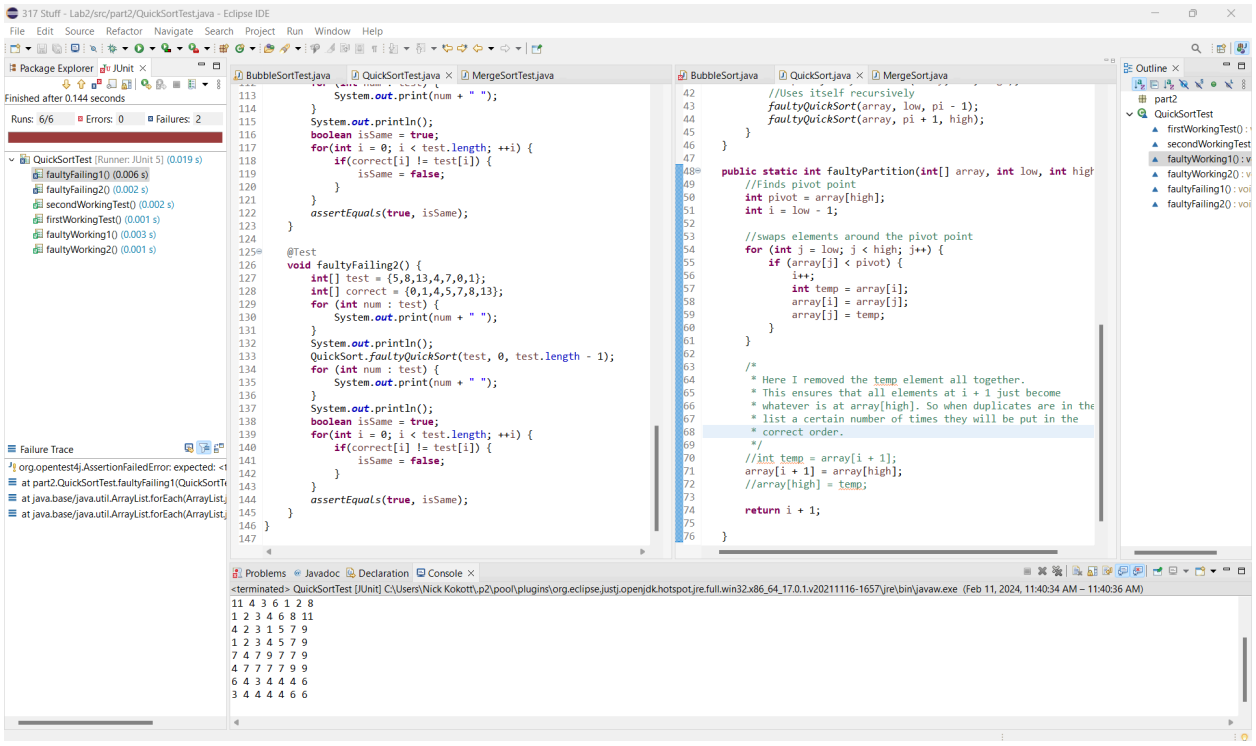
This screenshot shows both working tests with the goodQuickSort method working within the tests.



