What is a unit testing framework? Unit Testing frameworks provides a wireframe for developers/testers to write and execute unit test cases. Through this, individual components or units of software can be tested. These units can be procedures, functions or classes. Additionally, they also provide comprehensive reports of test results. These frameworks are used in every stage of a software development life cycle including requirement gathering, performance optimization and quality assurance. Using frameworks, a software developer/tester can simulate different scenarios to test the program and check if it fails. When integrated properly, it helps software developers/testers ensure that no existing functionality has been amended.

How would a developer utilize a framework? A developer can do so by writing unit tests cases that test the functionality of a component (complex logic). This component can be further broken down into various chunks and each chunk can be evaluated for performance and functionality. A developer can take advantage of unit testing in all stages of the software development life cycle such as requirement gathering, development, optimization and many more. It will help the developer/tester write test cases quicker (familiar with the APIs), execute tests quicker than usual and generate comprehensive performance metrics. The developer is provided with base classes/interfaces that need to be inherited for testing purposes and attributes that can be placed in the program to mark certain functions/classes as test-based functions/classes.

What benefits does a framework provide?

1. Helps with easier and quicker creation of test cases (framework provides base classes/attributes)
2. These frameworks help save time and money in the long run. (existing functionality does not break)
3. These frameworks help in easy and early bug detection and refactoring of source code.
4. Help improve the quality of code. (helps identify defects & makes developer think about edge cases)
5. It helps you understand the performance metrics of your software.

I compare 2 Java-based testing frameworks - JUnit and TestNG. Mentioned below are my comparison results.

Category	**JUnit**	**TestNG**
Parallel Test Runs	Not Supported	Supported
Dependency Tests	Not Supported	Supported
Implementation of Assumptions	Supported	Not Supported
Annotation Support	Included	Included
Test Time Out	Supported	Supported

Part 2

There are 2 Java files.

- HeapSort.java - This class has all the core logic for heapsort.
- HeapSortTest.java - This class has all the test cases for HeapSort.java

HeapSort.java

```
import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;


public class HeapSort {


   void sort(ArrayList<Integer> inputArray)

   {

       int n = inputArray.size();


       for (int i = n / 2 - 1; i >= 0; i--){

           heapify(inputArray, n, i);

       }


       for (int i = n - 1; i > 0; i--) {

           Collections.swap(inputArray, 0, i);

           heapify(inputArray, i, 0);

       }

   }


   void heapify(ArrayList<Integer> inputArray, int n, int i) {

       int largest = i; // Initialize largest as root

       int l = 2 * i + 1; // left = 2*i + 1

       int r = 2 * i + 2; // right = 2*i + 2
```

```
        if (l < n && inputArray.get(l) > inputArray.get(largest))
            largest = l;
        if (r < n && inputArray.get(r) > inputArray.get(largest))
            largest = r;

        if (largest != i) {
            Collections.swap(inputArray, i, largest);
            heapify(inputArray, n, largest);
        }
    }

    // Driver code
    public static void main(String args[])
    {
        ArrayList<Integer> inputArray = new ArrayList<Integer>(Arrays.asList(6,5,3,10,8,9));
        HeapSort ob = new HeapSort();
        ob.sort(inputArray);
        System.out.print(inputArray);
    }
}
```

