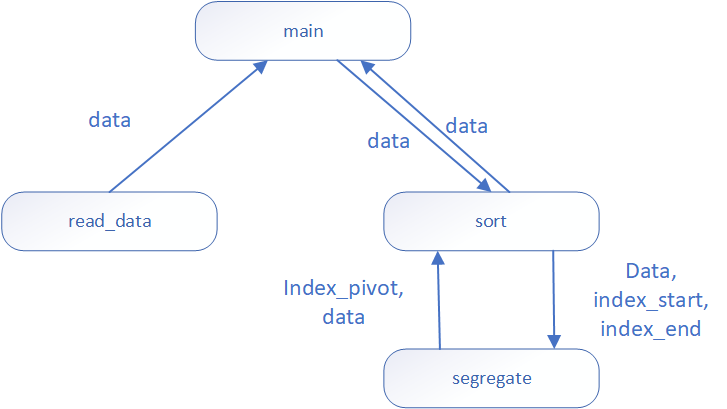
Segregation Sort Analysis

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]