Step 1: By Hand

2	5	8	6	1	3	4	7
2	5	7	6	1	3	4	8
2	5	4	6	1	3	7	8
2	5	4	3	1	6	7	8
2	1	4	3	5	6	7	8
2	1	3	4	5	6	7	8
2	1	3	4	5	6	7	8
1	2	3	4	5	6	7	8

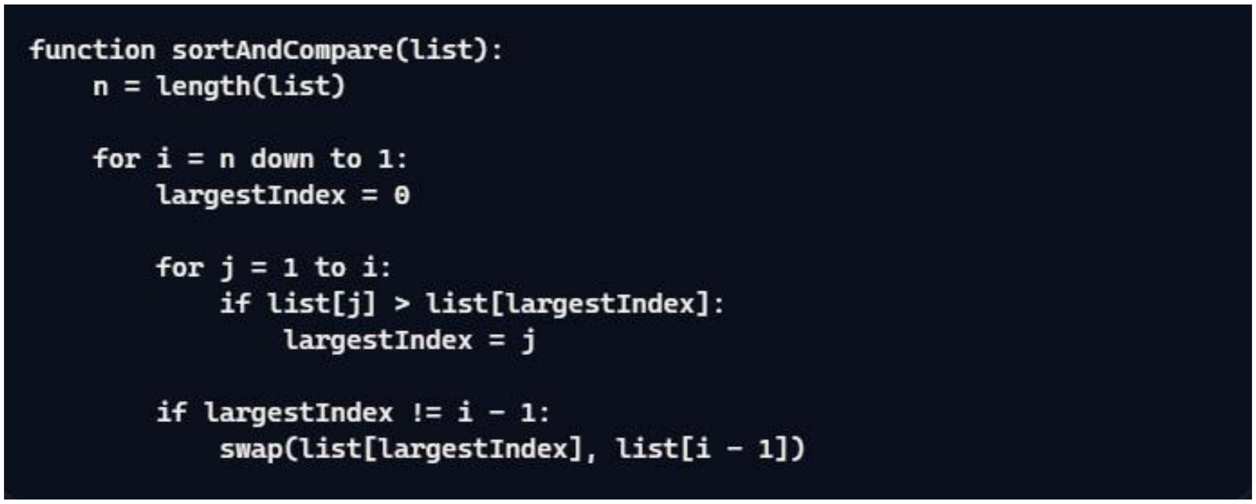
Step 2: Approach

Find the largest value in the list and compare it to the value at the end of the list. If the largest value is not the value at the end of the list, swap them places in the list. If the largest is in the right spot at the end of the list already, then do nothing and move on. The last index is now sorted so move to the 2nd to last index and use it as the end of the list and repeat the process until you get all the way through the first index in the list.

Step 3: Pseudocode

```
PROMPT for list of values
GET values

lastIndex ← length of values + 1

WHILE lastIndex > 0
      max ← values[0]
      FOR i ← 0 … lastIndex
            IF values[i] > max
                  max ← values[i]
                  maxIndex ← i

      IF max > values[lastIndex]
            values[maxIndex] ← values[lastIndex]
            values[lastIndex] ← max

      lastIndex ← lastIndex - 1
```

Step 4: Microsoft Copilot



```
function sortAndCompare(list):
    n = length(list)

    for i = n down to 1:
        largestIndex = 0

        for j = 1 to i:
            if list[j] > list[largestIndex]:
                largestIndex = j

        if largestIndex != i - 1:
            swap(list[largestIndex], list[i - 1])
```

Step 5: Compare and Contrast

Provide an analysis as to the pros and cons of the two solutions?

Pros of mine is its easier to understand to me at least when getting the max value in the list. But they are similar. Mine uses a while loop instead of the for loop like copilot but it works the same way so it’s neither a pro nor con in this context I think. A pro of the copilot version is they just say swap() instead of assigning it how I do. The way copilot does that is more simple to understand and it did it allows the programmer more adaptation for their language and many even have the swap function.

How can your solution be improved based on what Copilot provided?

I can just say swap() instead of assigning the values. I also made a mistake in getting the last index in the list, it should be length of the list - 1 instead of + 1. I can also just use the max index variable instead of keeping the max number, it’s unnecessary. The for loop is also a good way instead of my while loop so I’ll change that too

How can Copilot's solution be improved based on what you know?

It can use the arrow instead of an equals sign for assigning variables to not get it confused with any comparisons and also make the keywords all uppercase instead of lowercase.

Does the pseudocode in Step 3 and Step 4 match the algorithm you performed in Step1?

Yes it does, it will work as expected

Step 3: Update

```
PROMPT for list of values
GET values

i_pivot ← length of values - 1

FOR i_pivot … 1
    i_largest = 0

    FOR i_check ← 1 … i_pivot
        IF values[i_check] > values[i_largest]
            i_largest ← i_check

    IF values[i_largest] > values[i_pivot]
        swap(values[i_largest], values[i_pivot])
```

Step 7: Trace

line number	values	i_pivot	i_largest	i_check
2	26, 6, 90, 55	/	/	/
4	26, 6, 90, 55	3	/	/
6	26, 6, 90, 55	3	/	/
7	26, 6, 90, 55	3	0	/
9	26, 6, 90, 55	3	0	1
9	26, 6, 90, 55	3	0	2
11	26, 6, 90, 55	3	2	2
9	26, 6, 90, 55	3	2	3
14	26, 6, 55, 90	3	2	3
6	26, 6, 55, 90	2	2	3
7	26, 6, 55, 90	2	0	3
9	26, 6, 55, 90	2	0	1
9	26, 6, 55, 90	2	0	2
11	26, 6, 55, 90	2	2	2
6	26, 6, 55, 90	1	2	2
7	26, 6, 55, 90	1	0	2
9	26, 6, 55, 90	1	0	1
14	6, 24, 55, 90	1	0	1

Step 8: Efficiency

```
PROMPT for list of values                                 // O(1)
GET values                                                // O(1)

i_pivot ← length of values - 1                            // O(1)

FOR i_pivot … 1                                           // O(n)
    i_largest = 0                                         // O(1)

    FOR i_check ← 1 … i_pivot                             // O(n)
        IF values[i_check] > values[i_largest]            // O(1)
            i_largest ← i_check                           // O(1)

    IF values[i_largest] > values[i_pivot]                // O(1)
        swap(values[i_largest], values[i_pivot])          // O(1)
```

Total efficiency = O(n^2)
This is because theres a nested O(n) loop inside another O(n) loop

Time it took

Step 1 By Hand: 15 minutes
Step 2 Approach: 10 minutes
Step 3 Pseudocode: 30 minutes
Step 4 Copilot: 5 minutes
Step 5 Compare and Contrast: 25 minutes
Step 6 Update: 20 minutes
Step 7 Trace: 45 minutes
Step 8 Efficiency: 10 minutes