Lab 3 Quiz
ENGG 4450
November 22,2024
Arvind Dhaliwal

Link to GitHub:
https://github.com/ArvindDhaliwal/Quiz3

BubbleSort Corrected Code:

```
package com.jwetherell.algorithms.sorts;

public class BubbleSort<T extends Comparable<T>> {

   private BubbleSort() { }

   public static <T extends Comparable<T>> T[] sort(T[] unsorted) {
       boolean swapped = true;
       int length = unsorted.length;
       while (swapped) {
           swapped = false;
           for (int i = 1; i < length; i++) {
               if (unsorted[i].compareTo(unsorted[i - 1]) < 0) { // Fixed
condition for ascending order
                   swap(i, i - 1, unsorted);
                   swapped = true;
               }
           }
           length--;
       }
       return unsorted;
   }

   private static <T extends Comparable<T>> void swap(int index1, int index2,
T[] unsorted) {
       T value = unsorted[index1];
       unsorted[index1] = unsorted[index2];
       unsorted[index2] = value;
   }
}
```

BubbleSort Test Code:

```
package com.jwetherell.algorithms.sorts.test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

import com.jwetherell.algorithms.sorts.BubbleSort;

public class BubbleSortTest {

   @Test
```

```java
public void testSortIntegers() {
    Integer[] unsorted = {5, 3, 8, 4, 2};
    Integer[] expected = {2, 3, 4, 5, 8};
    Integer[] result = BubbleSort.sort(unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortStrings() {
    String[] unsorted = {"banana", "apple", "cherry", "date"};
    String[] expected = {"apple", "banana", "cherry", "date"};
    String[] result = BubbleSort.sort(unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortEmptyArray() {
    Integer[] unsorted = {};
    Integer[] expected = {};
    Integer[] result = BubbleSort.sort(unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortSingleElement() {
    Integer[] unsorted = {42};
    Integer[] expected = {42};
    Integer[] result = BubbleSort.sort(unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortAlreadySorted() {
    Integer[] unsorted = {1, 2, 3, 4, 5};
    Integer[] expected = {1, 2, 3, 4, 5};
    Integer[] result = BubbleSort.sort(unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortDescending() {
    Integer[] unsorted = {5, 4, 3, 2, 1};
    Integer[] expected = {1, 2, 3, 4, 5};
    Integer[] result = BubbleSort.sort(unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortArrayWithDuplicates() {
```

```
        Integer[] unsorted = {5, 3, 8, 5, 2, 8, 1};
        Integer[] expected = {1, 2, 3, 5, 5, 8, 8};
        Integer[] result = BubbleSort.sort(unsorted);
        assertArrayEquals(expected, result);
    }

    @Test
    public void testSortArrayWithNegativeValues() {
        Integer[] unsorted = {-5, 3, -8, 4, 2};
        Integer[] expected = {-8, -5, 2, 3, 4};
        Integer[] result = BubbleSort.sort(unsorted);
        assertArrayEquals(expected, result);
    }

    @Test
    public void testSortLargeArray() {
        Integer[] unsorted = new Integer[1000];
        for (int i = 0; i < 1000; i++) {
            unsorted[i] = (int) (Math.random() * 1000); // Random values between
0 and 999
        }
        Integer[] result = BubbleSort.sort(unsorted);
        // Test if the result is sorted
        for (int i = 1; i < result.length; i++) {
            assertTrue(result[i-1] <= result[i]);
        }
    }
}
```

