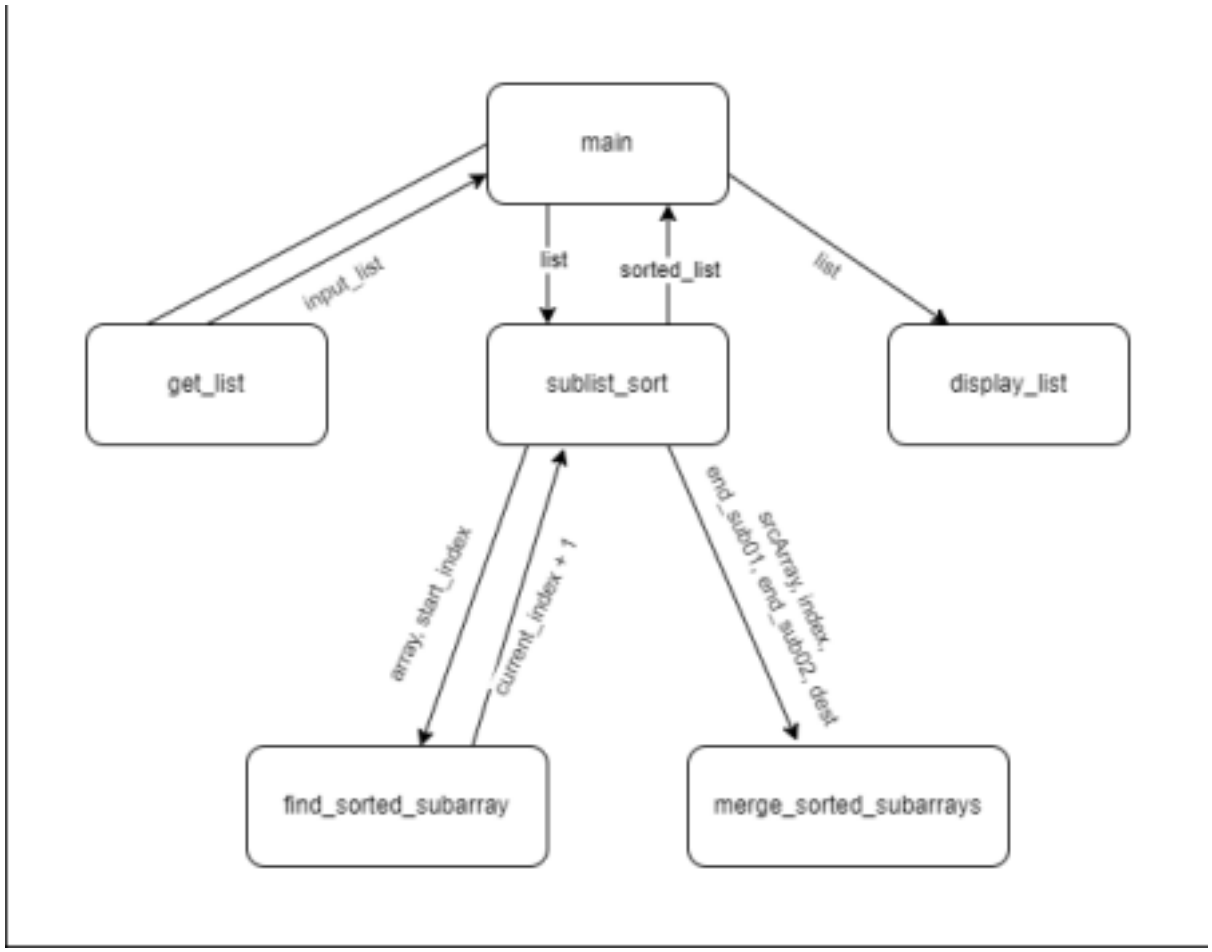
Structure Chart:



