Report

Introduction

The testing efforts were centered on sortalgo.py, a Python package that includes a variety of sorting algorithms such as Bubble Sort, Selection Sort, Merge Sort, and more. This module was created from scratch, with no preceding codebase, and adheres to current Python3 standards and syntax. Furthermore, the sorting algorithms' capabilities have been improved to accommodate more complex techniques for rigorous testing.

Sorting Algorithms

The sortalgo.py file contains a set of sorting algorithms written in Python3. All of the algorithms were thoroughly tested using various techniques. The following is a list of all algorithms present in the file.

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Heap Sort
7. Shell Sort
8. Counting Sort
9. Radix Sort
10. Bucket Sort

Scope

The testing on the sortalgo.py file uses a wide variety of techniques which include unit tests, mutations and TSTL testing.
The unit tests were done using the Python Unittest package which already comes preinstalled with the base Python3 installation.
The constraints for the unit tests are as follows:

1. Typical case: Tests the sorting algorithm with a typical list of integers.
2. Reversed list: Tests the sorting algorithm with a list sorted in descending order.
3. List with duplicates: Tests the sorting algorithm with a list containing duplicate elements.
4. Empty list: Tests the sorting algorithm with an empty list.
5. Single element list: Tests the sorting algorithm with a list containing only one element.
6. List with negative numbers: Tests the sorting algorithm with a list containing negative integers.

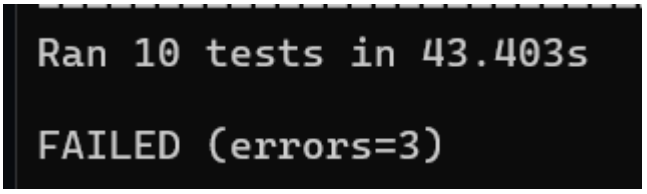
7. Large numbers: Tests the sorting algorithm with a list containing large integers.
8. List of tuples: Tests the sorting algorithm with a list of tuples.
9. All elements are the same: Tests the sorting algorithm with a list where all elements are the same.
10. List with zeros and negative numbers: Tests the sorting algorithm with a list containing both zeros and negative integers.
11. List containing both very small and very large numbers: Tests the sorting algorithm with a list containing very small and very large numbers.
12. Random list of integers: Tests the sorting algorithm with a randomly generated list of integers.
13. List in descending order: Tests the sorting algorithm with a list sorted in descending order.
14. List with mostly repeated elements: Tests the sorting algorithm with a list containing mostly repeated elements.
15. List of floats: Tests the sorting algorithm with a list of floating-point numbers.
16. List of strings: Tests the sorting algorithm with a list of strings.
17. List containing both very small and very large floats: Tests the sorting algorithm with a list containing very small and very large floating-point numbers.

Methodologies

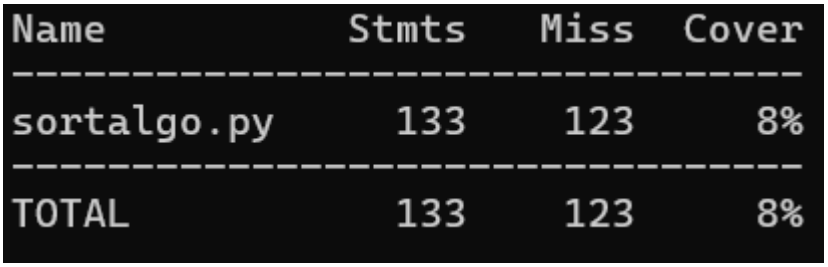
Three primary testing methodologies were incorporoted in the testing process.

1. Unit Tests

A package containing all the test cases was written from scratch called sortalgotest.py. The package contains a wide variety of tests which are tested on all the sorting algorithms. Since the primary goal of all the algorithms is to return a sorted list of elements, all of the cases were tested on all the algorithms with the same results being expected.



2. Coverage



Testing the coverage on the sorting algorithms proved to be very difficult due to the sorting algorithms such as quick sort, merge sort, and radix sort having multiple branches and

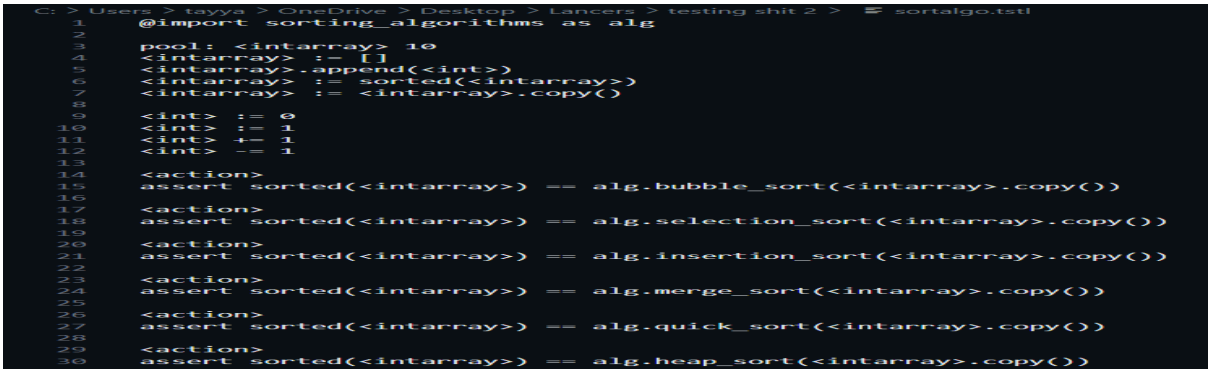
recursive calls. Achieving full path coverage requires carefully designed test cases for each possible path, which can be non-trivial due to the recursive and iterative nature of these algorithms.

Mutations

```
263 VALID MUTANTS
131 INVALID MUTANTS
18 REDUNDANT MUTANTS
Valid Percentage: 63.834951456310679%
```

The mutation testing using universalmutator showed that the testing was valid and covered majority cases. As can be seen, the number of valid mutants and the percentage is fairly high, which proves that the testing achieved positive results.

TSTL



TSTL testing was also performed which yield positive results.

Conclusion

In conclusion, our testing technique, which includes unittest, universalmutator, coverage, and TSTL, was thorough and informative. Unittest allowed for systematic testing of each sorting algorithm across a variety of input scenarios. Universalmutator identified potential vulnerabilities by introducing code mutations. The coverage analysis identified opportunities for improvement in our test suite. TSTL automates test case generation, which increases productivity. Together, these measures have improved the reliability and trustworthiness of sorting algorithms' accuracy and performance.