**Task 1**

For Task 1, the **best case time complexity for the new function sum\_items() is O(1),** which occurs when the list is empty. While the **worse case time complexity is O(N),** which occurs when the list, with size n, is not empty, and n items is iterated to be summed up.

Test Cases for sum\_items()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Description | Test Steps | Expected Results | Result |
| 1 | Testing sum\_items() | Create a list with valid numerical values, then call sum\_items() with this list as input.  [1,2,3,4,6] | It should return the sum of all elements in the list.  [1,2,3,4,6] = 16 | Pass |
| 2 | Testing large inputs | Call sum\_items() with an input of 10000 valid numerical value in the list. | It should return the sum of all elements in the list.  Range(0,10001) = 50005000 | Pass |
| 3 | Testing invalid values | Call sum\_items() with alphabetical inputs in the list | It should return error message: “Please only insert numerical type lists.” | Pass |
| 4 | Testing empty list. | Call sum\_items() with an empty list | It should return 0. | Pass |
| 5 | Testing real values. | Call sum\_items() with a list of real numbers. E.g  [0.1,0.3,0.5,0.1] | It should correctly return the sum of the numbers.  [0.1,0.3,0.5,0.1] = 1.0 | Pass |

**Task 2**

sum\_items()

Worst Case - O(N): When summing all values of a list with N items

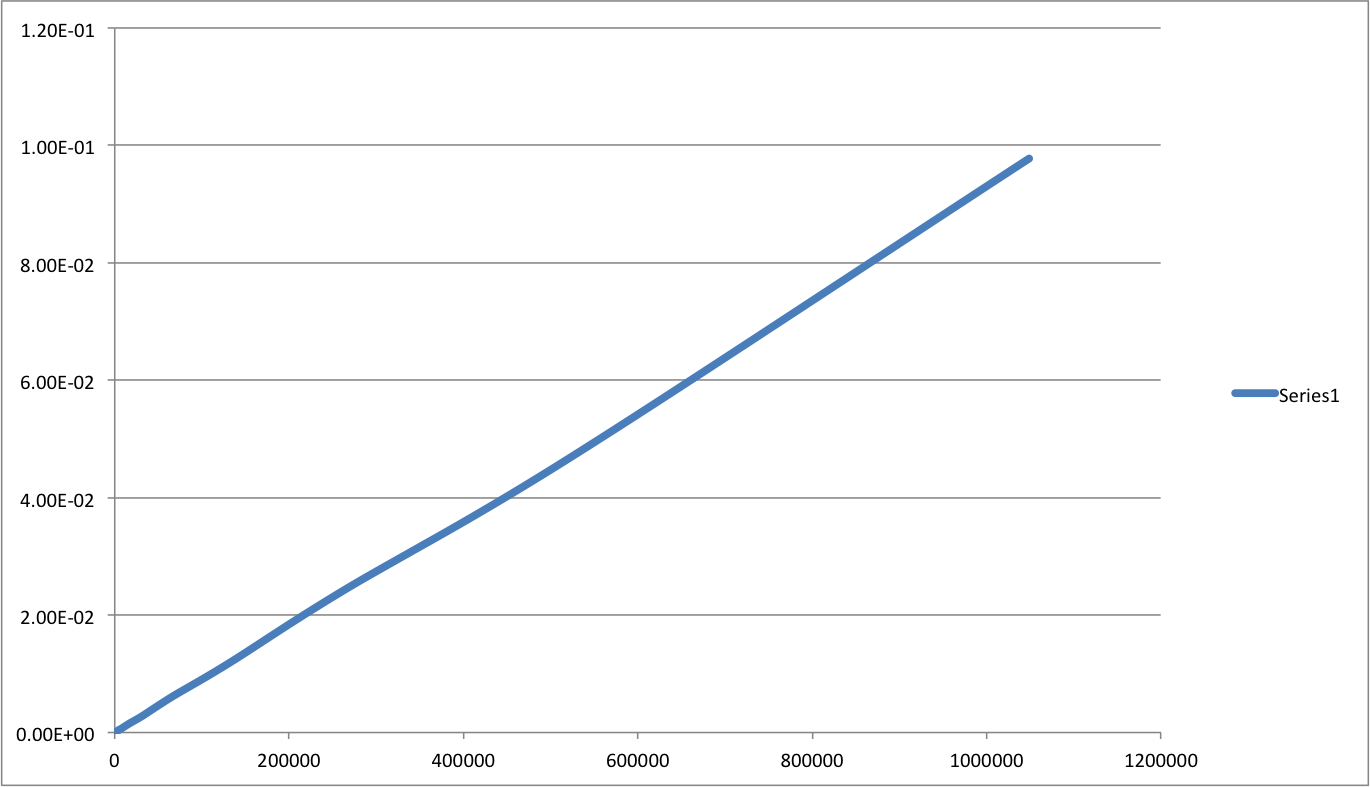
Best Case - O(1): when passing empty list

table\_time\_sum\_items():

