

# Software Unit Testing Framework Project

Praveen kumar Manoharan

## 1. Development of an Algorithm:

In this project I have implemented Heapsort Algorithm using copilot with my IDE visual Studio code. Heapsort is a comparison-based sorting algorithm uses a binary heap data structure. This program performs sorting in ascending order.

In this algorithm we build max heap from the input array ensuring that the largest element is at the root of the heap. This is achieved by calling the heapify method on each non-leaf node, starting from middle of the array and moving toward the root. Once the max heap is constructed, the algorithm repeatedly swaps the largest value with the last element of the heap, effectively moving it to the correct sorted position. After each swap the heap size is reduced by one, and then repeat the heapify for the remaining of elements. This process continues till entire array is sorted.

This heapify method is responsible for maintaining the maxheap property for subtree rooted at a given index. It compares the root; it swaps them and recursively heapify the affected subtree. This ensures that the largest element always bubbles up to the root of the heap.

```
/**
 * HeapSort is a class that implements the heap sort algorithm.
 * It provides methods to sort an array of integers using the heap sort technique.
 */
public class HeapSort {

    // Main function to sort an array using heap sort
    public void sort(int arr[]) {

        if (arr == null || arr.length == 0) {
            return; // Handle empty or null array
        }

        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract elements from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }
}
```

```

// To heapify a subtree rooted with node i which is an index in arr[], n is size of heap
void heapify(int arr[], 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 left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

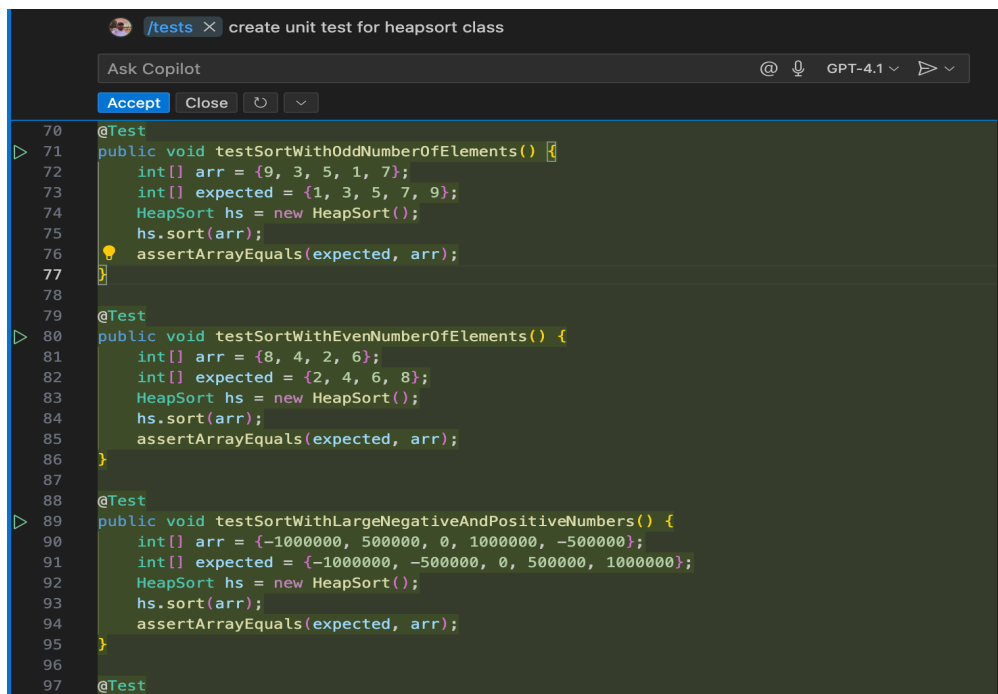
```

## 2. Explanation of Unit Testing framework and prompt generation

I used co-pilot to generate unit test cases for the heapsort algorithm. For this java project I have used junit to unit test the code.

I have prompted co-pilot to “create unit test for heapsort class”, then it gave me 8 test outputs, which are listed and explained in following.

