

Nicholas Kokott - kokottni

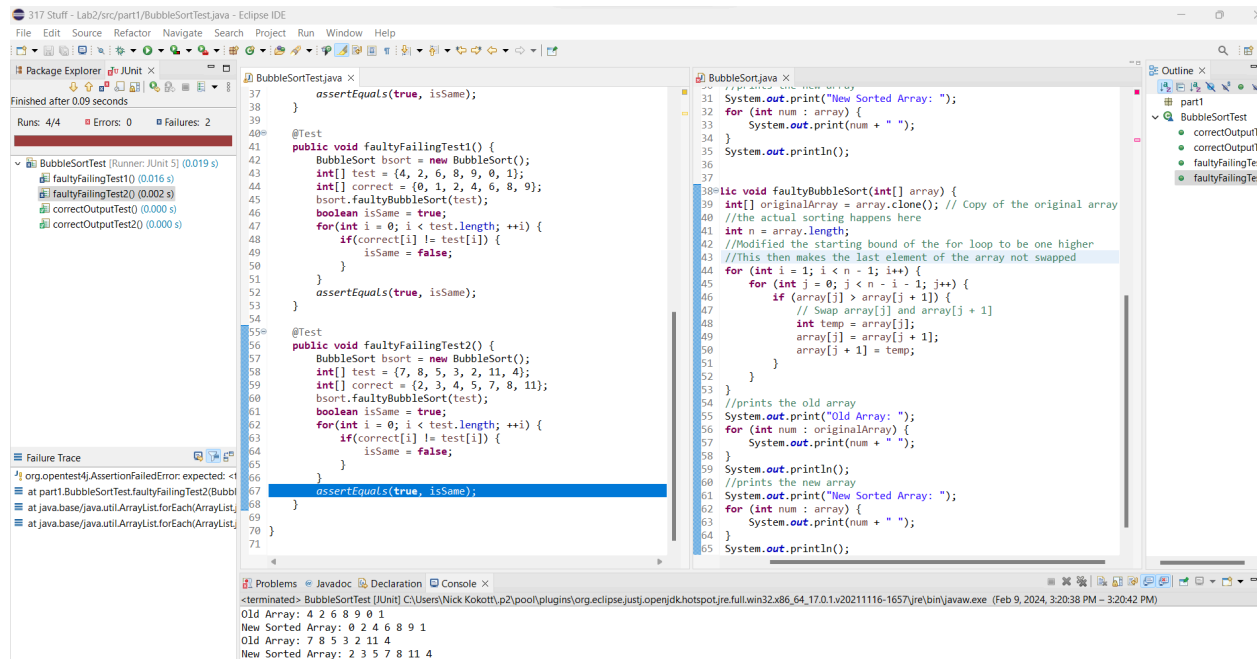
Bubble Sort pictures

The screenshot shows the Eclipse IDE with three open files: `BubbleSortTest.java`, `QuickSort.java`, and `MergeSort.java`. The `BubbleSortTest.java` file contains two test methods: `correctOutputTest()` and `correctOutputTest2()`. Both tests pass, as indicated by the 'Runs: 2/2' status in the console. The `BubbleSort.java` file shows the implementation of the bubble sort algorithm, which includes a `main` method and a `goodBubbleSort` method. The console output shows the original array `{5, 3, 8, 2, 1, 9, 4, 7, 6}` and the sorted array `{0, 1, 2, 3, 4, 5, 6, 8, 9}`.