**Best Case: O(log n): When the list is generated, then go through log n iterations**

**Worst Case: O(N log n): When the algorithm loops through log n iteration and sums up n items.**

In table\_time\_sum\_items(), the algorithm will always generate N items, then go through log(N) iterations to sum up N items. Hence it’s O(N log N)



**Task 3**

Test Cases for sum\_until\_negative()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Description | Test Steps/Data | Expected Results | Result |
| 1 | A list of valid positive numerical values | [1,2,3,4] | It should return a sum of 10. | Pass |
| 2 | An empty list | [] | It should return 0. | Pass |
| 3 | An list with first item as negative. | [-1,2,3] | It should return 0. | Pass |
| 4 | A list mixed negative and positive integers | [1,3,-2,4] | It should a sum of 4, showing that it stopped when it reached -2. | Pass |
| 5 | A list with invalid value. | [“a”] | It should return an error asking the user to input a numerical list. | Pass |

**sum\_until\_negative():**

Worst Case - O(N): goes through the whole list when none is negative

Best Case - O(1): empty list or first is negative

Task 3-2

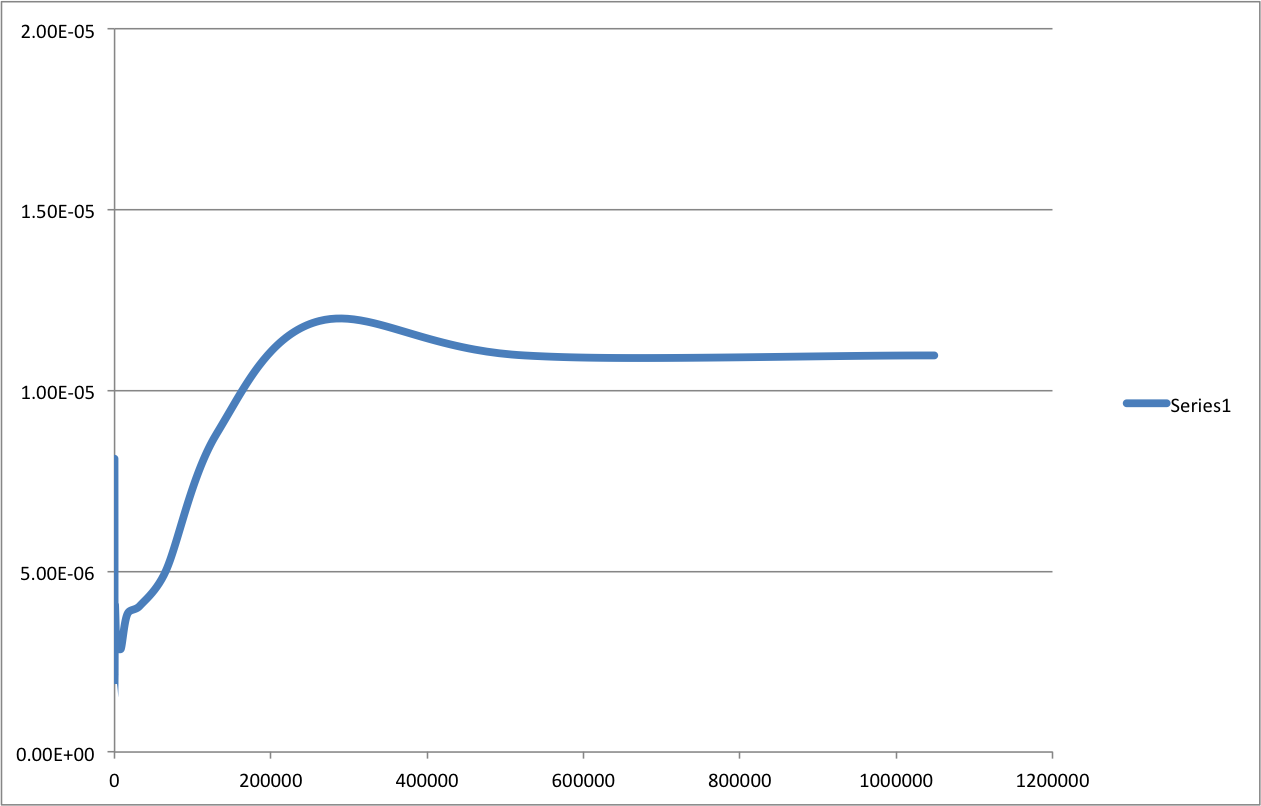
Shown below is the graph for the program with table\_time\_sum\_until\_negative\_1(), which has a worst case time complexity of O(n log n), which occurs when none of the items in list is negative.