Below screen show shows the out test cases generated by the co-pilot chat.



```

/tests x create unit test for heapsort class
Ask Copilot @ GPT-4.1
Accept Close
70 @Test
71 public void testSortWithOddNumberOfElements() {
72     int[] arr = {9, 3, 5, 1, 7};
73     int[] expected = {1, 3, 5, 7, 9};
74     HeapSort hs = new HeapSort();
75     hs.sort(arr);
76     assertEquals(expected, arr);
77 }
78
79 @Test
80 public void testSortWithEvenNumberOfElements() {
81     int[] arr = {8, 4, 2, 6};
82     int[] expected = {2, 4, 6, 8};
83     HeapSort hs = new HeapSort();
84     hs.sort(arr);
85     assertEquals(expected, arr);
86 }
87
88 @Test
89 public void testSortWithLargeNegativeAndPositiveNumbers() {
90     int[] arr = {-1000000, 500000, 0, 1000000, -500000};
91     int[] expected = {-1000000, -500000, 0, 500000, 1000000};
92     HeapSort hs = new HeapSort();
93     hs.sort(arr);
94     assertEquals(expected, arr);
95 }
96
97 @Test

```

### 3. Explanation of the test cases by AI tool

The AI provided test case are

- testSortWithOddNumberOfElements – test Input array with only odd number and get the necessary sorted odd number only array
- testSortWithEvenNumberOfElements -test Input array with only even number and get the necessary sorted even number array.
- testSortWithLargeNegativeAndPositiveNumbers -test input array with Large Negative and positive Numbers, the sorting executed to validate the negative and then positive numbers
- testSortWithZeros- test Array of zeros to be validated with the Sort, this sorting executed to validate if only zero array is validated and sorted by the program.
- testSortWithAlternatingSigns – test Array with negative and positive number alternative in the array.
- testSortWithSingleNegativeelement -Test for only one negative integer
- testSortWithMaxIntValues- test with maximum int value
- testSortWithMinIntValues - test with min int value.

### 4. Report out of the testcase execution

Executed all test case provided by the tool, the execution was success and shared below the test execution screen shots.

The screenshot displays an IDE with the following components:

- Code Editor:** Shows the `HeapSortTest` class with a single test method `testSortWithOddNumberOfElements()`. The code imports `org.junit.Assert` and `org.junit.Test`, and uses `assertArrayEquals` to validate the sorted array.
- TEST RESULTS Panel:** Lists all test cases executed, including `testSortWithEvenNumberOfElements`, `testSortWithMaxIntValues`, `testSortWithMinIntValues`, `testSortWithZeros`, `testSortWithAlternatingSigns`, `testSortWithSingleNegativeElement`, `testSortWithLargeNegativeAndPositiveNumbers`, and `testSortWithOddNumberOfElements`. Each test is marked as passed with a green checkmark.
- Test Runner for Java Panel:** Provides a summary of the test results, showing the number of tests passed and failed.

```
package mcs.example.project1;

import static org.junit.Assert.*;
import org.junit.Test;

public class HeapSortTest {

    @Test
    public void testSortWithOddNumberOfElements() {
        int[] arr = { 9, 3, 5, 1, 7 };
        int[] expected = { 1, 3, 5, 7, 9 };
        HeapSort hs = new HeapSort();
        hs.sort(arr);
        assertArrayEquals(expected, arr);
    }
}
```

TEST RESULTS

```
%STREET8, testSortWithEvenNumberOfElements(mcs.example.project1.HeapSortTest), false, 1, false, -1
%TESTE9, testSortWithMaxIntValues(mcs.example.project1.HeapSortTest), false, 1, false, -1, testSortWithMinIntValues(mcs.example.project1.HeapSortTest)
%TESTS 2, testSortWithMinIntValues(mcs.example.project1.HeapSortTest)
%TESTE 3, testSortWithZeros(mcs.example.project1.HeapSortTest)
%TESTE 3, testSortWithZeros(mcs.example.project1.HeapSortTest)
%TESTS 4, testSortWithAlternatingSigns(mcs.example.project1.HeapSortTest)
%TESTE 4, testSortWithAlternatingSigns(mcs.example.project1.HeapSortTest)
%TESTS 5, testSortWithOddNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTE 5, testSortWithOddNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTS 6, testSortWithSingleNegativeElement(mcs.example.project1.HeapSortTest)
%TESTE 6, testSortWithSingleNegativeElement(mcs.example.project1.HeapSortTest)
%TESTS 7, testSortWithLargeNegativeAndPositiveNumbers(mcs.example.project1.HeapSortTest)
%TESTE 7, testSortWithLargeNegativeAndPositiveNumbers(mcs.example.project1.HeapSortTest)
%TESTS 8, testSortWithEvenNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTS 9, testSortWithMaxIntValues(mcs.example.project1.HeapSortTest)
%TESTE 9, testSortWithMaxIntValues(mcs.example.project1.HeapSortTest)

%RUNTIME22
```