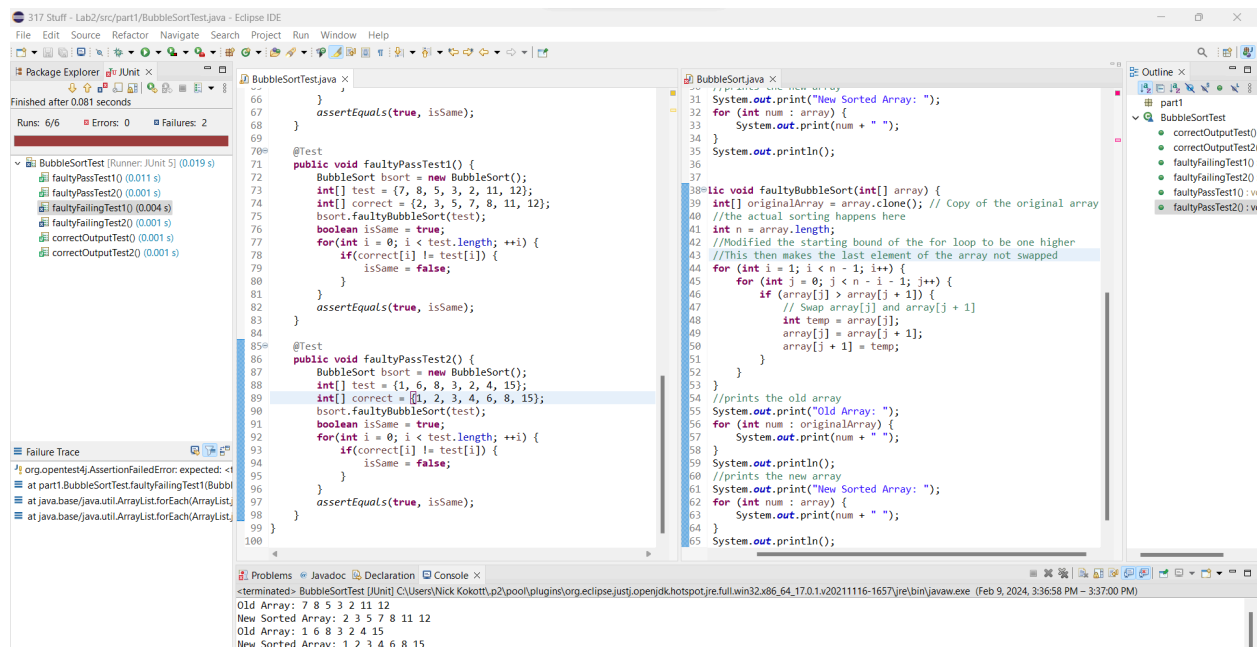
This is the original working good bubble sort with 2 passing tests to back it up

The screenshot shows the Eclipse IDE with the same three files as the previous image. The `BubbleSortTest.java` file has been modified to include a new test method, `faultyFailingTest()`, which fails. The `BubbleSort.java` file has been modified to include a new method, `faultyBubbleSort`, which implements a faulty bubble sort algorithm. The console output shows the original array `{5, 3, 8, 2, 1, 9, 4, 7, 6}` and the sorted array `{0, 1, 2, 3, 4, 5, 6, 8, 9}`. The `faultyFailingTest()` method fails because the `faultyBubbleSort` method does not sort the array correctly.

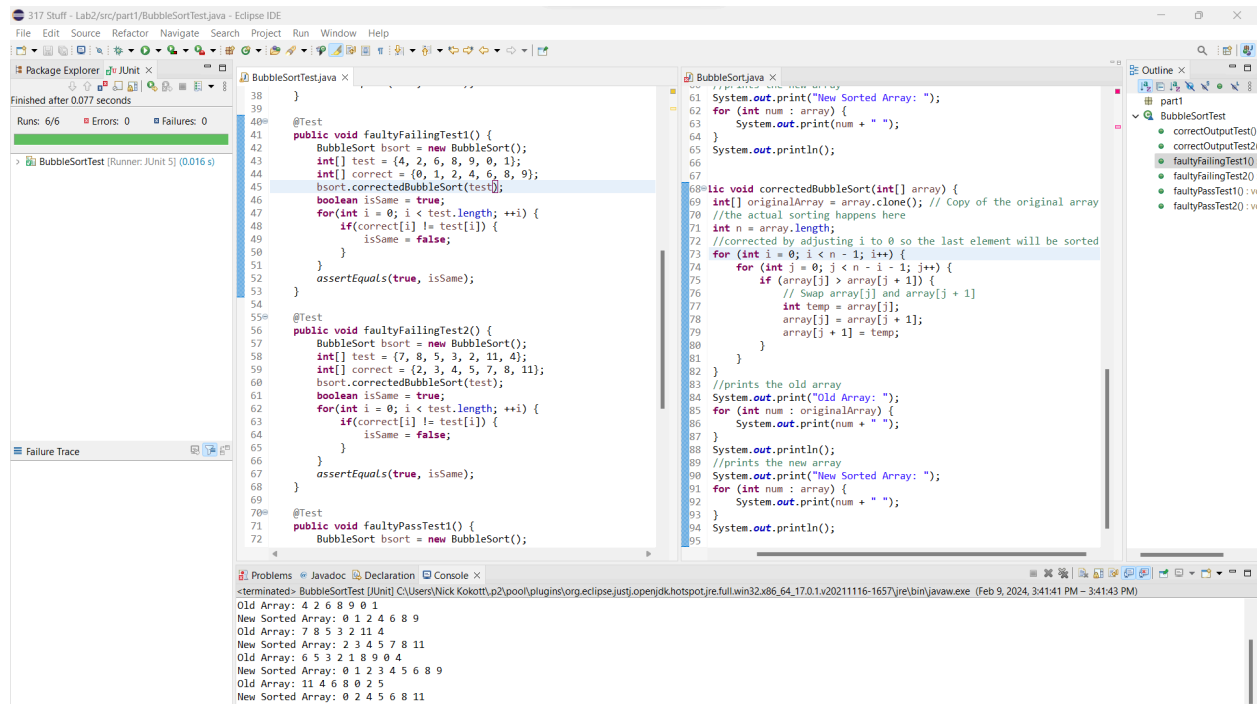
This is a screenshot of the fault being injected. This fault is an off by one error in the outermost loop in the loop structure. Instead of starting `i` at 0, I started `i` at 1, which causes the last element of the array to not be sorted.