Pseudocode:

```
   FUNCTION find_sorted_subarray(array, start_index)                                    # O(n)
     current_index = start_index                                                        # O(1)

     WHILE current_index < (length(array) - 1) AND array[current_index] <= array[current_index + 1]   # O(n)
         current_index += 1                                                             # O(n)
     RETURN current_index + 1                                                           # O(1)

 FUNCTION merge_sorted_subarrays(sourceArray, start_index, end_of_subarray01, end_of_subarray02, destination)# O(n)
     current_index_in_sub01 = start_index                                               # O(1)
     current_index_in_sub02 = end_of_subarray01                                         # O(1)
     destination_index = start_index                                                    # O(1)

     WHILE current_index_in_sub01 < end_of_subarray01 AND current_index_in_sub02 < end_of_subarray02    # O(n)
         IF sourceArray[current_index_in_sub01] <= sourceArray[current_index_in_sub02]  # O(n)
             destination[destination_index] = sourceArray[current_index_in_sub01]       # O(n)
             current_index_in_sub01 += 1                                                # O(n)
         ELSE
             destination[destination_index] = sourceArray[current_index_in_sub02]       # O(n)
             current_index_in_sub02 += 1                                                # O(n)

         destination_index += 1                                                         # O(n)

     WHILE current_index_in_sub01 < end_of_subarray01                                   # O(n)
         destination[destination_index] = sourceArray[current_index_in_sub01]           # O(n)
         current_index_in_sub01 += 1                                                    # O(n)
         destination_index += 1                                                         # O(n)

     WHILE current_index_in_sub02 < end_of_subarray02                                   # O(n)
         destination[destination_index] = sourceArray[current_index_in_sub02]           # O(n)
         current_index_in_sub02 += 1                                                    # O(n)
```

```
        destination_index += 1                                                          # O(n)

 FUNCTION sublist_sort(array)                                                           # O(1)
    temp_array = Array equal to the size of the original array                          # O(1)
    array_size = length(array)                                                          # O(1)
    is_sorted = FALSE                                                                   # O(1)

    WHILE NOT is_sorted                                                                 # O(log n)
        is_sorted = TRUE                                                                # O(log n)
        start_index = 0                                                                 # O(log n)

        WHILE start_index < array_size                                                  # O(n log n)
            end_of_subarray01 = find_sorted_subarray(array, start_index)                # O(n log n)
            end_of_subarray02 = find_sorted_subarray(array, end_of_subarray01)          # O(n log n)

            IF end_of_subarray02 > array_size
                end_of_subarray02 = array_size                                          # O(n log n)

            merge_sorted_subarrays(array, start_index, end_of_subarray01, end_of_subarray02, temp_array)# O(n log n)
            IF end_of_subarray02 != array_size
                is_sorted = FALSE                                                       # O(n log n)
            start_index = end_of_subarray02                                             # O(n log n)
        SWAP array with temp_array                                                      # O(n log n)
    RETURN array                                                                        # O(n log n)


FUNCTION run_test_cases()
    # Test cases an array of arrays input and expected output
    test_inputs = [
        [6, 12, 11]         # Trace example Unsorted
        [3, 2, 1, 5, 4],    # Normal Unsorted
        [1, 2, 3, 4, 5],    # Already Sorted
        [5, 4, 3, 2, 1],    # Reverse Sorted
        [1],                # Single Element
```

```
    [],                 # Empty Array
    [2, 2, 2, 2, 2],    # All elements the same
    ["witch", "pumpkin", "ghost", "vampire", "zombie"] # Array of Strings Unsorted
]
expected_outputs = [
    [1, 2, 3, 4, 5],
    [1, 2, 3, 4, 5],
    [1, 2, 3, 4, 5],
    [1],
    [],
    [2, 2, 2, 2, 2],
    ["ghost", "pumpkin", "vampire", "witch", "zombie"]
]

# Loop through each test case using its index
FOR i from 0 to length(test_inputs) - 1
    # Sort the input array using the sublist_sort function
    sorted_array = sublist_sort(test_inputs[index])

    # Assert that the sorted_array matches the expected output
    ASSERT sorted_array == expected_outputs[index] # "Test failed for input: " + test_inputs[i]

    # If the assert does not fail, display a success message
    PUT "Test passed for input: ", test_inputs[index]
```

Modularization Metrics

Include a copy of the structure chart you are working off of. This could be the structure chart you produced last week, or it could be the key presented by the instructor. For every function in the structure chart, determine the cohesion and coupling level. Give a brief (1–2 sentence) justification why the level is what you think it is.

