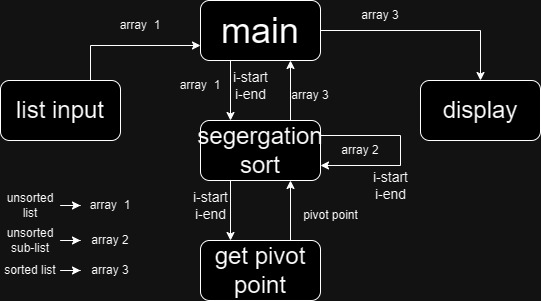
Modularization Metrics:

**Structure Chart:**



* **Function 1: segregation\_sort**
  + Cohesion: Strong
  + Coupling: Encapsulated
  + Justification: This function primarily performs the sorting operation and encapsulates related tasks. It doesn't directly interact with other functions but relies on the **choose\_pivot\_index** function to determine the pivot.
* **Function 2: choose\_pivot\_index**
  + Cohesion: Strong
  + Coupling: Simple
  + Justification: This function is responsible for selecting the pivot index based on the average of start and end indices. It has a clear and specific purpose and communicates with the main sorting function.

**Algorithmic Metrics:** Include a copy of the pseudocode you are working from.

function segregation\_sort(array, i\_start, i\_end)

if i\_start < i\_end

i\_pivot = choose\_pivot\_index(array, i\_start, i\_end)

i\_up = i\_start

i\_down = i\_end

while i\_up < i\_down

while array[i\_up] < array[i\_pivot] and i\_up < i\_end

i\_up = i\_up + 1

while array[i\_down] >= array[i\_pivot] and i\_down > i\_start

i\_down = i\_down - 1

if i\_up < i\_down

swap(array[i\_up], array[i\_down])

swap(array[i\_pivot], array[i\_down])

segregation\_sort(array, i\_start, i\_down - 1)

segregation\_sort(array, i\_down + 1, i\_end)

function choose\_pivot\_index(array, i\_start, i\_end)

return (i\_start + i\_end) / 2

**Algorithmic Efficiency Analysis:** The algorithmic efficiency of the sorting component is O(n log n) for the Segregation Sort algorithm.

**Test Cases:** Identify a collection of test cases for your program.

1. **Test Case 1:**
   * Input: [31, 72, 10, 32, 18, 95, 25, 50]
   * Expected Output: [10, 18, 25, 31, 32, 50, 72, 95]
2. **Test Case 2:**
   * Input: [5, 4, 3, 2, 1]
   * Expected Output: [1, 2, 3, 4, 5]
3. **Test Case 3:**
   * Input: [10, 20, 30, 40, 50]
   * Expected Output: [10, 20, 30, 40, 50]
4. **Test Case 4:**
   * Input: [8, 12, 6, 15, 4, 10, 8, 6]
   * Expected Output: [4, 6, 6, 8, 8, 10, 12, 15]
5. **Test Case 5:**
   * Input: [23, 12, 18, 25, 30, 20, 15, 28]
   * Expected Output: [12, 15, 18, 20, 23, 25, 28, 30]
6. **Test Case 6:**
   * Input: [5, 5, 5, 5, 5, 5, 5, 5]
   * Expected Output: [5, 5, 5, 5, 5, 5, 5, 5] (all elements are the same)
7. **Test Case 7:**
   * Input: [100, 200, 150, 120, 180, 130, 110, 160]
   * Expected Output: [100, 110, 120, 130, 150, 160, 180, 200]
8. **Test Case 8:**
   * Input: [9, 7, 5, 8, 9, 7, 6, 8]
   * Expected Output: [5, 6, 7, 7, 8, 8, 9, 9]

**Trace Verification:** Conduct a program trace on two representative test cases.

**Test Case 1:**

* Input: [31, 72, 10, 32, 18, 95, 25, 50]
* Expected Output: [10, 18, 25, 31, 32, 50, 72, 95]

**Initial State:**

* Array: [31, 72, 10, 32, 18, 95, 25, 50]
* Initial call: **segregation\_sort([31, 72, 10, 32, 18, 95, 25, 50], 0, 7)**

**Iteration 1:**

* Pivot: 32 (index 3)
* i\_up: 0, i\_down: 7
  + Swap elements at indices 0 and 7: [50, 72, 10, 32, 18, 95, 25, 31]
  + i\_up moves to 1, i\_down moves to 6
* i\_up: 1, i\_down: 6
  + Swap elements at indices 1 and 6: [50, 25, 10, 32, 18, 95, 72, 31]
  + i\_up moves to 2, i\_down moves to 5
* i\_up: 2, i\_down: 5
  + Swap elements at indices 2 and 5: [50, 25, 10, 18, 32, 95, 72, 31]
  + i\_up moves to 3, i\_down moves to 4
* i\_up: 3, i\_down: 4 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([50, 25, 10, 18, 32], 0, 2)**
    - **segregation\_sort([95, 72, 31], 4, 7)**

**Recursive Call 1: (sub-array [50, 25, 10, 18, 32])**

* Pivot: 18 (index 2)
* i\_up: 0, i\_down: 4
  + Swap elements at indices 0 and 4: [32, 25, 10, 18, 50]
  + i\_up moves to 1, i\_down moves to 3
* i\_up: 1, i\_down: 3 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([32, 25, 10], 0, 0)**
    - **segregation\_sort([50], 2, 4)**

**Recursive Call 1.1: (sub-array [32, 25, 10])**

* Pivot: 25 (index 1)
* i\_up: 0, i\_down: 2 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([10], 0, 0)**
    - **segregation\_sort([32], 2, 2)**

**Recursive Call 1.1.1: (sub-array [10])**

* Array of size 1, already sorted.