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.

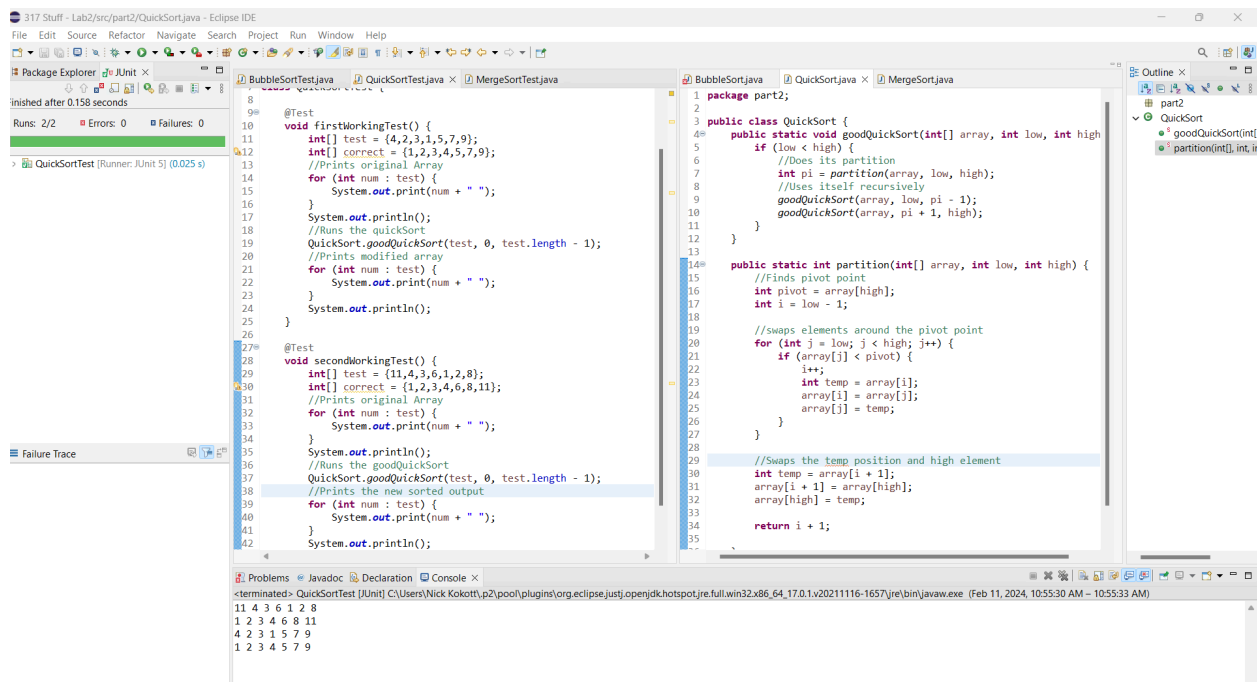


This screenshot shows two faulty tests that are passing. This happens because in these given test arrays the last element is the greatest element in the array. Due to this, the tests pass because the rest of the list was still sorted in the correct order.