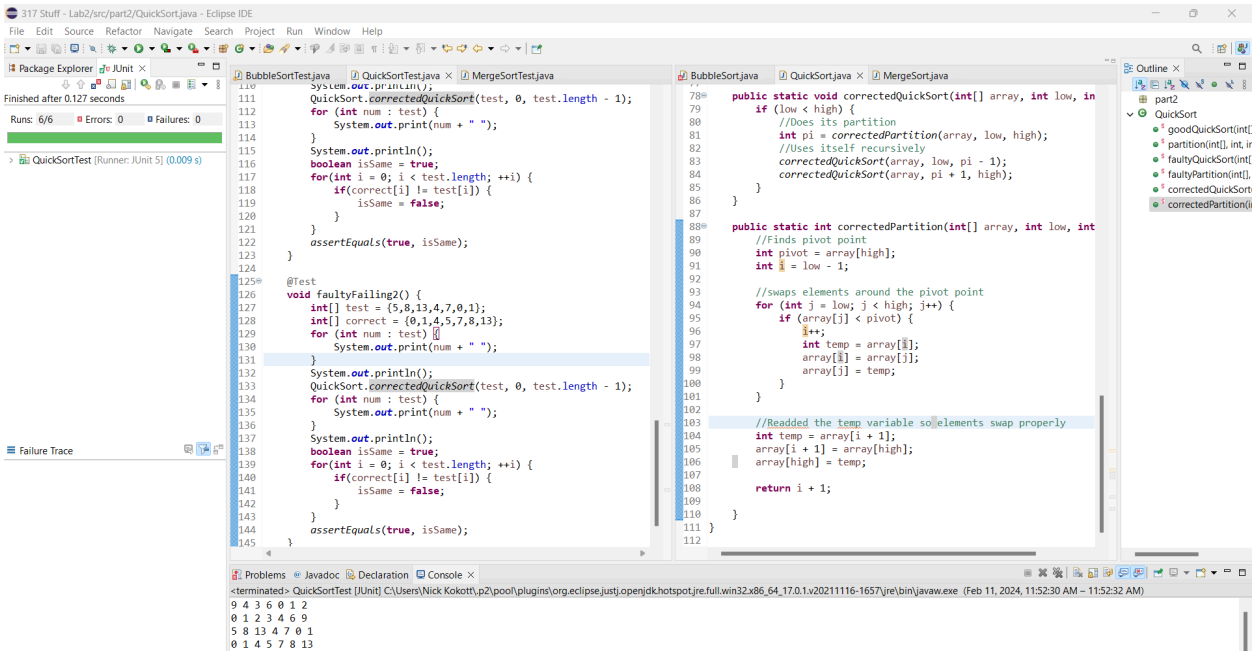
This screenshot shows that my faultyQuickSort is compiling and the previous tests are still running. My faultyQuickSort removes the temp variable in the partition method and just assigns all elements at array[i + 1] to array[temp]. This leads to duplication of elements in the array, but if you trick the test with the same number of duplicates it will end up being correct.



This screenshot shows the two tests that do not reveal the fault within my faultyQuickSort. They have the correct number of duplicates that the correct and test array will end up looking identical.

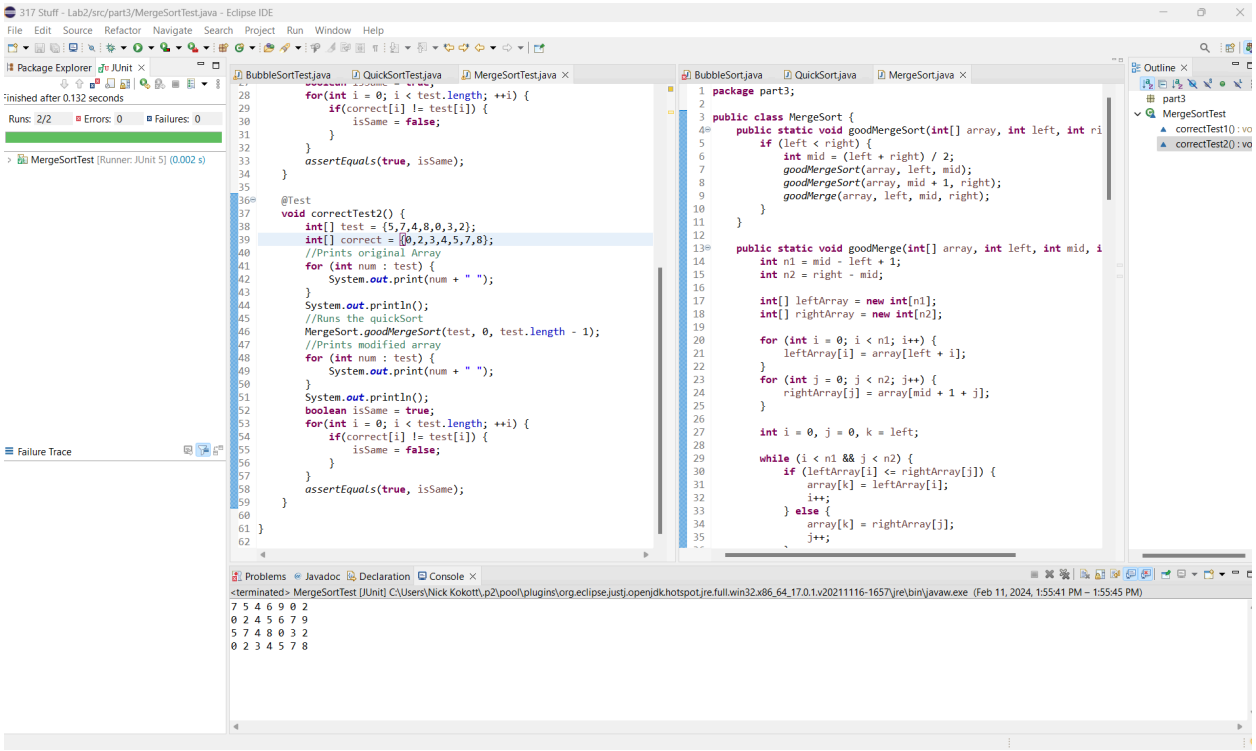


This picture shows all of the tests needed, including the new 2 failing tests, original two working tests, and the two tests that work with the faulty partition. These two new tests don't have duplicate values and due to this will fail every time when used with the faultyQuickSort.