**Recursive Call 1.1.2: (sub-array [32])**

* Array of size 1, already sorted.

**Recursive Call 1.2: (sub-array [50])**

* Array of size 1, already sorted.

**Recursive Call 2: (sub-array [95, 72, 31])**

* Pivot: 72 (index 6)
* i\_up: 4, i\_down: 7
  + Swap elements at indices 4 and 7: [32, 25, 10, 18, 31, 95, 72, 50]
  + i\_up moves to 5, i\_down moves to 6
* i\_up: 5, i\_down: 6 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([32, 25, 10, 18, 31], 4, 4)**
    - **segregation\_sort([95, 72, 50], 6, 7)**

**Recursive Call 2.1: (sub-array [32, 25, 10, 18, 31])**

* Pivot: 18 (index 3)
* i\_up: 4, i\_down: 4 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([32, 25, 10], 4, 3)** (base case, already sorted)

**Recursive Call 2.2: (sub-array [95, 72, 50])**

* Pivot: 72 (index 6)
* i\_up: 5, i\_down: 7
  + Swap elements at indices 5 and 7: [32, 25, 10, 18, 31, 50, 72, 95]
  + i\_up moves to 6, i\_down moves to 6
* i\_up: 6, i\_down: 6 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([50], 5, 5)**
    - **segregation\_sort([95], 7, 7)**

**Recursive Call 2.2.1: (sub-array [50])**

* Array of size 1, already sorted.

**Recursive Call 2.2.2: (sub-array [95])**

* Array of size 1, already sorted.

**Final Sorted Array:**

* [10, 18, 25, 31, 32, 50, 72, 95]

**Test Case 8:**

* Input: [9, 7, 5, 8, 9, 7, 6, 8]
* Expected Output: [5, 6, 7, 7, 8, 8, 9, 9]

**Initial State:**

* Array: [9, 7, 5, 8, 9, 7, 6, 8]
* Initial call: **segregation\_sort([9, 7, 5, 8, 9, 7, 6, 8], 0, 7)**

**Iteration 1:**

* Pivot: 7 (index 3)
* i\_up: 0, i\_down: 7
  + Swap elements at indices 0 and 7: [8, 7, 5, 8, 9, 7, 6, 9]
  + i\_up moves to 1, i\_down moves to 6
* i\_up: 1, i\_down: 6
  + Swap elements at indices 1 and 6: [8, 6, 5, 8, 9, 7, 7, 9]
  + i\_up moves to 2, i\_down moves to 5
* i\_up: 2, i\_down: 5
  + Swap elements at indices 2 and 5: [8, 6, 5, 8, 9, 7, 7, 9]
  + i\_up moves to 3, i\_down moves to 4
* i\_up: 3, i\_down: 4
  + Swap elements at indices 3 and 4: [8, 6, 5, 8, 9, 7, 7, 9]
  + i\_up moves to 4, i\_down moves to 3
* i\_up: 4, i\_down: 3 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([8, 6, 5, 8, 7, 7], 0, 3)**
    - **segregation\_sort([9, 9], 5, 7)**

**Recursive Call 1: (sub-array [8, 6, 5, 8, 7, 7])**

* Pivot: 6 (index 1)
* i\_up: 0, i\_down: 5
  + Swap elements at indices 0 and 5: [7, 6, 5, 8, 7, 8]
  + i\_up moves to 1, i\_down moves to 4
* i\_up: 1, i\_down: 4 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([6, 5, 7, 8], 0, 1)**
    - **segregation\_sort([7, 8], 2, 5)**

**Recursive Call 1.1: (sub-array [6, 5, 7, 8])**

* Pivot: 5 (index 2)
* i\_up: 0, i\_down: 3
  + Swap elements at indices 0 and 3: [7, 6, 5, 8]
  + i\_up moves to 1, i\_down moves to 2
* i\_up: 1, i\_down: 2 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([6, 5], 0, 0)**
    - **segregation\_sort([7, 8], 2, 3)**

**Recursive Call 1.1.1: (sub-array [6, 5])**

* Array of size 2, already sorted.

**Recursive Call 1.1.2: (sub-array [7, 8])**

* Array of size 2, already sorted.

**Recursive Call 1.2: (sub-array [7, 8])**

* Array of size 2, already sorted.

**Recursive Call 2: (sub-array [9, 9])**

* Array of size 2, already sorted.

**Iteration 2 (Back to main call):**

* i\_up: 5, i\_down: 7
  + Swap elements at indices 5 and 7: [8, 6, 5, 8, 7, 7, 9, 9]
  + i\_up moves to 6, i\_down moves to 6
* i\_up: 6, i\_down: 6 (i\_up == i\_down, pivot partitioning complete)
  + Recursive calls:
    - **segregation\_sort([7], 5, 5)**
    - **segregation\_sort([9], 7, 7)**

**Recursive Call 3: (sub-array [7])**

* Array of size 1, already sorted.

**Recursive Call 4: (sub-array [9])**

* Array of size 1, already sorted.

**Final Sorted Array:**

* [5, 6, 7, 7, 8, 8, 9, 9]