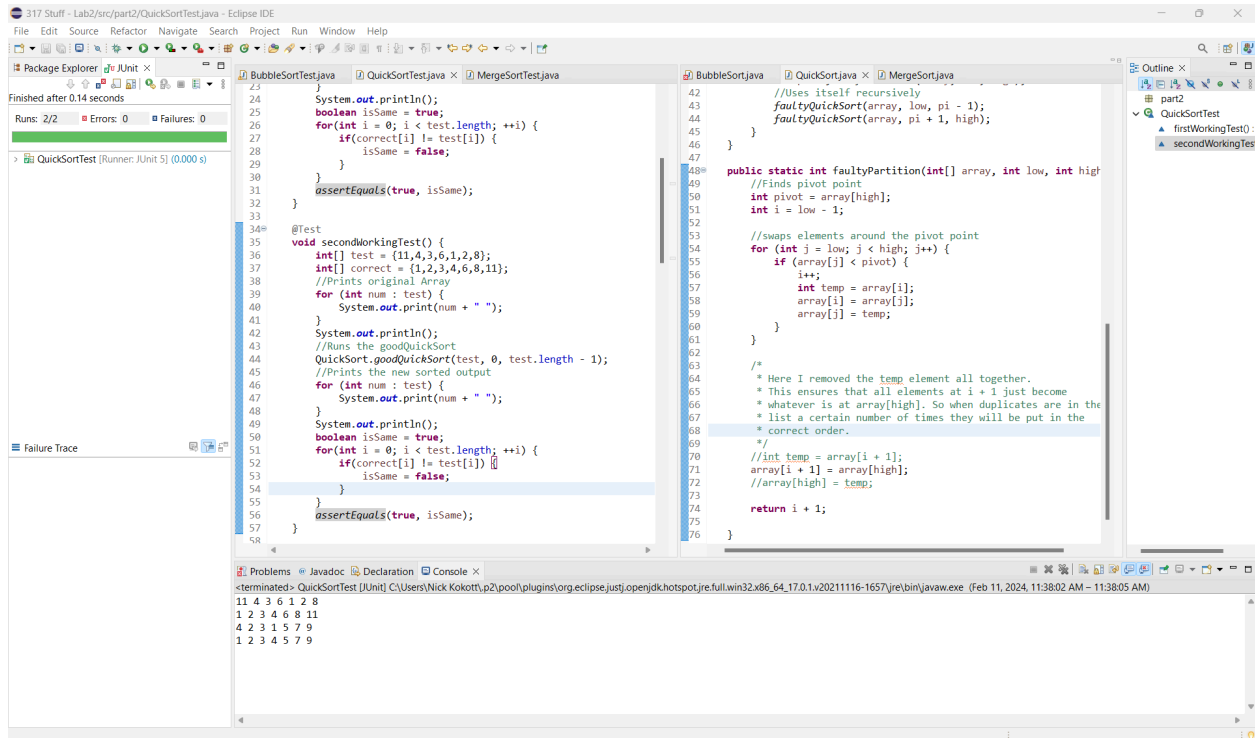


This screenshot shows all of the tests passing now that they are using the `correctedBubbleSort` method.