table\_time\_sum\_until\_negative\_1():

**Worst Case: O(n log n) :** when none of the item in list is negative.

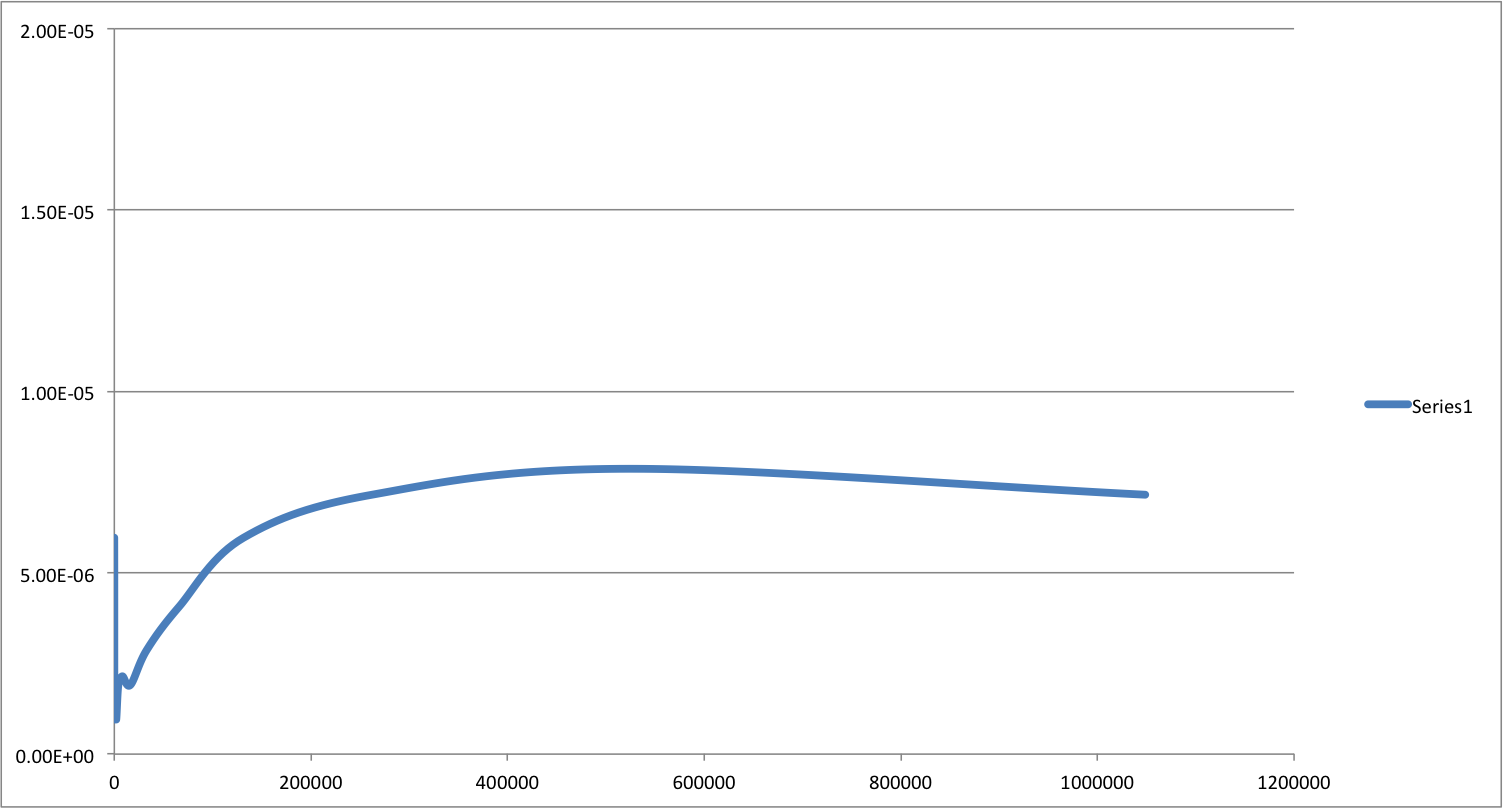
**Best Case Time Complexity: O(N): when the first item is negative.** Because the algorithm always generate N items, then loop through log n items, then stop when it encounters the negative item.

Compared with Task 2’s graph, this one is more “flattened” throughout the later part, as the algorithm has encountered a negative number and only sums up to that negative number after that encounter. Best case time complexity would be O(N), when the first item is negative and the algorithm stops at the first item for all n.

