CMPE3008 Assignment 1

Software Engineering Testing

12/06/2022

Presented by,

Harshi Kasundi Banaranayake
20583387

Pramoda Gunarathne
20531218

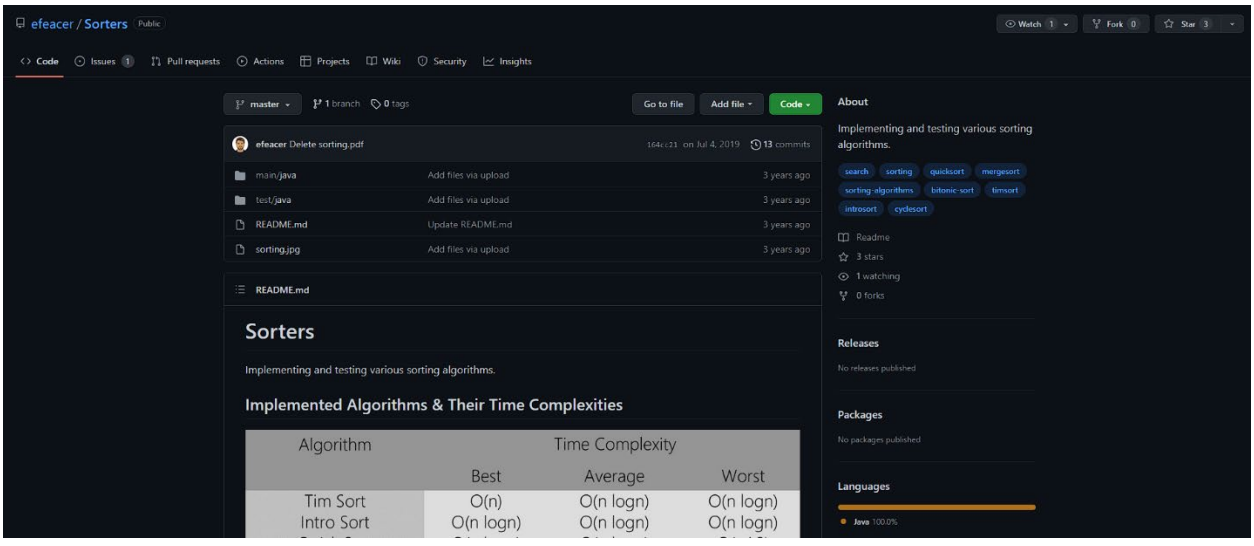
Sahas Renuja Gunasekara
20462075

Table of Contents

1. Details of Project

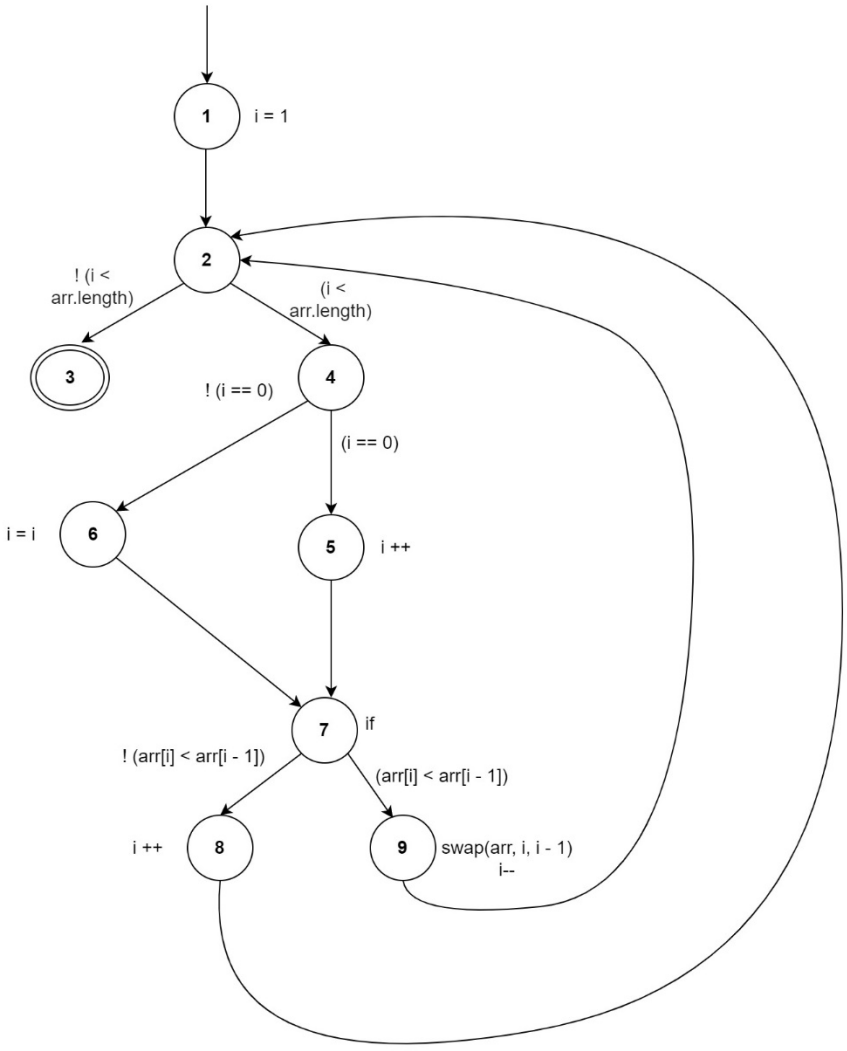
The project is chosen from GitHub User Efe Acer (https://github.com/efeacer), a Master of Computer Science Student at the University of Oxford. The repository is a Sorters Repository (https://github.com/efeacer/Sorters) to implement and test various sorting algorithms according to the README.md file.