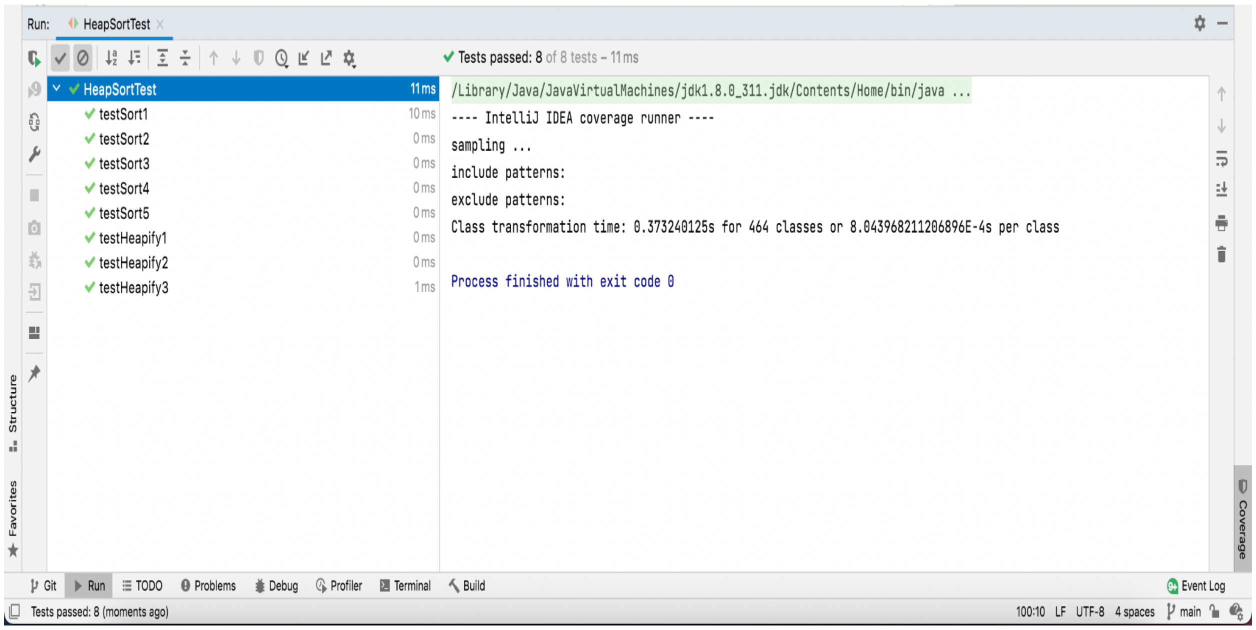
HeapSortTest.java

```
import org.junit.Before;
import org.junit.Test;


import static org.junit.Assert.assertArrayEquals;


import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;


public class HeapSortTest {


   private HeapSort heatSortTester;


   // Test array used in tests

   private final static ArrayList<Integer> testSortArray_1 = new

ArrayList<Integer>(Arrays.asList(9,5,3,6));

   private final static ArrayList<Integer> testSortArray_2 = new

ArrayList<Integer>(Arrays.asList(Integer.MAX_VALUE, Integer.MIN_VALUE));

   private final static ArrayList<Integer> testSortArray_3 = new ArrayList<Integer>();

   private final static ArrayList<Integer> testSortArray_4 = new

ArrayList<Integer>(Collections.singletonList(0));

   private final static ArrayList<Integer> testSortArray_5 = new

ArrayList<Integer>(Arrays.asList(10,5,15,6,7,4,5,2,3,5,2,3,2));


   private final static ArrayList<Integer> testHeapifyArray_1 = new

ArrayList<Integer>(Arrays.asList(Integer.MAX_VALUE, Integer.MIN_VALUE));

   private final static ArrayList<Integer> testHeapifyArray_2 = new

ArrayList<Integer>(Collections.singletonList(0));

   private final static ArrayList<Integer> testHeapifyArray_3 = new

ArrayList<Integer>(Arrays.asList(10,5,15,6,7,4,5,2,3,5,2,3,2));
```

```
@Before
public void setUp() {
    heatSortTester = new HeapSort();
}

@Test
public void testSort1() {
    ArrayList<Integer> result = new ArrayList<Integer>(testSortArray_1);
    Collections.sort(result);

    heatSortTester.sort(testSortArray_1);

    assertArrayEquals(testSortArray_1.toArray(), result.toArray());
}

@Test
public void testSort2() {
    ArrayList<Integer> result = new ArrayList<Integer>(testSortArray_2);
    Collections.sort(result);

    heatSortTester.sort(testSortArray_2);

    assertArrayEquals(testSortArray_2.toArray(), result.toArray());
}

@Test
public void testSort3() {
    ArrayList<Integer> result = new ArrayList<Integer>(testSortArray_3);
    Collections.sort(result);

    heatSortTester.sort(testSortArray_3);
```