Test Runner for Java

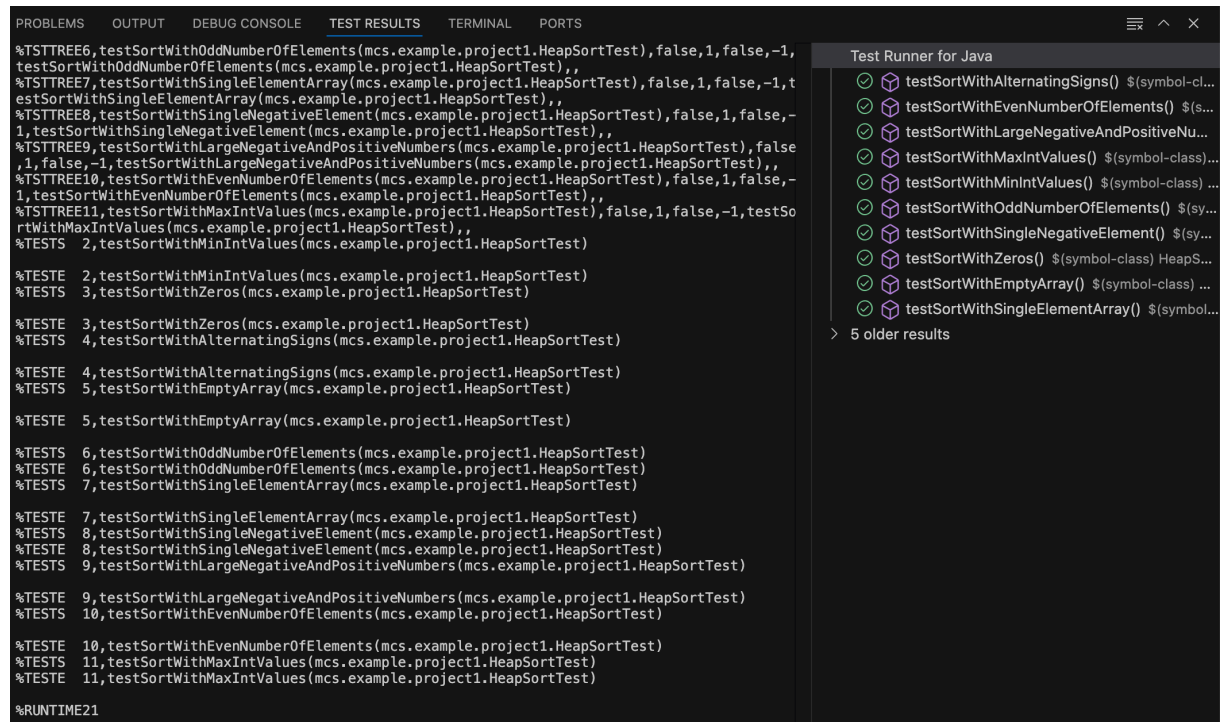
```
testSortWithAlternatingSigns() $(symbol-cl...
testSortWithEvenNumberOfElements() $(s...
testSortWithLargeNegativeAndPositiveNu...
testSortWithMaxIntValues() $(symbol-class)...
testSortWithMinIntValues() $(symbol-class) ...
testSortWithOddNumberOfElements() $(sy...
testSortWithSingleNegativeElement() $(sy...
testSortWithZeros() $(symbol-class) HeapS...
```

> 1 older result

## 5. Assessment and further improvement of test cases

I have improved my unit test case with couple of additional test cases, also these required additional changes in my code to handle null or empty array for my sort method.

1. TestSortWithNullObject – This test is to validate if the array passed to the sort is nullable and software doesn't throw any error, but return empty output.
2. TestSortWithEmptyArray – This test is to validate an empty array as input and the function doesn't break or throw any exceptions, rather handled gracefully.