The two chosen codes are the OddEvenSorter (https://github.com/efeacer/Sorters/blob/master/main/java/sorting/OddEvenSorter.java) and the Gnome Sorter (https://github.com/efeacer/Sorters/blob/master/main/java/sorting/GnomeSorter.java) from the repository and the test code is located at SorterTests.java (https://github.com/efeacer/Sorters/blob/master/test/java/SorterTests.java#L25). The graphs are created for these two source methods.

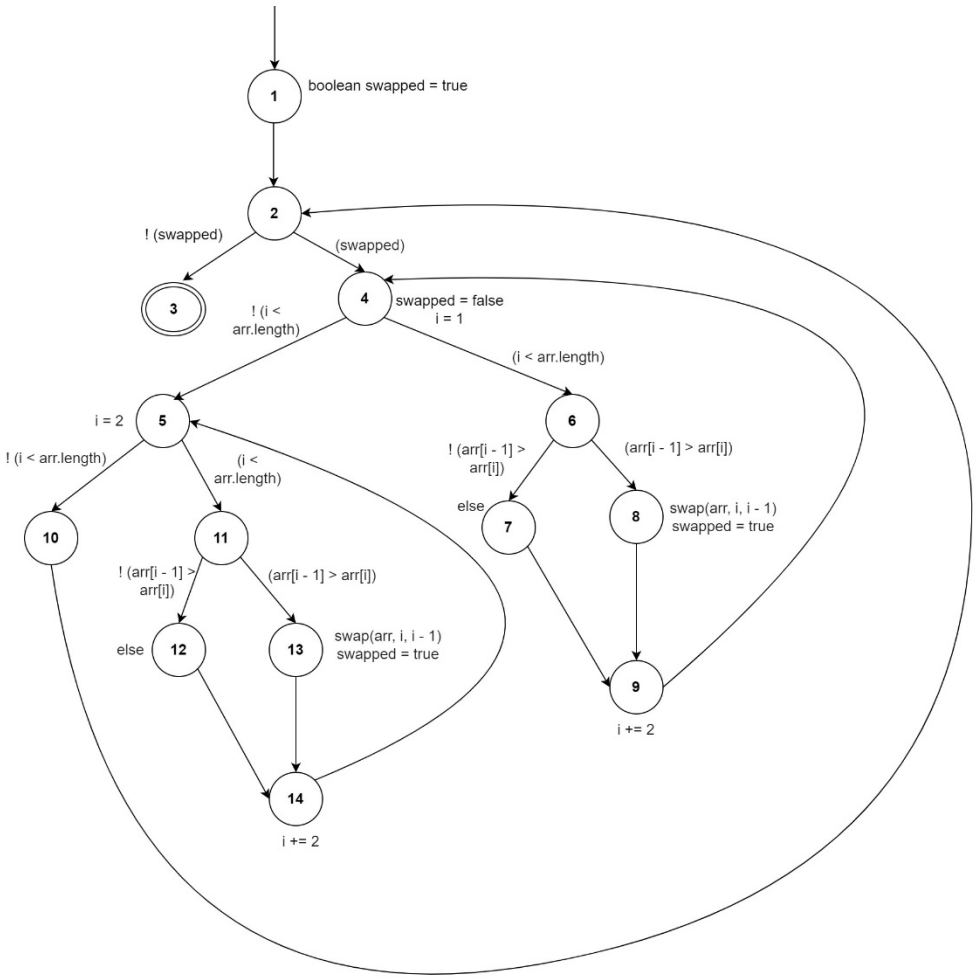


2. Chosen Methods and Associated Graphs

GnomeSort



OddorEvenSort



3. Coverage for the Graphs

GnomeSort

Prime Path Coverage: { [1,2,3], [1,2,4,5,7,8], [1,2,4,5,7,9], [2,4,5,7,8,2], [2,4,5,7,9,2], [2,4,6,7,8,2], [2,4,6,7,9,2], [4,5,7,9,2,4], [4,5,7,9,2,3], [4,5,7,8,2,4], [4,5,7,8,2,3], [6,7,8,2,3], [6,7,9,2,3], [6,7,8,2,4], [6,7,9,2,4], [8,2,4,5,7,8], [8,2,4,6,7,8] [9,2,4,5,7,9], [9,2,4,6,7,9], [7,8,2,4,5,7], [7,8,2,4,6,7] }

OddorEvenSort

Prime Path Coverage: { [1,2,3], [1,2,4,5,10], [1,2,4,5,11,12,14], [1,2,4,5,11,13,14], [1,2,4,6,7,9], [1,2,4,6,8,9], [2,4,5,10,2], [4,6,7,9,4], [4,6,8,9,4], [9,4,6,8,9], [9,4,6,7,9], [5,11,12,14,5], [5,11,13,14,5], [14,5,11,12,14], [14,5,11,13,14], [12,14,5,10,2,3], [12,14,5,11,13], [12,14,5,11,12], [12,14,5,10,2,4,6,8,9], [12,14,5,10,2,4,6,7,9], [13,14,5,10,2,3], [13,14,5,11,12], [10,2,4,5,10], [10,2,4,5,11,12,14], [13,14,5,10,2,4,6,7,9], [13,14,5,10,2,4,6,8,9], [13,14,5,11,13],[6,7,9,4,6], [6,8,9,4,6], [6,7,9,4,5,10,2,3], [6,7,9,4,5,11,12,14], [6,7,9,4,5,13,14], [6,8,9,4,5,10,2,3], [6,8,9,4,5,11,12,14], [6,8,9,4,5,11,13,14], [7,9,4,6,7], [8,9,4,6,8] }

Base Choice Coverage

Base choice coverage is same for both methods.

Characteristic	Block 1	Block 2	Block 3
Length of the input array	equals to 0	equals to 1	greater than 1
Input array is a not initialized (null)	True	False	
Input array is sorted in ascending order	True	False	
Input array is sorted in descending order	True	False	
Input array contains ONLY negative integers	True	False	
Input array contains ONLY positive integers	True	False	
Input array contains duplicate values	True	False	

(Array length, is input array null, sorted ascending, sorted descending, is array made of only negative integers, is array made of only positive integers, does array contain duplicate values)

Base : "(3,F,T,F,F,F,F)"

(2,F,T,F,F,F,F) (3,T,T,F,F,F,F) (3,F,F,F,F,F,F) (3,F,T,T,F,F,F) (3,F,T,F,T,F,F) (3,F,T,F,F,T,F) (3,F,T,F,F,F,T)
(0,F,T,F,F,F,F) (3,F,T,F,F,F,F) (3,F,T,F,F,F,F) (3,F,T,F,F,F,F) (3,F,T,F,F,F,F) (3,F,T,F,F,F,F) (3,F,T,F,F,F,F)
(1,F,T,F,F,F,F)

16 Base Choice tests were found. 6 of these were redundant.

Furthermore, some infeasible Base Choice tests can be identified by handling constraints.