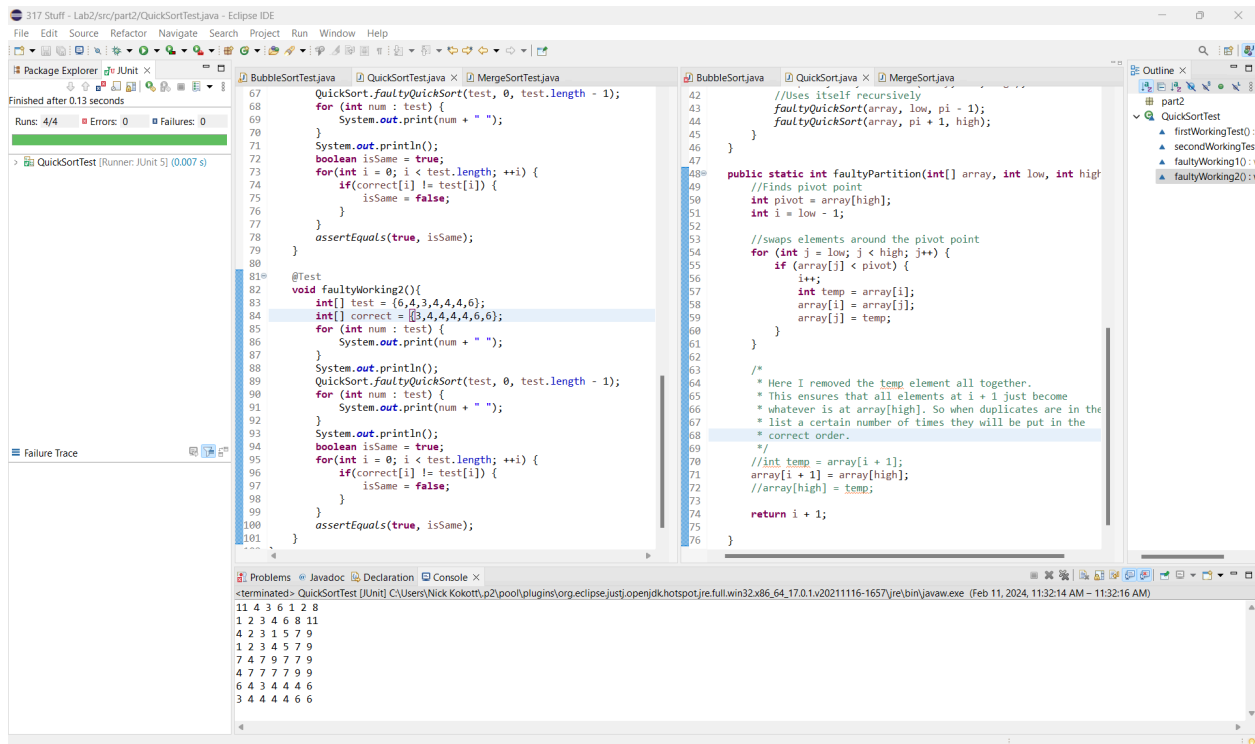
Quick Sort Pictures



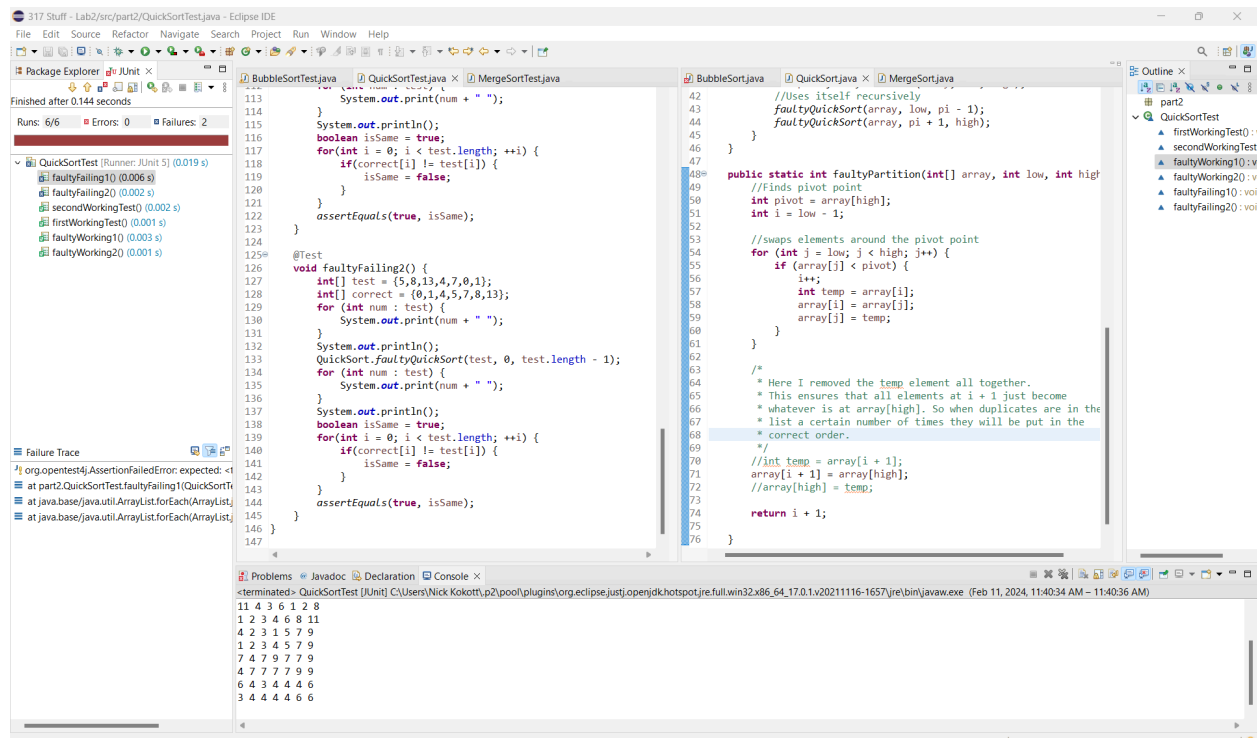
This screenshot shows both working tests with the `goodQuickSort` method working within the tests.



