

# Utilizing Generative AI for Unit Testing of the Heap Sort Algorithm: An Exhaustive Report

Sumanth Reddy Velagala

Student, School of Computing and Augmented Intelligence, Arizona State University

svelag11@asu.edu

**Abstract**—This report evaluates the implementation of the Heapsort algorithm and its testing using the pytest unit testing framework and the OpenAI GPT-4.0 mini generative AI tool. The study examines the efficiency of AI-generated test cases in covering various scenarios and provides insights into the strengths and limitations of Generative AI tool in software testing.

**Keywords:** Unit testing, Heap Sort Algorithm, Heapify, Generative AI, pytest, Spyder.

## I. INTRODUCTION

Heap Sort algorithm is a sorting technique based on the Binary Heap Data Structure, in which each internal node is greater than or equal to its children. It is a more optimized sorting algorithm than selection sort, where the max element is swapped with the last (or first) element until the array is sorted. The time complexity of heap sort is  $O(n \log n)$ . [1]

Pytest is a unit testing framework for Python, known for its simple syntax and plugins. OpenAI's ChatGPT-4.0 Mini model is a Generative AI model that generates content based on provided prompts.

This study focuses on the usage of a Generative AI model to generate test cases for testing the Heap Sort algorithm using a selected unit testing framework.

## II. DEVELOPMENT OF AN ALGORITHM

Convert the array into a max heap using heapify, and then apply the following steps repeatedly until the heap contains a single element:

1. Swap the largest element of the heap, which is the root node, with the last element of the heap.
2. Eliminate the last element of the heap.
3. Apply heapify to the remaining elements of the heap. [1]

```
def heapify(arr, n, i):  
    largest = i  
    l = 2 * i + 1  
    r = 2 * i + 2  
  
    if l < n and arr[l] > arr[largest]:  
        largest = l  
  
    if r < n and arr[r] > arr[largest]:  
        largest = r  
  
    if largest != i:  
        arr[i], arr[largest] = arr[largest], arr[i]  
        heapify(arr, n, largest)  
  
def heapSort(arr):  
    n = len(arr)  
  
    for i in range(n // 2 - 1, -1, -1):  
        heapify(arr, n, i)  
  
    for i in range(n - 1, 0, -1):  
        arr[0], arr[i] = arr[i], arr[0]  
        heapify(arr, i, 0)
```

Fig - 1 : Implementation of Heap Sort Algorithm in python

## III. EXPLANATION OF THE UNIT TESTING FRAMEWORK AND PROMPT GENERATION

A Unit Testing Framework is a software tool used to run unit tests to verify the correctness of individual components of a program. Individual components can include methods or classes. There are several unit testing frameworks available for each programming language, such as pytest, unittest, nose2, doctest, tox, and hypothesis in Python. Each framework has its own advantages.

In this study, I have used the pytest framework, which is a simple yet compact framework with multiple plugins for HTML. It uses parameter combinations without rewriting test cases. [2]

Key Features of pytest framework include :

1. Auto Discovery of Tests - Detects the prefix or suffix “test” keyword in file naming and executes the file.
2. Parameterization - pooling of multiple test cases using `@mark.parametrize`.
3. Fixtures - Used to setup database connections.
4. Plug-ins - used for parallel computing by using Xdist plug-in.[3]

Generative AI tool is a model that can learn from large amounts of data to create content


such as text, images, music, videos, code based on prompts.[4]

Generative AI tools include ChatGPT, Bard. I have used ChatGPT-4o mini for generating test cases for unit testing of HeapSort Algorithm using the pytest Framework. GPT-4o mini is large language model that can be utilized for content generation and can be integrated using OpenAi API by developers into their applications.

```
def heapify(arr, n, i):  
  
    largest = i  
    l = 2 * i + 1  
    r = 2 * i + 2  
  
    if l < n and arr[l] > arr[largest]:  
        largest = l  
  
    if r < n and arr[r] > arr[largest]:  
        largest = r  
  
    if largest != i:  
        arr[i], arr[largest] = arr[largest], arr[i]  
        heapify(arr, n, largest)  
  
def heapSort(arr):  
  
    n = len(arr)  
  
    for i in range(n // 2 - 1, -1, -1):  
        heapify(arr, n, i)  
  
    for i in range(n - 1, 0, -1):  
        arr[0], arr[i] = arr[i], arr[0]  
        heapify(arr, i, 0)  
  
Generate Test cases for above mentioned HeapSort  
algorithm using pytest framework for unit testing  
covering all scenarios
```