Arrays that are not initialized (i.e., null arrays) cannot be tested under any given condition.
Tests that can be excluded - (3,T,T,F,F,F,F)

Arrays of length 0 (empty arrays cannot be sorted in any order)
Test that can be excluded - (0,F,T,F,F,F,F)

Input array cannot be in ascending AND descending order unless the input array contains duplicate values.
Test that can be excluded - (3,F,T,T,F,F,F)

These infeasible base choice tests can be handled by changing a value to another non-base choice to find a feasible combination.
(3,T,T,F,F,F,F) -> (3,F,T,F,F,F,F) redundant base choice test. will be ignored.
(0,F,T,F,F,F,F) -> (0,F,F,F,F,F,F)
(3,F,T,T,F,F,F) -> (3,F,T,T,F,F,T)

Reachable Base Choice Tests

(3,F,T,F,F,F,F) (3,F,F,F,F,F,F) (3,F,T,T,F,F,T) (3,F,T,F,T,F,F) (3,F,T,F,F,T,F) (3,F,T,F,F,F,T)
(2,F,T,F,F,F,F)
(0,F,F,F,F,F,F)
(1,F,T,F,F,F,F)

4. Analysis on Tests

testWithReverseSortedLongArray() for OddEvenSorter

Since the input is a reversely sorted array with elements ranging from 0 to a constant positive value defined LONG_ARR_LENGTH, the test method testWithReverseSortedLongArray() fails to execute few test scenarios identified in Part 3 for method OddEvenSorter. They are,

- Test cases where the input array in not initialized (i.e., null arrays).
- Test cases where the input array is empty (length of the array is zero).
- Test cases where the input array has only one element (length of the array is 1).
- Test cases where the input only contains negative integers.
- Test cases where the input array contains integers of mixed sign + zero as array elements.
- Test cases where the input array is in ascending order (already sorted).
- Test cases where the input array is arbitrarily ordered/not sorted.
- Test cases where the input array may contain duplicate values.

By comparing the test method against the graph produced for Odd-Even sorting algorithm. It is evident that the chosen test method fails to execute some prime paths discovered.

- Since the array is sorted in descending order, the test method will never execute test path [1,2,3] and it will also fail to test path [1,2,4,6,7,9] and all test paths that tours the said path using a side-trip or a detour since element at array[i-1] will always be greater than the element at array[i] due to the same reason.

- Similarly, the test method will also fail to capture path [1,2,4,5,11,12,14] and associated test paths that traverses using a side-trip or a detour due to the input being a reversely sorted array of integers.

NOTE: Though the chosen test method fails to capture all test scenarios, the test suite itself is capable of handling most of these “overlooked” scenarios.

testWithLongArrays() for GnomeSort

The test method testWithLongArrays() will succeed in some of the scenarios found in the base choice analysis as the method uses a long array of integers and the randInt method. But since the input is a long array of integers with elements varying from 0 to a constant positive value defined as LONG ARR LENGTH, it will fail to perform some of the test cases. They are :

- Test cases where the input array is not initialized (i.e., null arrays).
- Test cases where the input array is empty (length of the array is zero).
- Test cases where the input array has only one element (length of the array is 1).

However, the randInt method implemented in this program will also not account for negative values as it uses constants ranging from 0 to 10000. This will in turn result in failing to execute test scenarios where the input only contains negative integers and test cases where the input array contains integers of mixed sign + zero as array elements.

When the test method is compared to the graph generated by the GnomeSorter algorithm, almost all the prime paths are covered with no failure to execute any path as the long array will succeed in passing test cases such as input arrays sorted in ascending order, arbitrarily ordered and arrays that may contain duplicate values.

In contrast, to the test method testWithReversedSortedArray(), this test method will execute all the prime paths and the last three characteristics in the base choice test cases mentioned above, hence this method is better than the last test method selected.

5. Research Tool Report

TestNG

TestNG is a testing framework self-described as “inspired from JUnit and NUnit” whilst introducing new functionalities that make it a much more extensive framework. Created by Cedric Beust, a seasoned software executive who was one of the early members of the Android Team also known for creating the Android Gmail application and JCommander Java Libraries, out of “frustration for some JUnit deficiencies” (TestNG, 2019).