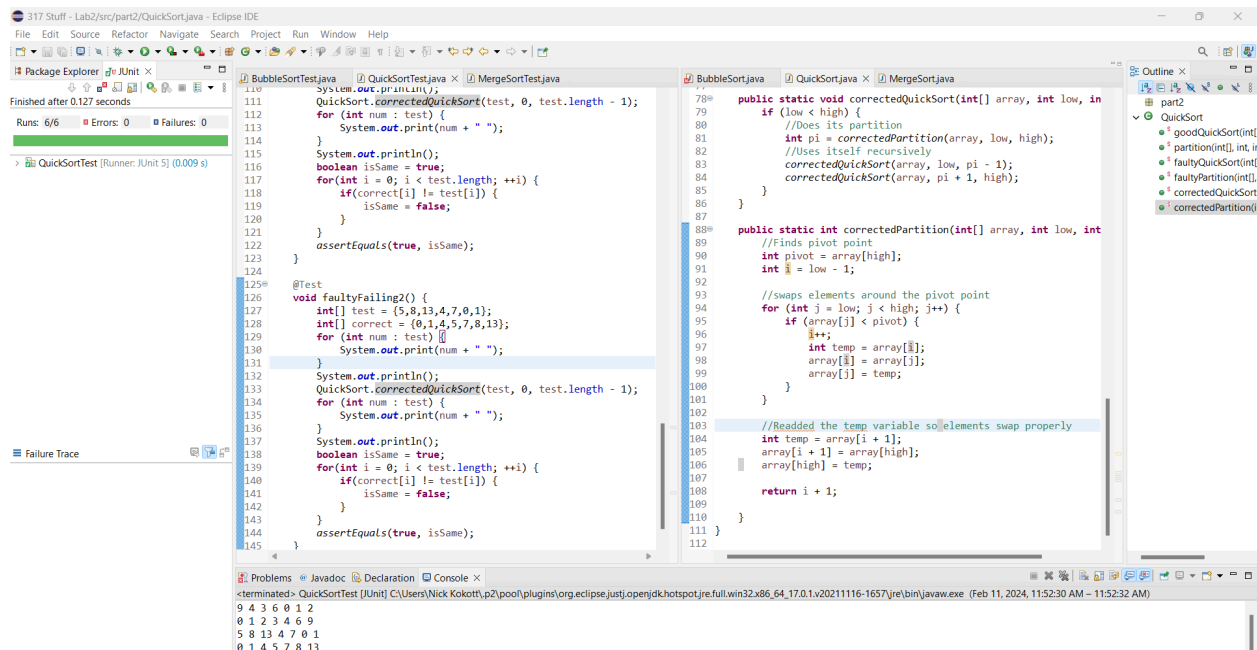
This screenshot shows that my `faultyQuickSort` is compiling and the previous tests are still running. My `faultyQuickSort` removes the `temp` variable in the partition method and just assigns all elements at `array[i + 1]` to `array[temp]`. This leads to duplication of elements in the array, but if 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.

