Lab 07 : Sort Design

Step 1: By Hand

List = [45, 12, 78, 34, 56, 89, 23, 67]
Largest: 89
Swap position with: 67

List = [45, 12, 78, 34, 56, 67, 23, 89]
New Largest: 78
Swap position with: 23

List = [45, 12, 23, 34, 56, 67, 78, 89]
New Largest: 67
Swap position with: 56

List = [45, 12, 23, 34, 56, 67, 78, 89]
Largest: 56
Swap position with: 34

List = [45, 12, 23, 34, 56, 67, 78, 89]
Largest: 45
Swap position with: 34

List = [34, 12, 23, 45, 56, 67, 78, 89]
Largest: 34
Swap position with: 23

List = [23, 12, 34, 45, 56, 67, 78, 89]
Largest: 23
Swap position with: 12
List = [12, 23, 34, 45, 56, 67, 78, 89]

The list is now sorted

Step 2: Approach

To solve this sorting problem, I will repeatedly search for the largest unsorted element in the list and swap it with the last position in the unsorted portion of the list. Starting from the full list, I will compare each element to find the current largest and move it to its sorted position at the end. This process will continue by narrowing the unsorted section of the list from right to left until the entire list is sorted in ascending order. This approach ensures each element is placed in its correct position by focusing on the largest remaining value in each pass.

Step 3: Pseudocode

```
SET Length_of_list  TO length OF numbers - 1
SET unsorted_index TO length_of_list
     While unsorted > 0 DO
          SET index_largest TO 0
          FOR current_index FROM 1 TO unsorted_index:
               IF numbers[current_index] > numbers[index_largest] THEN:
                    SET index_largest TO current_index
          SWAP numbers[index_largest] WITH numbers[unsorted_index]
          DECREASE unsorted_index BY 1
     END WHILE
     PUT numbers
```

Step 4: Microsoft Copilot

```
SortList(array)
    n = length(array)
    FOR i = n – 1 down TO 1
        max_index = 0
        FOR j = 1 TO i
            IF array[j] > array[max_index]
                max_index = j
        End FOR
        SWAP array[max_index} WITH array[i]
    END FOR
```

Step 5: Compare and Contrast

My Solution:

Pros: I use descriptive variable names like length_of_list, unsorted_index, and index_largest, which make my code easier to understand. The unsorted_index variable clearly marks the unsorted boundary, showing where each largest element will be placed.

Cons: My code could be more concise, and the line DECREASE unsorted_index BY 1 could be simplified.

Copilot's Solution:

Pros: Copilot’s structure is concise and uses a simple descending loop with i representing the boundary of the unsorted section. The logic is clear, and max_index effectively tracks the largest element.

Cons: Variable names like i and j are less descriptive than those in my solution, which could make the logic harder to follow.

How I Can Improve My Solution

I could adopt Copilot's loop structure, using FOR i from n minus 1 down to 1, for a simpler layout and improved clarity.

How Copilot's Solution Can Be Improved

Copilot’s code would be more readable with descriptive variable names, like using unsorted_index for i and index_of_largest for max_index.

Matching the Algorithm in Step 1

Both versions match the Selection Sort algorithm performed by hand, finding the largest element in the unsorted portion and placing it at the end, progressively shrinking the unsorted boundary.

<u>Step 6: Update</u>

```
SET length_of_list TO length OF numbers
SET unsorted_index TO length_of_list - 1
     WHILE unsorted_index > 0 DO
     SET index_of_largest TO 0
     FOR current_index FROM 1 TO unsorted_index DO
          IF numbers[current_index] >
          numbers[index_of_largest] THEN
          SET index_of_largest TO current_index
     END FOR
     SWAP numbers[index_of_largest] WITH
     numbers[unsorted_index]
     SET unsorted_index TO unsorted_index - 1
END WHILE
RETURN numbers
```

Step 7: Trace

Pass	Initial Array	Largest Found (Index)	Swap Position (Index)	Array After Swap
1	[26, 6, 90, 55]	90 (Index 2)	3	[26, 6, 55, 90]
2	[26, 6, 55, 90]	55 (Index 2)	2	[26, 6, 55, 90]
3	[26, 6, 55, 90]	26 (Index 0)	1	[6, 26, 55, 90]
4	[6, 26, 55, 90]	-	-	[6, 26, 55, 90]

Step 8: Efficiency

CODE TO ANALYZE:

```
SET length_of_list TO length OF numbers
SET unsorted_index TO length_of_list - 1
    WHILE unsorted_index > 0 DO
    SET index_of_largest TO 0
    FOR current_index FROM 1 TO unsorted_index DO
        IF numbers[current_index] >
        numbers[index_of_largest] THEN
        SET index_of_largest TO current_index
    END FOR
    SWAP numbers[index_of_largest] WITH
    numbers[unsorted_index]
    SET unsorted_index TO unsorted_index - 1
END WHILE
RETURN numbers
```

EFFICIENCY:

The efficiency for this program is <u>O(N^2)</u>. It has two loops, each contributing O(N) individually, but since they are nested, they build on top of each other, resulting in O(N^2). This is because the inner loop runs N times for each pass of the outer loop, making the <u>nested loops</u> effectively <u>N * N = O(N^2)</u>. The outer loop iterates over each element in the list, and the inner loop finds the largest unsorted element in each pass. Since <u>each element is processed multiple times</u>, the program has an overall efficiency of <u>O(N^2)</u>, as it iterates through the list and sorts it through comparisons and swaps.