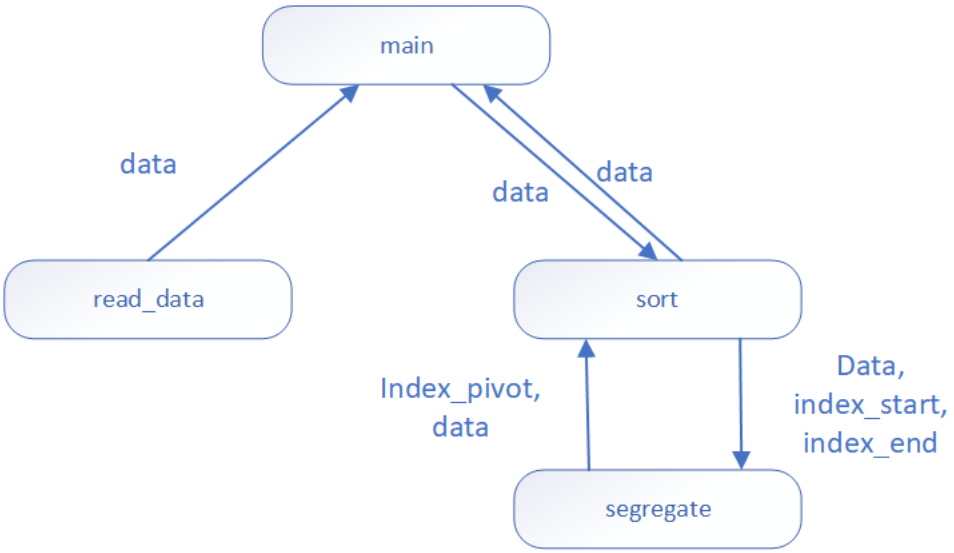
Segregation Sort Analysis

Modularization Metrics:



Read Function:

- It has strong cohesion because it's only reading an array and returning it to the main function.
- The coupling is encapsulated because all the information exchanged is in a convenient form and is guaranteed to be in a valid state.

Sort Function:

- It has strong cohesion because it's completing the one task of organizing the array in order of smallest to largest.
- The coupling is encapsulated because all the information exchanged is in a convenient form and is guaranteed to be in a valid state.

Segregate Function:

- The segregate function has strong cohesion because all the tasks in the function are designed to swap values around based on index location and value comparison then return the lists as one main list.
- The coupling for this function is complex because some of the parameters require knowledge about them in order to make connections.

Algorithmic Metrics:

```
1  Segregate(data, index_start, index_end)
2     SET original_index_start = index_start
3     SET original_index_end = index_end
4
5     IF index_start = index_end
6           RETURN index_start
7
8     index_pivot <- (index_start + index_end) / 2
9
10    WHILE index_start < index_end
11          WHILE data[index_start] <= data[index_pivot]
12                index_start += 1
13          WHILE data[index_end] >= data[index_pivot]
14                index_end -= 1
15
16          IF index_start < index_end
17                SWAP data[index_start], data[index_end]
18
19    IF index_start > index_pivot AND data[index_start] > data[index_pivot]
20          index_start -= 1
21
22    IF array[index_start] > pivot_value:
23          SWAP array[index_start - 1] WITH array[index_pivot]
24          SET index_pivot = index_start - 1
25    ELSE:
26          SWAP array[index_start] WITH array[index_pivot]
27          SET index_pivot = index_start
28
29    CALL segregate(array, original_index_start, index_pivot - 1)
30    CALL segregate(array, index_pivot + 1, original_index_end)
31
32    RETURN index_pivot, data
```

Line 2: O(1) copying a value into a variable
Line 3: O(1) copying a value into a variable
Line 5 - 6: O(1) simple check statement comparing single values and returning a value to caller.
Line 8: O(1) simple calculation and storing the value into a variable
Line 10 - 17: O(n) the time taken to run this portion is dependent on the size of the parameters being used.
Line 19 - 20: O(1) simple check statement that takes no extra time to execute and increments a single value by 1
Line 22 - 27: O(1) simple check statement that takes no extra time, swaps values in the array based on index location, and sets a variable's value.

Line 29 and 30: O(n log n) and O(n^2) the complexity is dependent on how the pivot splits the array. If the pivot is at the end of the array then the complexity is O(n^2) but if it splits the array evenly the complexity switches to O(n log n)
Line 32: O(1) returning value of pivot and array

Test:

```
test_cases()

      # Empty
      assert sort([]) == []

      # Single Item
      assert sort([15]) == [15]

      # Multiple Items
      assert sort([1, 10, 8, 3, 5, 9, 6]) == [1, 3, 5, 6, 8, 9, 10]

      # Largest Value in Middle
      assert sort([1, 3, 8, 10, 5, 9, 6]) == [1, 3, 5, 6, 8, 9, 10]

      # Smallest Value in Middle
      assert sort([10, 3, 8, 1, 5, 9, 6]) == [1, 3, 5, 6, 8, 9, 10]

      # Duplicate Values
      assert sort([1, 3, 8, 10, 5, 9, 3]) == [1, 3, 3, 5, 8, 9, 10]
```