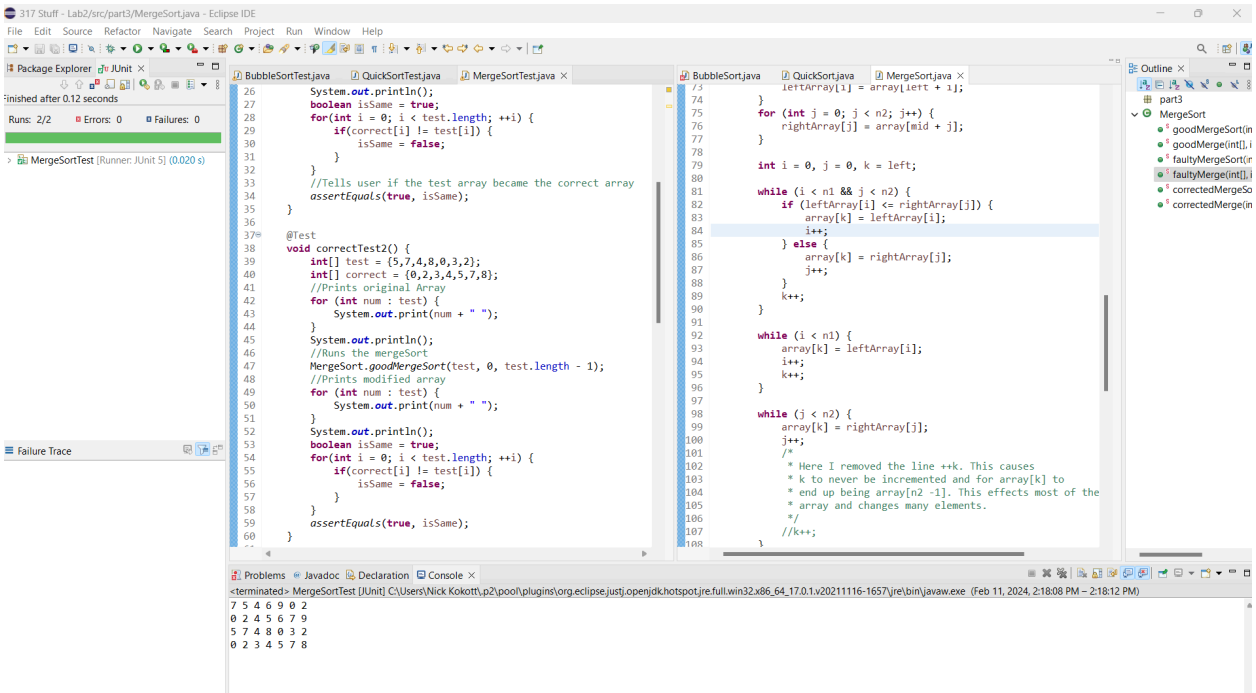


Now with adding the temp variable back in, elements are able to swap properly within the array and will be sorted in the proper order. All tests are now passing and the case arrays are printed at the bottom of the screenshot.