The screenshot shows an IDE with two panels. The left panel, titled 'TEST RESULTS', displays a list of test cases and their outcomes. The right panel, titled 'Test Runner for Java', shows a list of test methods with green checkmarks indicating they passed.

```
%TSTTREE6, testSortWithOddNumberOfElements(mcs.example.project1.HeapSortTest), false, 1, false, -1,
testSortWithOddNumberOfElements(mcs.example.project1.HeapSortTest),,
%TSTTREE7, testSortWithSingleElementArray(mcs.example.project1.HeapSortTest), false, 1, false, -1, t
estSortWithSingleElementArray(mcs.example.project1.HeapSortTest),,
%TSTTREE8, testSortWithSingleNegativeElement(mcs.example.project1.HeapSortTest), false, 1, false, -
1, testSortWithSingleNegativeElement(mcs.example.project1.HeapSortTest),,
%TSTTREE9, testSortWithLargeNegativeAndPositiveNumbers(mcs.example.project1.HeapSortTest), false
, 1, false, -1, testSortWithLargeNegativeAndPositiveNumbers(mcs.example.project1.HeapSortTest),,
%TSTTREE10, testSortWithEvenNumberOfElements(mcs.example.project1.HeapSortTest), false, 1, false, -
1, testSortWithEvenNumberOfElements(mcs.example.project1.HeapSortTest),,
%TSTTREE11, testSortWithMaxIntValues(mcs.example.project1.HeapSortTest), false, 1, false, -1, testSo
rtWithMaxIntValues(mcs.example.project1.HeapSortTest),,
%TESTS 2, testSortWithMinIntValues(mcs.example.project1.HeapSortTest)
%TESTE 2, testSortWithMinIntValues(mcs.example.project1.HeapSortTest)
%TESTS 3, testSortWithZeros(mcs.example.project1.HeapSortTest)
%TESTE 3, testSortWithZeros(mcs.example.project1.HeapSortTest)
%TESTS 4, testSortWithAlternatingSigns(mcs.example.project1.HeapSortTest)
%TESTE 4, testSortWithAlternatingSigns(mcs.example.project1.HeapSortTest)
%TESTS 5, testSortWithEmptyArray(mcs.example.project1.HeapSortTest)
%TESTE 5, testSortWithEmptyArray(mcs.example.project1.HeapSortTest)
%TESTS 6, testSortWithOddNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTE 6, testSortWithOddNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTS 7, testSortWithSingleElementArray(mcs.example.project1.HeapSortTest)
%TESTE 7, testSortWithSingleElementArray(mcs.example.project1.HeapSortTest)
%TESTS 8, testSortWithSingleNegativeElement(mcs.example.project1.HeapSortTest)
%TESTE 8, testSortWithSingleNegativeElement(mcs.example.project1.HeapSortTest)
%TESTS 9, testSortWithLargeNegativeAndPositiveNumbers(mcs.example.project1.HeapSortTest)
%TESTE 9, testSortWithLargeNegativeAndPositiveNumbers(mcs.example.project1.HeapSortTest)
%TESTS 10, testSortWithEvenNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTE 10, testSortWithEvenNumberOfElements(mcs.example.project1.HeapSortTest)
%TESTS 11, testSortWithMaxIntValues(mcs.example.project1.HeapSortTest)
%TESTE 11, testSortWithMaxIntValues(mcs.example.project1.HeapSortTest)
%RUNTIME21
```

The right panel, 'Test Runner for Java', lists the following test methods, all marked with a green checkmark:

- testSortWithAlternatingSigns() \$(symbol-cl...
- testSortWithEvenNumberOfElements() \$(s...
- testSortWithLargeNegativeAndPositiveNu...
- testSortWithMaxIntValues() \$(symbol-class)...
- testSortWithMinIntValues() \$(symbol-class) ...
- testSortWithOddNumberOfElements() \$(sy...
- testSortWithSingleNegativeElement() \$(sy...
- testSortWithZeros() \$(symbol-class) HeapS...
- testSortWithEmptyArray() \$(symbol-class) ...
- testSortWithSingleElementArray() \$(symbol...

> 5 older results

## 6. An assessment of the generative AI tool

1. Co-pilot helped me in creating unit test quick.
2. Few of the required negative tests were been missed, once it was added to my unit test, I felt good with test coverage and test scenarios covered as part of my unit testing.
3. I felt this helped me in my coding this algorithm and test very quick. Also gave me thought process on few tests like testing for int max and min value.
4. I felt, so the tests created by GEN AI, is not required. And can be written as single test with generic inputs.
  - Eg: testSortWithMaxintValues & testSortwithMinIntValues this can be covered with testSortWithLargeNegativeAndPositiveNumbers test, if the integer max and min values used.
5. Overall, as part of this exercise, I liked use of copilot and wanted to encourage myself in using it for my day-to-day coding to increase, test efficiencies with effective logic testing and covering good amount of edge cases of the code.