QuickSort code:

```
package com.jwetherell.algorithms.sorts;

import java.util.Random;

public class QuickSort<T extends Comparable<T>> {

    private static final Random RAND = new Random();

    public static enum PIVOT_TYPE {
        FIRST, MIDDLE, RANDOM
    }

    public static PIVOT_TYPE type = PIVOT_TYPE.RANDOM;

    private QuickSort() { }
```

```
   public static <T extends Comparable<T>> T[] sort(PIVOT_TYPE pivotType, T[]
unsorted) {
       type = pivotType; // Set the pivot type globally
       sort(0, unsorted.length - 1, unsorted);
       return unsorted;
   }

   private static <T extends Comparable<T>> void sort(int start, int finish, T[]
unsorted) {
       if (start < finish) {
           int pivotIndex = partition(start, finish, unsorted);
           sort(start, pivotIndex - 1, unsorted); // Sort the left partition
           sort(pivotIndex, finish, unsorted); // Sort the right partition
       }
   }

   private static <T extends Comparable<T>> int partition(int start, int finish, T[]
unsorted) {
       T pivot = choosePivot(start, finish, unsorted);
       int s = start;
       int f = finish;
       while (s <= f) {
           while (unsorted[s].compareTo(pivot) < 0) s++; // Move left to right
           while (unsorted[f].compareTo(pivot) > 0) f--; // Move right to left
           if (s <= f) {
               swap(s, f, unsorted); // Swap elements at s and f
               s++;
               f--;
           }
       }
       return s; // Return the partition point
   }

   private static <T extends Comparable<T>> T choosePivot(int start, int finish, T[]
unsorted) {
       int pivotIndex;
       if (type == PIVOT_TYPE.RANDOM) {
           pivotIndex = RAND.nextInt(finish - start + 1) + start; // Random pivot
       } else if (type == PIVOT_TYPE.MIDDLE) {
           pivotIndex = (start + finish) / 2; // Middle pivot
       } else {
           pivotIndex = start; // First element as pivot
```

```
        }
        T pivot = unsorted[pivotIndex];
        swap(pivotIndex, finish, unsorted); // Move pivot to the end for partitioning
        return pivot;
    }

    private static <T extends Comparable<T>> void swap(int index1, int index2, T[]
unsorted) {
        T temp = unsorted[index1];
        unsorted[index1] = unsorted[index2];
        unsorted[index2] = temp;
    }
}
```

QuickSort Test Code:

```
package com.jwetherell.algorithms.sorts.test;
package com.jwetherell.algorithms.sorts.test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

import com.jwetherell.algorithms.sorts.QuickSort;

public class QuickSortTest {

    @Test
    public void testSortIntegersRandomPivot() {
        Integer[] unsorted = {5, 3, 8, 4, 2};
        Integer[] expected = {2, 3, 4, 5, 8};
        Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
        assertArrayEquals(expected, result);
    }

    @Test
    public void testSortStringsMiddlePivot() {
        String[] unsorted = {"banana", "apple", "cherry", "date"};
        String[] expected = {"apple", "banana", "cherry", "date"};
        String[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.MIDDLE, unsorted);
        assertArrayEquals(expected, result);
    }

    @Test
```

```
public void testSortEmptyArray() {
    Integer[] unsorted = {};
    Integer[] expected = {};
    Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortSingleElement() {
    Integer[] unsorted = {42};
    Integer[] expected = {42};
    Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.FIRST, unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortAlreadySorted() {
    Integer[] unsorted = {1, 2, 3, 4, 5};
    Integer[] expected = {1, 2, 3, 4, 5};
    Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.MIDDLE, unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortDescending() {
    Integer[] unsorted = {5, 4, 3, 2, 1};
    Integer[] expected = {1, 2, 3, 4, 5};
    Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortArrayWithDuplicates() {
    Integer[] unsorted = {5, 3, 8, 5, 2, 8, 1};
    Integer[] expected = {1, 2, 3, 5, 5, 8, 8};
    Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
    assertArrayEquals(expected, result);
}

@Test
public void testSortArrayWithNegativeValues() {
    Integer[] unsorted = {-5, 3, -8, 4, 2};
```