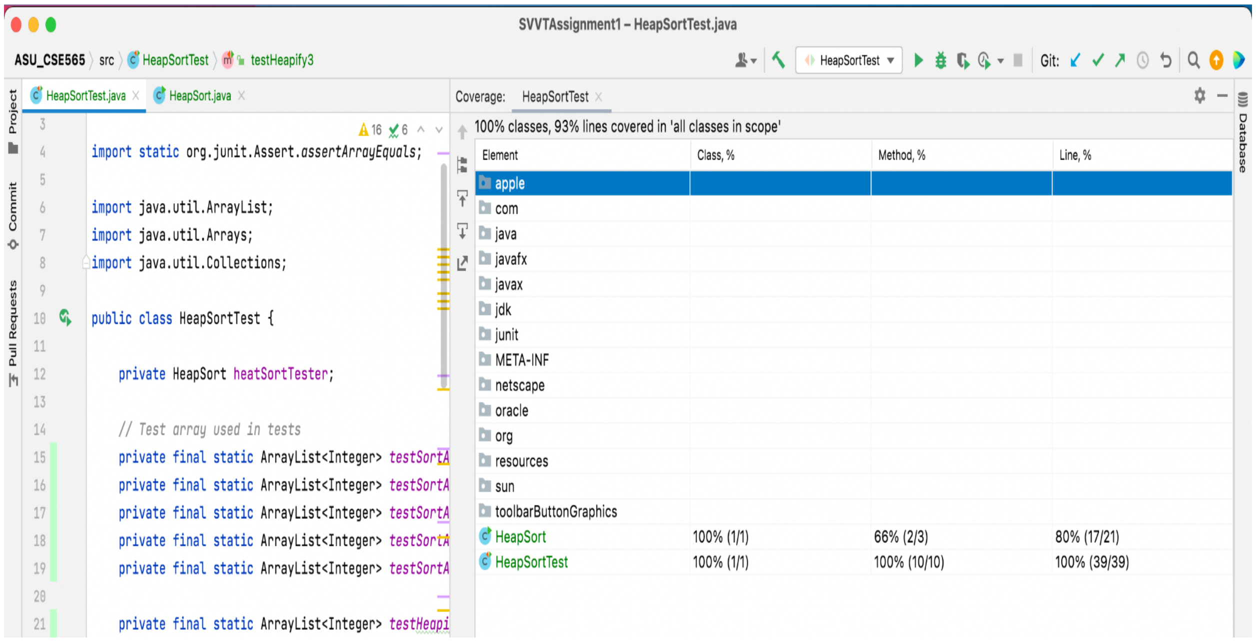
```
        assertArrayEquals(testSortArray_3.toArray(), result.toArray());
    }

    @Test
    public void testSort4() {
        ArrayList<Integer> result = new ArrayList<Integer>(testSortArray_4);
        Collections.sort(result);

        heatSortTester.sort(testSortArray_4);

        assertArrayEquals(testSortArray_4.toArray(), result.toArray());
    }

    @Test
    public void testSort5() {
        ArrayList<Integer> result = new ArrayList<Integer>(testSortArray_5);
        Collections.sort(result);

        heatSortTester.sort(testSortArray_5);

        assertArrayEquals(testSortArray_5.toArray(), result.toArray());
    }

    @Test
    public void testHeapify1() {
        ArrayList<Integer> result = new ArrayList<Integer>(Arrays.asList(Integer.MAX_VALUE,
Integer.MIN_VALUE));

        heatSortTester.heapify(testHeapifyArray_1, result.size(), 0);

        assertArrayEquals(testHeapifyArray_1.toArray(), result.toArray());
    }
    @Test
    public void testHeapify2() {
        ArrayList<Integer> result = new ArrayList<Integer>(testHeapifyArray_2);

        heatSortTester.heapify(testHeapifyArray_2, result.size(), 0);

        assertArrayEquals(testHeapifyArray_2.toArray(), result.toArray());
    }

    @Test
    public void testHeapify3() {
        ArrayList<Integer> result = new
ArrayList<Integer>(Arrays.asList(10,7,15,6,5,4,5,2,3,5,2,3,2));

        heatSortTester.heapify(testHeapifyArray_3, result.size(), 1);

        assertArrayEquals(testHeapifyArray_3.toArray(), result.toArray());
    }
}
```

Output Report



```
Run: HeapSortTest
✓ Tests passed: 8 of 8 tests – 11 ms

✓ HeapSortTest      11 ms
  ✓ testSort1       10 ms
  ✓ testSort2        0 ms
  ✓ testSort3        0 ms
  ✓ testSort4        0 ms
  ✓ testSort5        0 ms
  ✓ testHeapify1     0 ms
  ✓ testHeapify2     0 ms
  ✓ testHeapify3     1 ms

/Library/Java/JavaVirtualMachines/jdk1.8.0_311.jdk/Contents/Home/bin/java ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Class transformation time: 0.373240125s for 464 classes or 8.043968211206896E-4s per class

Process finished with exit code 0

Tests passed: 8 (moments ago)
```

