**Prog985t**

**(Merge Sort Unit Testing)**

**Program Description:** Implement the Merge Sort algorithm and write comprehensive unit tests to ensure its correctness and efficiency. This assignment is designed to deepen your understanding of sorting algorithms and the importance of testing in software development. Your program should have two scripts: “**prog985t**” and “**test\_prog985t**” (you can also add Merge Sort to your sorting library from Prog408aSort).

**Requirements:**

**Sorting Algorithm:** Implement the Merge Sort algorithm.It should sort a list of numbers in ascending order. Write it as a helper class with two methods: **sort** and **merge**.

**Unit Tests:** Write unit tests to validate the correctness of your sorting algorithm.Tests should cover various cases, including:

* Normal cases with typical lists of numbers.
* Edge cases like an empty list, a list with one element, and a list where all elements are identical.
* Lists with negative numbers and mixed integer/float values.
* Test for performance (e.g., time taken to sort) for large datasets.

**Data Location:** unit tests (see next page)

**Sample Output:**

TestMergeSort.test\_empty\_list: 0.000013

TestMergeSort.test\_identical\_elements: 0.000019

TestMergeSort.test\_mixed\_types: 0.000012

TestMergeSort.test\_negative\_numbers: 0.000010

TestMergeSort.test\_normal\_case: 0.000011

TestMergeSort.test\_performance\_large\_dataset: 4.566719

TestMergeSort.test\_single\_element: 0.000033

----------------------------------------------------------------------

Ran 7 tests in 4.569s

OK

**Write the following test cases (from Pseudocode):**

TestMergeSort:

// Test with a typical list of numbers

test\_normal\_case():

input = [4, 2, 5, 1, 3]

expected\_output = [1, 2, 3, 4, 5]

assert MergeSort.sort(input) == expected\_output

// Test with an empty list

test\_empty\_list():

input = []

expected\_output = []

assert MergeSort.sort(input) == expected\_output

// Test with a list having one element

test\_single\_element():

input = [1]

expected\_output = [1]

assert MergeSort.sort(input) == expected\_output

// Test with a list where all elements are identical

test\_identical\_elements():

input = [5, 5, 5, 5]

expected\_output = [5, 5, 5, 5]

assert MergeSort.sort(input) == expected\_output

// Test with negative numbers in the list

test\_negative\_numbers():

input = [-3, -1, -4, -2]

expected\_output = [-4, -3, -2, -1]

assert MergeSort.sort(input) == expected\_output

// Test with mixed integer and float values

test\_mixed\_types():

input = [3.2, 1.5, 4.8, 2.1]

expected\_output = [1.5, 2.1, 3.2, 4.8]

assert MergeSort.sort(input) == expected\_output

// (Optional) Test for performance on a large dataset

test\_performance\_large\_dataset():

input = generate\_large\_random\_list()

start\_time = current\_time()

MergeSort.sort(input)

end\_time = current\_time()

assert (end\_time - start\_time) < time\_threshold # try 10 seconds

// Helper function to generate a large random list

generate\_large\_random\_list():

// Implementation depends on the language and specific requirements

// Generate a list of 10 million random numbers between 1 and 1 million

return large\_random\_list