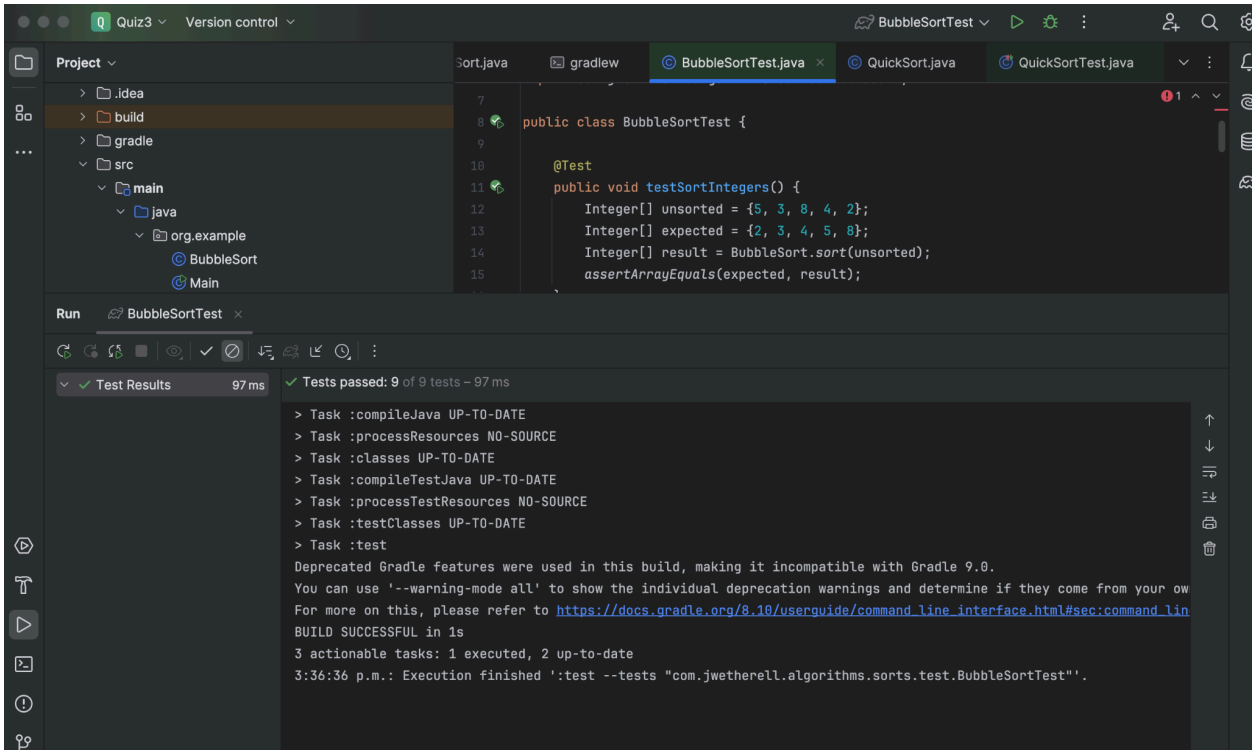
```
        Integer[] expected = {-8, -5, 2, 3, 4};
        Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
        assertArrayEquals(expected, result);
    }

    @Test
    public void testSortLargeArray() {
        Integer[] unsorted = new Integer[1000];
        for (int i = 0; i < 1000; i++) {
            unsorted[i] = (int) (Math.random() * 1000); // Random values between 0 and
999
        }
        Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
        // Test if the result is sorted
        for (int i = 1; i < result.length; i++) {
            assertTrue(result[i-1] <= result[i]);
        }
    }
}
```

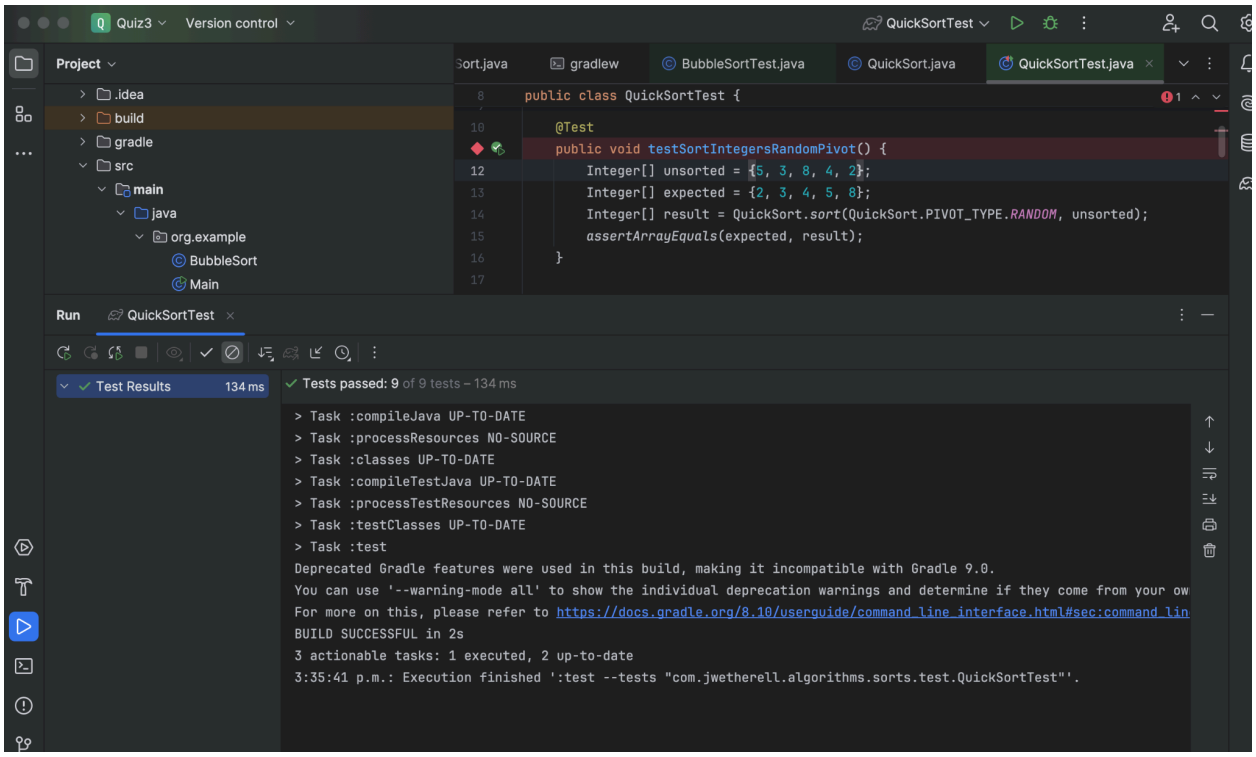
How Were Errors Fixed?