****

Task 3-3

Shown below is the graph for table\_time\_sum\_until\_negative\_2(), which has a **best case and worst case time complexity of O(log N),** because the algorithm will always go through to generate n real numbers, then go through log n iterations, and then halt at the first input which is negative.



**Advanced Question**

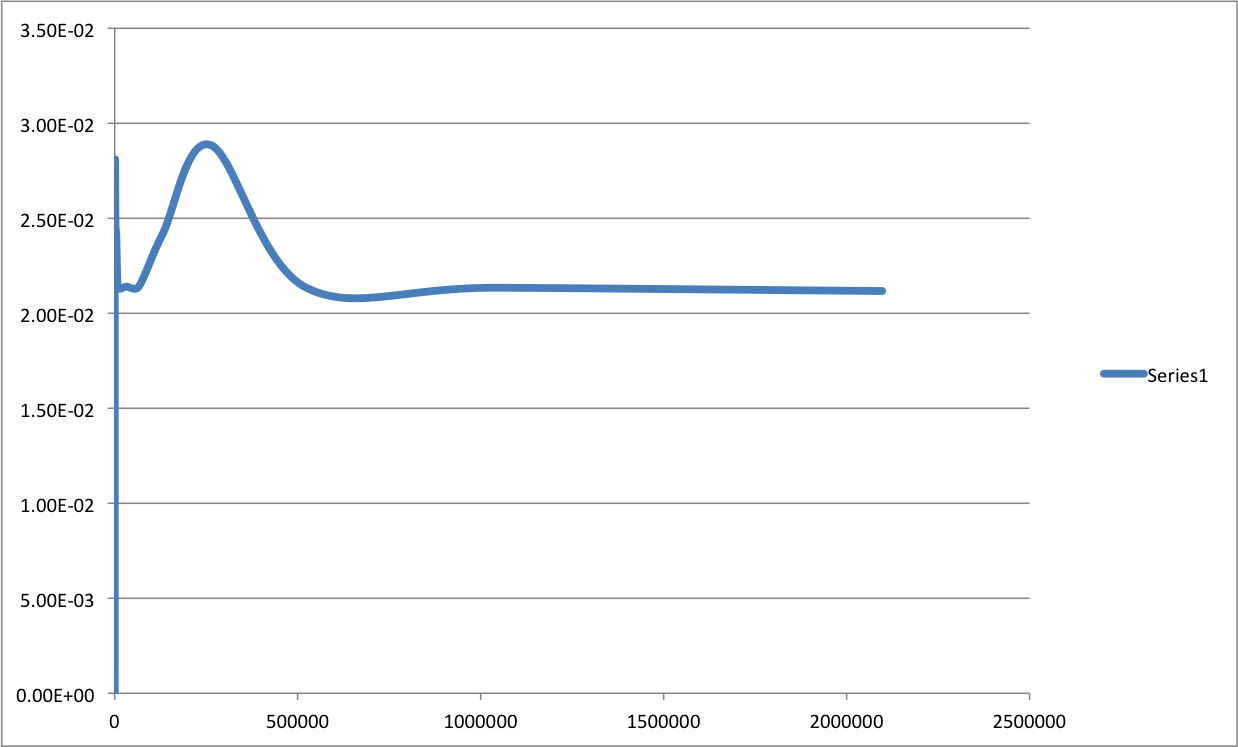
**Advanced Question & Hall of Fame Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Description | Test Steps/Data | Expected Results | Result |
| 1 | A list of valid positive numerical values | [1,2,3,4] | It should return (0,3). | Pass |
| 2 | A list of valid values consisting of negative and positive real numbers | [0.5, -2, 1.5, 3, 9, -3] | It should return (2,4).  Since 1.5+3+9 will yield 13.5 which is the max. | Pass |
| 3 | An empty list. | [] | It should return an error asking the user to input a list. | Pass |
| 4 | A list with invalid items, such as alphabets | [1, “b”, “c”, 5] | It should return an error asking the user to input a numerical list. | Pass |
| 5 | A list with an empty item. | [“”] | It should return an error asking the user to input a numerical list. | Pass |
| 6 | A list consisting of only negative numbers. | [-3,-4,-2,-1] | It should return the largest magnitude negative number’s index. In this case: (3,3) for [-1] | Pass |

find\_max\_sum\_interval()

Worst Case Time Complexity: O(N^3), because the algorithm always go through every single subset in the list to look for the maximum sum interval by doing two nested loops, then call sum\_items() which is O(N).

Best Case: O(1), when the list is empty.



quick\_find\_max\_sum\_interval()

**Worst Case Time Complexity: O(N),** which occurs when the algorithm goes through the whole list.

**Best Case Time Complexity: O(1),** when the list is empty.