Cohesion:

- FUNCTION find_sorted_subarray
  - The cohesion I feel is strong because the function does exactly what it should. There aren’t any extraneous or extra code. This function does exactly what the name implies.
- FUNCTION merge_sorted_subarray
  - The cohesion I feel is strong because the function does exactly what it should. There aren’t any extraneous or extra code. This function does exactly what the name implies.
- FUNCTION sublist_sort
  - The cohesion I feel is strong because all the steps in the function contribute directly to the sorting process. This function does exactly what the name implies.

Coupling:

- FUNCTION find_sorted_subarray
  - The function does not depend on any other functions. It takes an array and an index and returns a value based only on those inputs. So the coupling is simple and weak.
- FUNCTION merge_sorted_subarray
  - It takes specific indices and arrays as inputs and operates independently of any external functions. It doesn't modify the sourceArray but rather uses the data to merge into destination. So the coupling is simple and weak.
- FUNCTION sublist_sort
  - This function is moderately coupled because it relies on find_sorted_subarray and merge_sorted_subarray.

Algorithmic Metrics

Overall efficiency = O(n log n)
Most of the program runs O(n)
The sort function is O(log n)
The Find subarray function is O(n) because the worse case would require each element has to be visited each loop
The merge is O(n) for the same reason it has to visit each element

Test Cases

Identify a collection of test cases for your program. For each test case, identify the provided input and expected

output.

```
FUNCTION run_test_cases()
    # Test cases an array of arrays input and expected output
    test_inputs = [
        [6, 12, 11]                                             # Trace example Unsorted (required)
        [3, 2, 1, 5, 4],                                        # Normal Unsorted (required)
        [1, 2, 3, 4, 5],                                        # Already Sorted (required)
        [5, 4, 3, 2, 1],                                        # Reverse Sorted (required)
        [1],                                                    # Single Element (boundary)
        [],                                                     # Empty list (error)
        [42, 33],                                               # Empty Array (boundary)
        [2, 2, 2, 2, 2],                                        # All elements the same (error)
        [3, "apple", 2, "banana"],                              # Mixed values(error)
        ["witch", "pumpkin", "ghost", "vampire", "zombie"]      # Array of Strings Unsorted (required)
        [7, 91, 80, 56, 84, 32, 64, 88, 74, 79, 67, 1, 70, 75, 33, 17, 30, 8, 50, 71], # Large list (boundary)
    ]

    expected_outputs = [
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1],
        [],
        [33, 42],
        [2, 2, 2, 2, 2],
        [3, "apple", 2, "banana"],
        ["ghost", "pumpkin", "vampire", "witch", "zombie"],
        [1, 7, 8, 17, 30, 32, 33, 50, 56, 64, 67, 70, 71, 74, 75, 79, 80, 84, 88, 91],
    ]

    # Loop through each test case using its index
    FOR i from 0 to length(test_inputs) - 1
        # Sort the input array using the sublist_sort function
```

```
sorted_array = sublist_sort(test_inputs[index])

# Assert that the sorted_array matches the expected output
ASSERT sorted_array == expected_outputs[index] # "Test failed for input: " + test_inputs[i]

# If the assert does not fail, display a success message
PUT "Test passed for input: ", test_inputs[index]
```

Trace Verification

Line	Array/src	temp_array/dest	array_size	is_sorted	start_index	end_of_subarray01	end_of_subarray02	current_index	current_index_in_sub01	current_index_in_sub02	destination_index
33	[6, 12, 11]	\	\	\	\	\	\	\	\	\	\
34	[6, 12, 11]	[Null, Null, Null]	\	\	\	\	\	\	\	\	\
35	[6, 12, 11]	[Null, Null, Null]	3	\	\	\	\	\	\	\	\
36	[6, 12, 11]	[Null, Null, Null]	3	FALSE	\	\	\	\	\	\	\
37	[6, 12, 11]	[Null, Null, Null]	3	FALSE	\	\	\	\	\	\	\
38	[6, 12, 11]	[Null, Null, Null]	3	FALSE	\	\	\	\	\	\	\
39	[6, 12, 11]	[Null, Null, Null]	3	TRUE	\	\	\	\	\	\	\