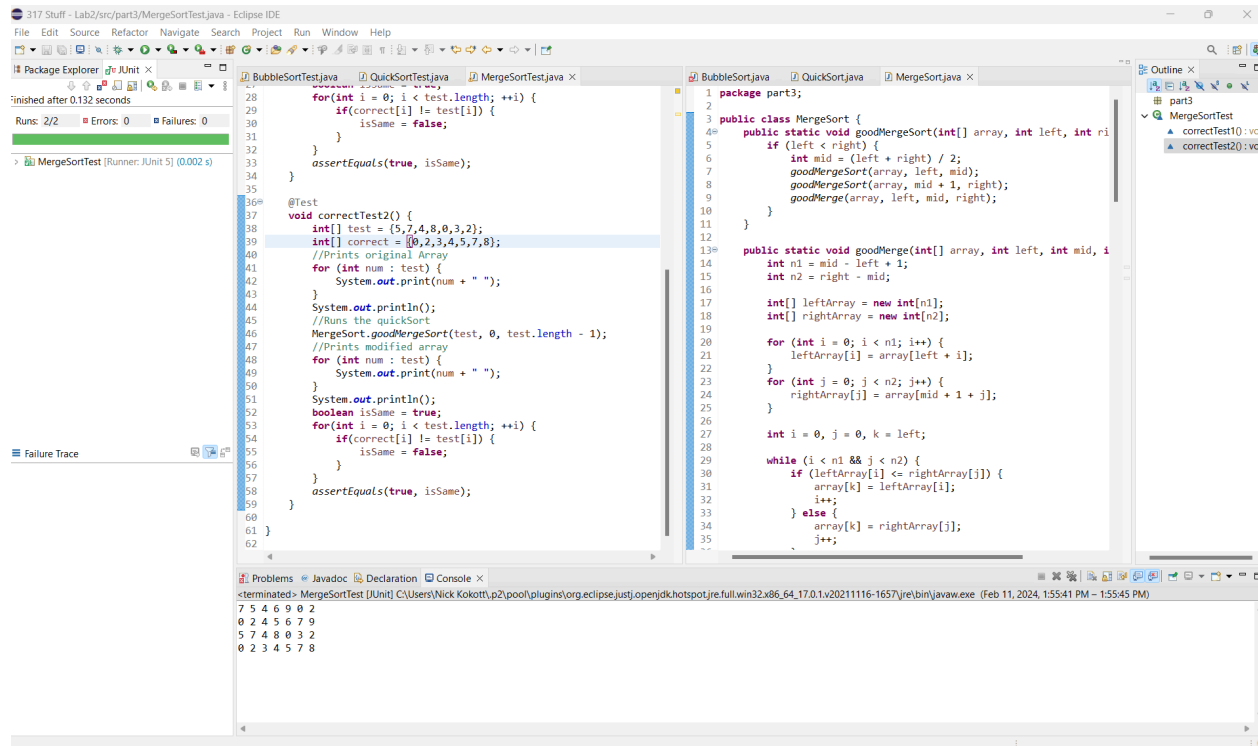


This picture shows all of the tests needed, including the new 2 failing tests, original two working tests, and the two tests that work with the faulty partition. These two new tests don't have duplicate values and due to this will fail every time when used with the faultyQuickSort.



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



The screenshot displays the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with 'part3' containing 'MergeSortTest' and 'MergeSort.java'.
- JUnit Console:** Shows the test results for 'MergeSortTest' (Runner: JUnit 5) with 2/2 runs, 0 errors, and 0 failures. The test duration is 0.002s.
- Source Editor:** Displays the 'MergeSortTest.java' file with the following code:

```
28 for(int i = 0; i < test.length; ++i) {
29     if(correct[i] != test[i]) {
30         isSame = false;
31     }
32 }
33 assertEquals(true, isSame);
34 }
35
36 @Test
37 void correctTest2() {
38     int[] test = {5,7,4,8,0,3,2};
39     int[] correct = {0,2,3,4,5,7,8};
40     //Prints original Array
41     for (int num : test) {
42         System.out.print(num + " ");
43     }
44     System.out.println();
45     //Runs the quickSort
46     MergeSort.goodMergeSort(test, 0, test.length - 1);
47     //Prints modified array
48     for (int num : test) {
49         System.out.print(num + " ");
50     }
51     System.out.println();
52     boolean isSame = true;
53     for(int i = 0; i < test.length; ++i) {
54         if(correct[i] != test[i]) {
55             isSame = false;
56         }
57     }
58     assertEquals(true, isSame);
59 }
60 }
61 }
62 }
```
- Source Editor:** Displays the 'MergeSort.java' file with the following code:

```
1 package part3;
2
3 public class MergeSort {
4     public static void goodMergeSort(int[] array, int left, int right) {
5         if (left < right) {
6             int mid = (left + right) / 2;
7             goodMergeSort(array, left, mid);
8             goodMergeSort(array, mid + 1, right);
9             goodMerge(array, left, mid, right);
10        }
11    }
12
13    public static void goodMerge(int[] array, int left, int mid, int right) {
14        int n1 = mid - left + 1;
15        int n2 = right - mid;
16
17        int[] leftArray = new int[n1];
18        int[] rightArray = new int[n2];
19
20        for (int i = 0; i < n1; i++) {
21            leftArray[i] = array[left + i];
22        }
23        for (int j = 0; j < n2; j++) {
24            rightArray[j] = array[mid + 1 + j];
25        }
26
27        int i = 0, j = 0, k = left;
28
29        while (i < n1 && j < n2) {
30            if (leftArray[i] <= rightArray[j]) {
31                array[k] = leftArray[i];
32                i++;
33            } else {
34                array[k] = rightArray[j];
35                j++;
36            }
37        }
38    }
39 }
```
- Console:** Shows the output of the test, displaying the original and modified arrays:

```
<terminated> MergeSortTest [JUnit] C:\Users\Nick Kokott\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.1.v20211116-1657\jre\bin\javaw.exe (Feb 11, 2024, 1:55:41 PM - 1:55:45 PM)
7 5 4 6 9 0 2
0 2 4 5 6 7 9
5 7 4 8 0 3 2
0 2 3 4 5 7 8
```