Fig - 2 : Prompt used for generating unit level test cases

#### IV. EXPLANATION OF TEST CASES BY AI TOOL



Here are some test cases for your `heapSort` algorithm using the `pytest` framework. These tests cover various scenarios to ensure that the sorting function behaves as expected:

```
python Copy Edit

import pytest
from typing import List

# Assuming heapSort function is imported above
# from your_module import heapSort

# Test case 1: Sorting an already sorted array
def test_sorted_array():
    arr = [1, 2, 3, 4, 5]
    heapSort(arr)
    assert arr == [1, 2, 3, 4, 5]

# Test case 2: Sorting a reverse-sorted array
def test_reverse_sorted_array():
    arr = [5, 4, 3, 2, 1]
    heapSort(arr)
    assert arr == [1, 2, 3, 4, 5]

# Test case 3: Sorting an array with all elements equal
def test_all_equal_elements():
    arr = [3, 3, 3, 3, 3]
    heapSort(arr)
    assert arr == [3, 3, 3, 3, 3]

# Test case 4: Sorting an empty array
def test_empty_array():
    arr = []
    heapSort(arr)
    assert arr == []
```

Fig - 3 : Result generated by AI tool for provided prompt [5]

```

# Test case 5: Sorting an array with one element
def test_single_element():
    arr = [42]
    heapSort(arr)
    assert arr == [42]

# Test case 6: Sorting a large array
def test_large_array():
    arr = [i for i in range(1000, 0, -1)] # reverse order
    heapSort(arr)
    assert arr == list(range(1, 1001))

# Test case 7: Sorting an array with negative numbers
def test_array_with_negative_numbers():
    arr = [-1, -5, 3, 2, 0, -2]
    heapSort(arr)
    assert arr == [-5, -2, -1, 0, 2, 3]

# Test case 8: Sorting an array with floating-point numbers
def test_array_with_floats():
    arr = [3.14, 2.71, 1.41, 4.2, 2.0]
    heapSort(arr)
    assert arr == [1.41, 2.0, 2.71, 3.14, 4.2]

# Test case 9: Sorting a large array with random values
def test_random_values():
    arr = [24, 18, 31, 7, 2, 16, 9, 5]
    heapSort(arr)
    assert arr == [2, 5, 7, 9, 16, 18, 24, 31]

# Test case 10: Sorting a large array with random negative values
def test_random_negative_values():
    arr = [-3, -1, -7, -2, -4, -9]
    heapSort(arr)
    assert arr == [-9, -7, -4, -3, -2, -1]

```

Fig - 4 : Result generated by AI tool for provided prompt [5]

#### Test Case explanation

1. **test\_sorted\_array:** Tests sorting an array which is already sorted.
2. **test\_reverse\_sorted\_array:** Tests sorting an array which is in reverse order.
3. **test\_all\_equal\_elements:** Tests sorting an array which has all equal elements.
4. **test\_empty\_array:** Tests sorting an array which is empty.
5. **test\_single\_element:** Tests sorting an array which contains a single element.
6. **test\_large\_array:** Tests sorting a large array which is initially in reverse order.
7. **test\_array\_with\_negative\_numbers:** Tests sorting an array which contains negative numbers.
8. **test\_array\_with\_floats:** Tests sorting an array which has floating-point numbers.
9. **test\_random\_values:** Tests sorting an array which has random values.

10. **test\_random\_negative\_values:** Tests sorting an array which has random negative values.