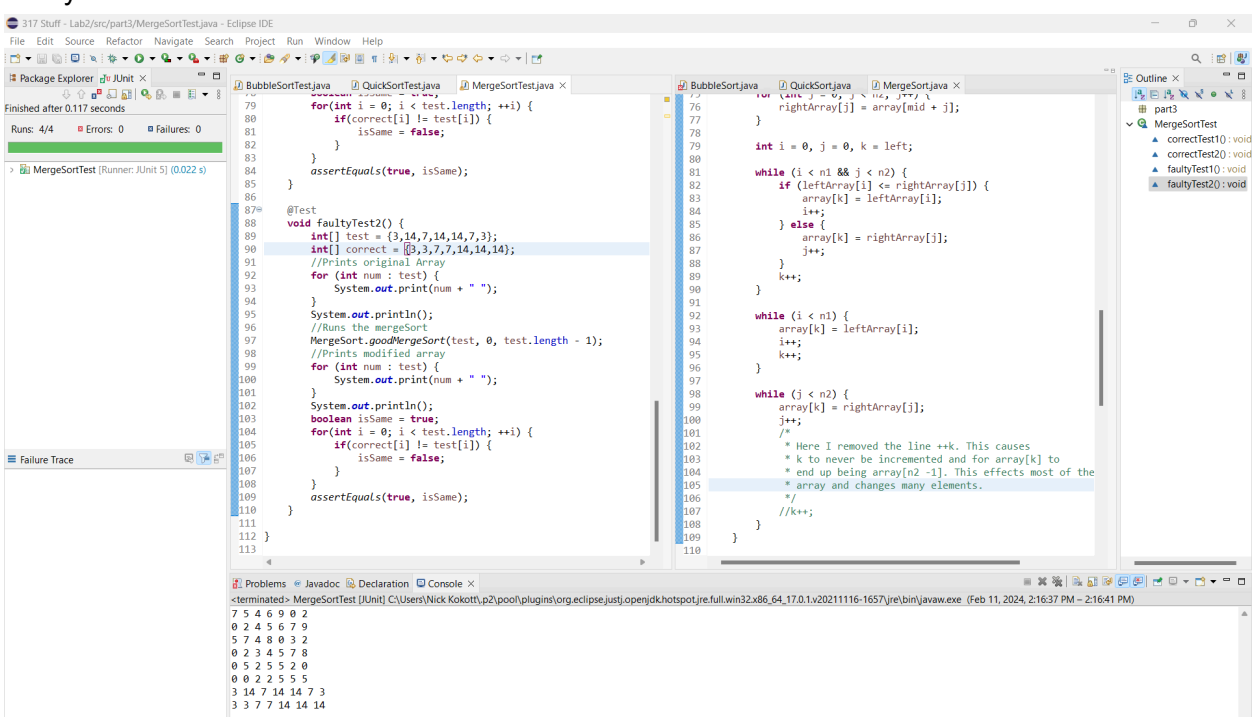
Merge Sort Pictures



This screenshot shows the goodMergeSort working and compiling with both tests working as expected.