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

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with 'part3' containing 'MergeSort'.
- JUnit Console:** Shows 'MergeSortTest (Runner: JUnit 5) (0.020 s)' with 'Runs: 2/2', 'Errors: 0', and 'Failures: 0'.
- Source Editor (MergeSortTest.java):** Contains a test method `correctTest2()` that uses `goodMergeSort` and `faultyMergeSort` on the array `{5, 7, 4, 8, 0, 3, 2}`. It prints the original array, the modified array, and asserts that they are the same.
- Source Editor (MergeSort.java):** Contains the `goodMergeSort` and `faultyMergeSort` methods. The `faultyMergeSort` method has a comment: `* Here I removed the line ++k. This causes * k to never be incremented and for array[k] to * end up being array[n2 -1]. This effects most of the * array and changes many elements.`
- Console:** Shows the output of the test, which is a corrupted array: `7 5 4 6 9 0 2`, `0 2 4 5 6 7 9`, `5 7 4 8 0 3 2`, and `0 2 3 4 5 7 8`.

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.

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with 'part3' containing 'MergeSort'.
- JUnit Console:** Shows 'MergeSortTest (Runner: JUnit 5) (0.022 s)' with 'Runs: 4/4', 'Errors: 0', and 'Failures: 0'.
- Source Editor (MergeSortTest.java):** Contains a test method `faultyTest2()` that uses `goodMergeSort` and `faultyMergeSort` on the array `{3, 14, 7, 14, 14, 7, 3}`. It prints the original array, the modified array, and asserts that they are the same.
- Source Editor (MergeSort.java):** Contains the `goodMergeSort` and `faultyMergeSort` methods. The `faultyMergeSort` method has a comment: `* Here I removed the line ++k. This causes * k to never be incremented and for array[k] to * end up being array[n2 -1]. This effects most of the * array and changes many elements.`
- Console:** Shows the output of the test, which is a corrupted array: `7 5 4 6 9 0 2`, `0 2 4 5 6 7 9`, `5 7 4 8 0 3 2`, `0 2 3 4 5 7 8`, `0 5 2 5 2 0`, `0 0 2 2 5 5 5`, `3 14 7 14 14 7 3`, and `3 3 7 7 14 14 14`.

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.

The screenshot shows the Eclipse IDE with the `MergeSortTest.java` file open. The `Package Explorer` on the left shows the test results for `MergeSortTest`, indicating that two tests are failing: `correctTest10` and `correctTest20`. The `Failure Trace` pane shows the error message: `org.opentest4j.AssertionFailedError: expected: <1> at part3.MergeSortTest.faultyFailingTest1(MergeSortTest.java:154): actual: <2> at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)`. The `Console` pane shows the output of the `MergeSort` method, which is `0 5 2 5 5 2 0`, indicating that the array is not sorted correctly. The `Outline` pane on the right shows the structure of the `MergeSortTest` class, including the `correctTest10` and `correctTest20` methods.

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.

The screenshot shows the Eclipse IDE with the `MergeSortTest.java` file open. The `Package Explorer` on the left shows the test results for `MergeSortTest`, indicating that all tests are now passing: `correctTest10`, `correctTest20`, `faultyWorkingTest1`, `faultyWorkingTest2`, `faultyFailingTest10`, and `faultyFailingTest20`. The `Failure Trace` pane is empty. The `Console` pane shows the output of the `MergeSort` method, which is `0 0 2 2 5 5 5`, indicating that the array is now sorted correctly. The `Outline` pane on the right shows the structure of the `MergeSortTest` class, including the `correctTest10` and `correctTest20` methods. The `Console` pane also shows the output of the `MergeSort` method, which is `0 0 2 2 5 5 5`, indicating that the array is now sorted correctly.

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.