## V. REPORT OUT OF TEST CASE EXECUTION

The Heap Sort algorithm tested using pytest unit testing framework has successfully passed all the test cases in 0.17 seconds. Below is a snippet of the console.

```
In [11]: runfile('/Users/sumanth/Documents/ASU/SVVP/test_heapify.py',
wdir='/Users/sumanth/Documents/ASU/SVVP')
===== test session starts =====
platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/sumanth/Documents/ASU/SVVP
plugins: anyio-4.2.0, typeguard-4.3.0
collected 10 items

test_heapify.py .....
[100%]

===== 10 passed in 0.17s =====
```

Fig - 5 : Execution Result

## VI. ASSESSMENT AND FURTHER IMPROVEMENT OF TEST CASES

The test cases generated by the generative AI tool has efficiently tested the Heap Sort Algorithm with 100 % execution covering most of the cases. However, the below mentioned test cases were not included, which will test the robustness of the heap sort algorithm :

1. Characters ['h','a','d','b','z']
2. Mixed data types [25, -6, 5.5, 0]
3. Characters with mixed case sensitivity ['J', 'D', 'z', 'G', 'a']
4. Lexicographical order of strings ['strawberry', 'apple', 'orange', 'grapes']
5. Special characters ['%', '@', '!', '<', '(']

```
# New Test Cases
def test_characters():
    arr = ['h','a','d','b','z']
    heapSort(arr)
    assert arr == ['a','b','d','h','z']

def test_mixed_data_types():
    arr = [25, -6, 5.5, 0]
    heapSort(arr)
    assert arr == [-6, 0, 5.5, 25]

def test_case_sensitivity():
    arr = ['J', 'D', 'z', 'G', 'a']
    heapSort(arr)
    assert arr == ['D', 'G', 'J', 'a', 'z']

def test_Lexicographical():
    arr = ['strawberry', 'apple', 'orange', 'grapes']
    heapSort(arr)
    assert arr == ['apple', 'grapes', 'orange', 'strawberry']

def test_Special_Char():
    arr = ['%', '@', '!', '<', '(']
    heapSort(arr)
    assert arr == ['!', '%', '(', '<', '@']
```

Fig - 6 : Improved test cases

```
In [2]: runfile('/Users/sumanth/Documents/ASU/SVVP/test_heapify.py',
wdir='/Users/sumanth/Documents/ASU/SVVP')
Reloaded modules: test_heapify
===== test session starts =====
platform darwin -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
rootdir: /Users/sumanth/Documents/ASU/SVVP
plugins: anyio-4.2.0, typeguard-4.3.0
collected 15 items

test_heapify.py .....
[100%]

===== 15 passed in 0.00s =====
```

Fig - 7 : Execution of test cases including the improved test cases

## VII. ASSESSMENT OF THE GENERATIVE AI TOOL

The Chat GPT-4.0 mini model is very useful for developers and testers in terms of creating standard test cases for unit testing. However, re-validating those test cases is recommended to make sure the test cases cover all the criteria and incorporate complex test cases for effective testing. In this report, we have observed that the tool have efficiently covered sorting an already sorted array, sorting a reversed sorted array, sorting an array with the same equal numbers, sorting an empty array, sorting an array with only one element, sorting a very large array, sorting an array with negative numbers, and sorting an array with floating-point numbers. However, the tool didn't include sorting characters, mixed data types, case-sensitive characters, sorting strings, or sorting special characters.

## REFERENCES

- [1] GeeksforGeeks, "Heap Sort," <https://www.geeksforgeeks.org/heap-sort/> (accessed Jan. 27, 2025).
- [2] BrowserStack, "Top Python Testing Frameworks," <https://www.browserstack.com/guide/top-python-testing-frameworks> (accessed Jan. 27, 2025).
- [3] DZone, "10 Awesome Features of Pytest," <https://dzone.com/articles/10-awesome-features-of-pytest> (accessed Jan. 27, 2025).
- [4] Harvard University, "AI at Harvard," <https://huit.harvard.edu/ai> (accessed Jan. 27, 2025).
- [5] ChatGPT, "Shareable Chat," <https://chatgpt.com/share/67946e08-fb0c-8005-a279-2387a8d8517d> (accessed Jan. 27, 2025).