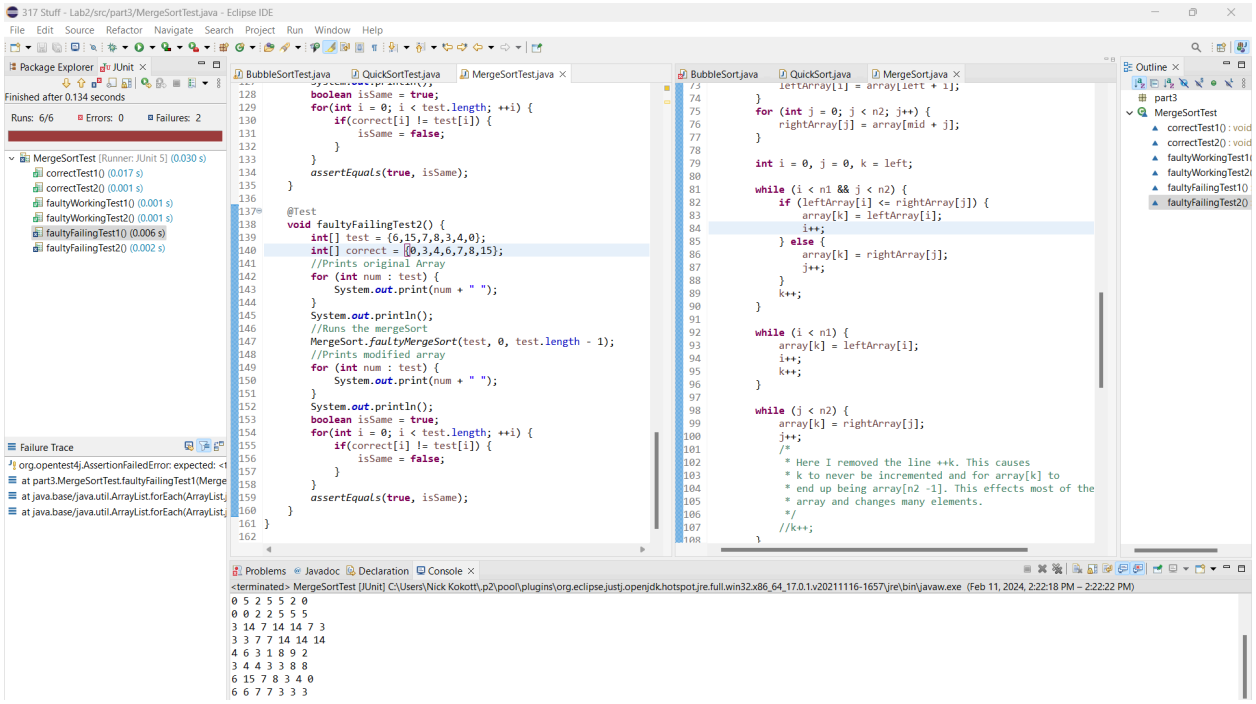


This screenshot shows the error that I injected into the goodMergeSort in order to make it faulty and run incorrectly. What I did was in the second while loop, where j goes to n2, was remove the line where it increments k by one to add a new element. This in turn just makes array[k] become array[n2 -1]. When doing this it not only messes up the array's order, but also changes integers if they are all different.

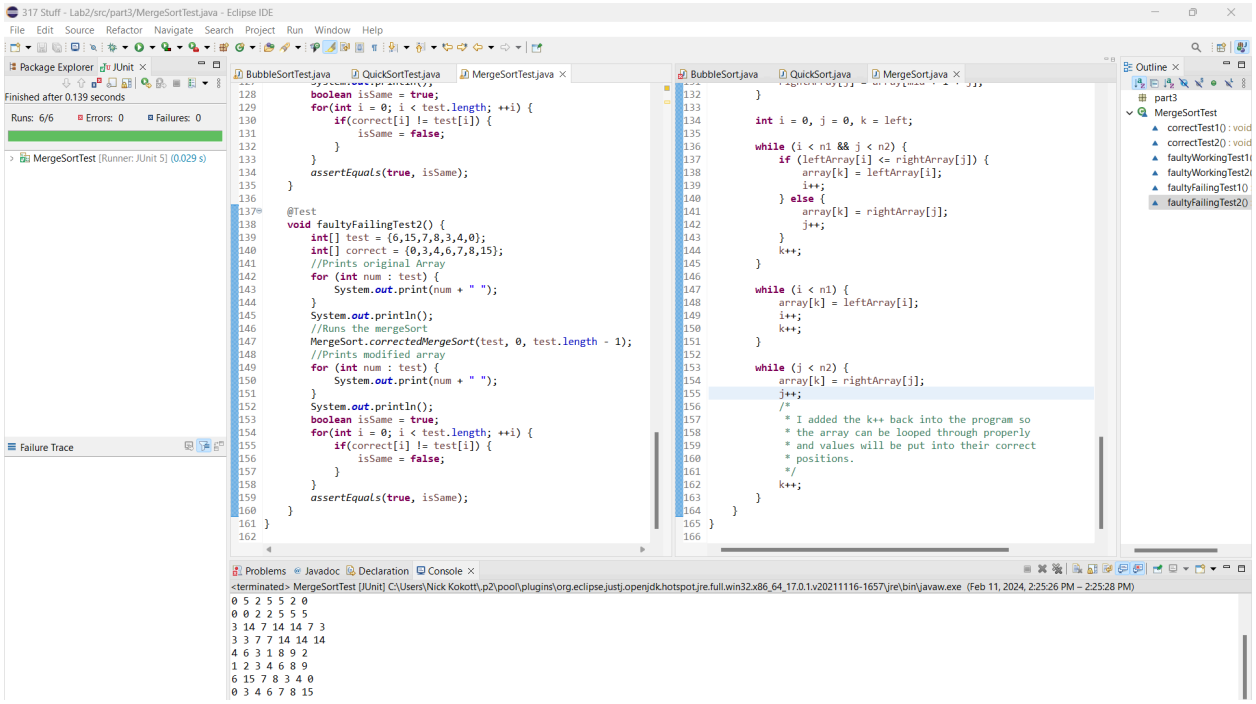


This screenshot shows the error I implemented into the merge sort as well as the tests that do not reveal the fault. In order to make these tests pass you had to use certain integers repeated

a certain number of times in a certain order. This was the only way to get these faulty tests to pass.



This screenshot shows the new two tests that I added that do reveal the fault in my modified mergeSort. As you can see in the bottom four lines, not only are the values incorrect, but there are more of some values than previously there were before.



As you can see when I ran the correctedMergeSort on the two failing tests, they now passed. This is because all values j and above actually had a chance to be merged into the array properly now that the k++ statement was back.