40	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	\	\	\	\	\	\
41	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	\	\	\	\	\	\
42	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	\	\	\	\	\	\
43	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	1	\	\	\
1	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	\	\	\	\
2	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	0	\	\	\
4	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	0	\	\	\
5	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	1	\	\	\
44	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	3	1	\	\	\
1	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	1	\	\	\
2	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\

4	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
5	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
49	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
8	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	\	\
9	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	\	\
10	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	\
11	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	0
13	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	0
14	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	0
15	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	0	2	0
16	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	0

21	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	1
13	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	1
14	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	1
18	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	2	1
19	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	1
21	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	2
13	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	2
14	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	2
18	[6, 12, 11]	[6, 11, 12]	3	TRUE	2	2	3	2	1	3	2
19	[6, 12, 11]	[6, 11, 12]	3	TRUE	2	2	3	2	1	3	2
21	[6, 12, 11]	[6, 11, 12]	3	TRUE	2	2	3	2	1	3	3

53	[6, 11, 12]	[6, 12, 11]	3	TRUE	2	2	3	2	1	3	3

Here is a more complete view of the trace:

Line	Array/src	temp_array/dest	array_size	is_sorted	start_index	end_of_subarray01	end_of_subarray02	current_index	current_index_in_sub01	current_index_in_sub02	destination_index
33	[6, 12, 11]	\	\	\	\	\	\	\	\	\	\
34	[6, 12, 11]	[Null, Null, Null]	\	\	\	\	\	\	\	\	\
35	[6, 12, 11]	[Null, Null, Null]	3	\	\	\	\	\	\	\	\
36	[6, 12, 11]	[Null, Null, Null]	3	FALSE	\	\	\	\	\	\	\
37	[6, 12, 11]	[Null, Null, Null]	3	FALSE	\	\	\	\	\	\	\
38	[6, 12, 11]	[Null, Null, Null]	3	FALSE	\	\	\	\	\	\	\
39	[6, 12, 11]	[Null, Null, Null]	3	TRUE	\	\	\	\	\	\	\
40	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	\	\	\	\	\	\
41	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	\	\	\	\	\	\
42	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	\	\	\	\	\	\
43	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	1	\	\	\
1	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	\	\	\	\
2	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	0	\	\	\
4	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	0	\	\	\
5	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	\	1	\	\	\
44	[6, 12, 11]	[Null, Null, Null]	3	TRUE	0	2	3	1	\	\	\
1	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	1	\	\	\
2	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
4	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
5	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
49	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	\	\	\
8	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	\	\
9	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	\	\
10	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	\
11	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	0
13	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	0
14	[6, 12, 11]	[Null, Null, Null]	3	TRUE	2	2	3	2	0	2	0
15	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	0	2	0
16	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	0
21	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	1
13	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	1
14	[6, 12, 11]	[6, Null, Null]	3	TRUE	2	2	3	2	1	2	1
18	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	2	1
19	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	1
21	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	2
13	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	2
14	[6, 12, 11]	[6, 11, Null]	3	TRUE	2	2	3	2	1	3	2
18	[6, 12, 11]	[6, 11, 12]	3	TRUE	2	2	3	2	1	3	2
19	[6, 12, 11]	[6, 11, 12]	3	TRUE	2	2	3	2	1	3	2
21	[6, 12, 11]	[6, 11, 12]	3	TRUE	2	2	3	2	1	3	3
53	[6, 11, 12]	[6, 12, 11]	3	TRUE	2	2	3	2	1	3	3

Submission Notes

- How long did it take for you to complete this assignment?
  - 10 Hours
- What was the hardest part of the assignment?
  - The hardest part of this was both finding the efficiency and the trace. The trace was by far the hardest with how my program is setup with functions within functions. It took a long time to complete and it was hard to not make mistakes while writing the trace.
- Was there anything unclear about the instructions on how you were to complete this lab?
  - The lab was very clear. The instructions were clear and understandable.