The advantages of TestNG over other frameworks such as JUnit are, the possibility to use Annotations, ability to have flexible test configuration, support for data-driven testing and parameters, support for a variety of tools and plug-in’s such as Eclipse, IDEA and much more. On top of the other features the three other pivotal features are that the annotations are much more intuitive, the test cases can be grouped effectively, and parallel testing is possible. However, in this report we will look at test groups and parameters as that is part of our CMPE3008 syllabi scope.

According to the TestNG official documentation for a simple test the usage of this framework is as follows.

```
package example1;

import org.testng.annotations.*;

public class SimpleTest {

 @BeforeClass
 public void setUp() {
   // code that will be invoked when this test is instantiated
 }

 @Test(groups = { "fast" })
 public void aFastTest() {
   System.out.println("Fast test");
 }

 @Test(groups = { "slow" })
 public void aSlowTest() {
    System.out.println("Slow test");
 }

}
```

For a .java file as such we have 3 Annotations of TestNG on display here. For example, @BeforeClass means that the annotated method will run before any test method belonging to the classes inside the <test> tag is run. This is useful in defining if the testing itself is multithreaded (which is possible inside TestNG) in setting up. We have also grouped Test cases as “fast” and “slow” as well which enables only certain groups to be run at a particular time.

A few key things to note here is that we didn’t have to extend classes or implement an interface, and even though the example uses JUnit conventions the methods can be called in any name as it’s the annotations which inform the TestNG what they are, and a test method can belong to one or many groups. Once this file has been compiled into a “build” directory we can invoke the test with the command line. (Alternatively, an ant task or an XML file)

This is how the TestNG .xml file looks like. What is specified in this folder is that we will be looking at all .class files inside the build directory and will evaluate the “fast” group of Tests. This means that setUp() and aFastTest() are the only two functions that will be invoked according to this specification of TestNG file.

```
<project default="test">

 <path id="cp">
   <pathelement location="lib/testng-testng-5.13.1.jar"/>
   <pathelement location="build"/>
 </path>

 <taskdef name="testng" classpathref="cp"
          classname="org.testng.TestNGAntTask" />

 <target name="test">
   <testng classpathref="cp" groups="fast">
     <classfileset dir="build" includes="example1/*.class"/>
   </testng>
 </target>

</project>
```

The simplest way in which we can invoke this TestNG file is by running the script “java org.testng.TestNG testing1.xml” according to the documentation. However, it can also be run using ant, Gradle, maven and many other tools.

```
@Parameters({ "first-name" })
@Test
public void testSingleString(String firstName) {
  System.out.println("Invoked testString " + firstName);
  assert "Cedric".equals(firstName);
}
```

In this example of a class file, we can see that we have an added annotation as @Parameters. This is because using TestNG we can pass parameters to functions directly.

```
<suite name="My suite">
  <parameter name="first-name"  value="Cedric"/>
  <test name="Simple example">
  <-- ... -->
```

This is not a feature present in JUnit. The actual usage of this can be depicted in the following testing.xml file where we use the <parameter/> section to refer to the parameter and the value being passed. This allows for multiple parameters to be passed directly to any functions inside the class file with annotations. This is not just limited to @Test functions but also to @BeforeClass, @AfterClass and all other annotations.

Overall, this framework can be quite easily viewed as a natural evolution of the existing and famous JUnit framework as there are even methodologies depicted on the actual TestNG Website on migrating JUnit code to be compatible with TestNG. The existing code base for TestNG has been as updated as recently as January 2022 with a core-team around it. Close to 200,000 lines of code of the TestNG framework has been changed this year showing that it is still an active framework that is updated often to meet to the changing demands of the Software Engineering Testing Landscape (GitHub, 2022). Overall, after viewing the possibilities of TestNG, this team can wholeheartedly recommend TestNG for Java related testing in any location where traditionally JUnit would’ve been used for the additional flexibility that is offered through this powerful framework. The team also realizes the added possibilities that TestNG offers as it is compatible with other browser automation tools such as Selenium as well.

References

GitHub. (2022). *Code Frequency - TestNG - GitHub*. https://github.com/cbeust/testng/graphs/code-frequency

TestNG. (2019). *TestNG*. https://testng.org/doc/index.html

Ammann, O. P. (2022). *Introduction To Software Testing*. CAMBRIDGE INDIA.