The error in the **BubbleSort** implementation was fixed by correcting the comparison logic in the if statement within the for loop. Originally, the code was sorting in descending order, but it was changed to sort in ascending order by using unsorted[i].compareTo(unsorted[i - 1]) < 0 instead of unsorted[i].compareTo(unsorted[i - 1]) > 0. This ensures the array is sorted in increasing order.

The error in the **QuickSort** implementation was fixed by correctly calculating the pivot index based on the selected pivot type (first, middle, or random). The logic was adjusted to properly compare and swap elements around the pivot, and the recursion was modified to sort the left and right partitions correctly after partitioning, ensuring proper array sorting.

Test Case Pass Confirmations


Quiz3
Version control
BubbleSortTest
Project
.idea
build
gradle
src
main
java
org.example
BubbleSort
Main
gradlew
BubbleSortTest.java
QuickSort.java
QuickSortTest.java
public class BubbleSortTest {
@Test
public void testSortIntegers() {
Integer[] unsorted = {5, 3, 8, 4, 2};
Integer[] expected = {2, 3, 4, 5, 8};
Integer[] result = BubbleSort.sort(unsorted);
assertArrayEquals(expected, result);
Run
BubbleSortTest
Test Results
97 ms
Tests passed: 9 of 9 tests – 97 ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your ow
For more on this, please refer to https://docs.gradle.org/8.10/userguide/command_line_interface.html#sec:command_lin
BUILD SUCCESSFUL in 1s
3 actionable tasks: 1 executed, 2 up-to-date
3:36:36 p.m.: Execution finished ':test --tests "com.jwetherell.algorithms.sorts.test.BubbleSortTest"'.


Quiz3
Version control
QuickSortTest
Project
.idea
build
gradle
src
main
java
org.example
BubbleSort
Main
gradlew
BubbleSortTest.java
QuickSort.java
QuickSortTest.java
public class QuickSortTest {
@Test
public void testSortIntegersRandomPivot() {
Integer[] unsorted = {5, 3, 8, 4, 2};
Integer[] expected = {2, 3, 4, 5, 8};
Integer[] result = QuickSort.sort(QuickSort.PIVOT_TYPE.RANDOM, unsorted);
assertArrayEquals(expected, result);
}
Run
QuickSortTest
Test Results
134 ms
Tests passed: 9 of 9 tests – 134 ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your ow
For more on this, please refer to https://docs.gradle.org/8.10/userguide/command_line_interface.html#sec:command_lin
BUILD SUCCESSFUL in 2s
3 actionable tasks: 1 executed, 2 up-to-date
3:35:41 p.m.: Execution finished ':test --tests "com.jwetherell.algorithms.sorts.test.QuickSortTest"'.