General Test Results - Time per test and Class Transformation Time



SVVTAssignment1 – HeapSortTest.java

ASU_CSE565 〉 src 〉 HeapSortTest 〉 testHeapify3

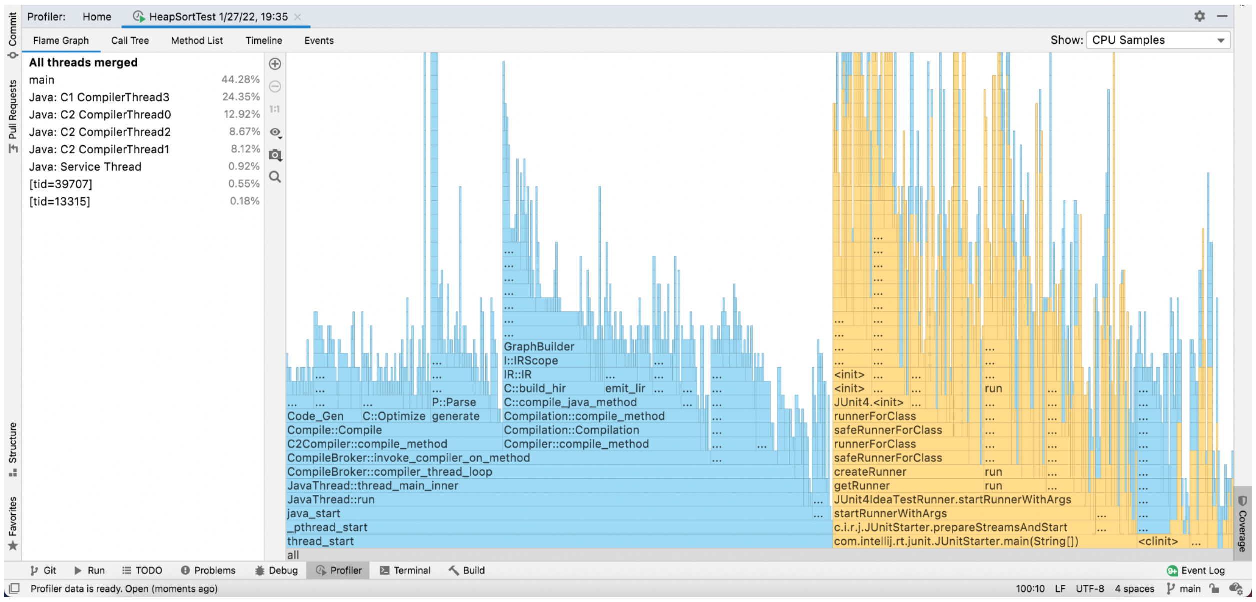
```
import static org.junit.Assert.assertArrayEquals;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class HeapSortTest {

    private HeapSort heatSortTester;

    // Test array used in tests
    private final static ArrayList<Integer> testSortA
    private final static ArrayList<Integer> testSortA
    private final static ArrayList<Integer> testSortA
    private final static ArrayList<Integer> testSortA
    private final static ArrayList<Integer> testSortA

    private final static ArrayList<Integer> testHeapi
```

Coverage: HeapSortTest

100% classes, 93% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
apple			
com			
java			
javafx			
javax			
jdk			
junit			
META-INF			
netscape			
oracle			
org			
resources			
sun			
toolbarButtonGraphics			
HeapSort	100% (1/1)	66% (2/3)	80% (17/21)
HeapSortTest	100% (1/1)	100% (10/10)	100% (39/39)

General Test Results - Coverage Report



Profiler: Home HeapSortTest 1/27/22, 19:35

Flame Graph Call Tree Method List Timeline Events Show: CPU Samples

All threads merged	
main	44.28%
Java: C1 CompilerThread3	24.35%
Java: C2 CompilerThread0	12.92%
Java: C2 CompilerThread2	8.67%
Java: C2 CompilerThread1	8.12%
Java: Service Thread	0.92%
[tid=39707]	0.55%
[tid=13315]	0.18%

Profiler data is ready. Open (moments ago)

General Test Results - Profiler Report

References

1. Tutorialspoint.com. 2022. *UnitTest Framework - Overview*. [online] Available at: <https://www.tutorialspoint.com/unittest_framework/unittest_framework_overview.htm>
2. O'Reilly Online Learning. 2022. *Unit Test Frameworks*. [online] Available at: <https://www.oreilly.com/library/view/unit-test-frameworks/0596006896/ch01.html>

3. 2022. [online] Available at: <https://www.baeldung.com/junit-vs-testng>
4. 2022. [online] Available at: <https://www.softwaretestinghelp.com/junit-vs-testng/>
5. 2022. [online] Available at: <https://en.wikipedia.org/wiki/Unit_testing/>
6. GeeksforGeeks. 2022. *HeapSort - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/heap-sort/>