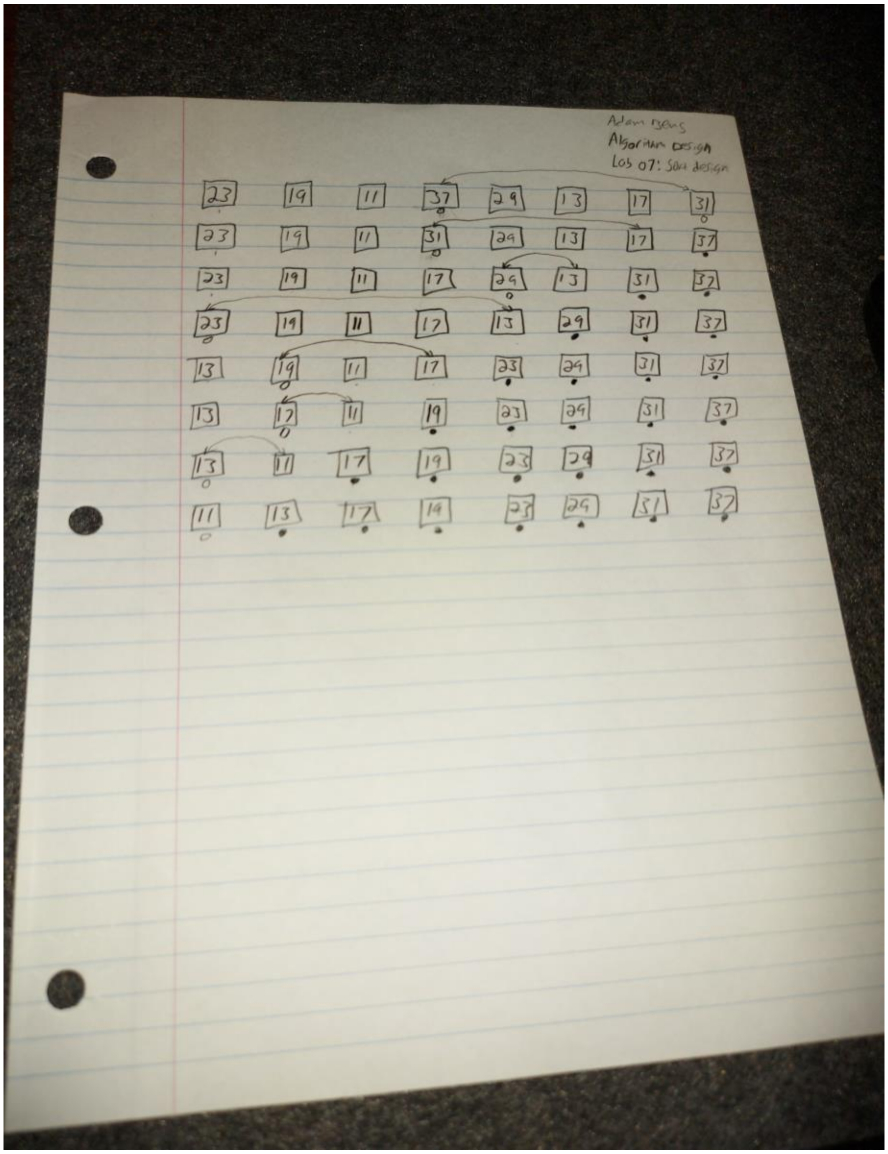
Adam Beus

Step 1: By Hand



Adam Beus
Algorithm design
Lab 07: Sort design

23	19	11	37	29	13	17	31
23	19	11	31	29	13	17	37
23	19	11	17	29	13	31	37
23	19	11	17	13	29	31	37
13	19	11	17	23	29	31	37
13	17	11	19	23	29	31	37
13	11	17	19	23	29	31	37
11	13	17	14	23	29	31	37

Step 2: Approach

I will take a list of 8 numbers, and go through them one by one, taking note of the largest one. Once I am through the list, I will swap the largest number with the final one, and leave that number at the end for the rest of this process. I will then repeat the process with the 7 remaining numbers, continuing to put the largest number I find at the end of the remaining numbers. This will continue until we get to the very end, and all of the numbers are in order.

Step 3: Pseudocode

```
numbers = [23, 19, 11, 37, 29, 13, 17, 31]
For i in range(len(numbers)):
     biggest_number = i
for j in range(i, i + 1):
     if j > i:
           biggest_number = j
function swap(i, j):
    temp = i
    i = j
    j = temp
```

Step 4: copilot

```
# Given a list of 8 numbers
numbers = [5, 2, 9, 1, 5, 6, 3, 8]

# Repeat the process for each position in the list
for i in range(len(numbers)):
    # Assume the current position has the largest number
    max_index = i

    # Compare with the remaining numbers
    for j in range(i + 1, len(numbers)):
        # If a larger number is found, update max_index
        if numbers[j] > numbers[max_index]:
            max_index = j

    # Swap the largest number with the last number in the remaining list
    numbers[i], numbers[max_index] = numbers[max_index], numbers[i]

# Print the sorted list
print("Sorted list:", numbers)
```

Step 5: Compare and Contrast

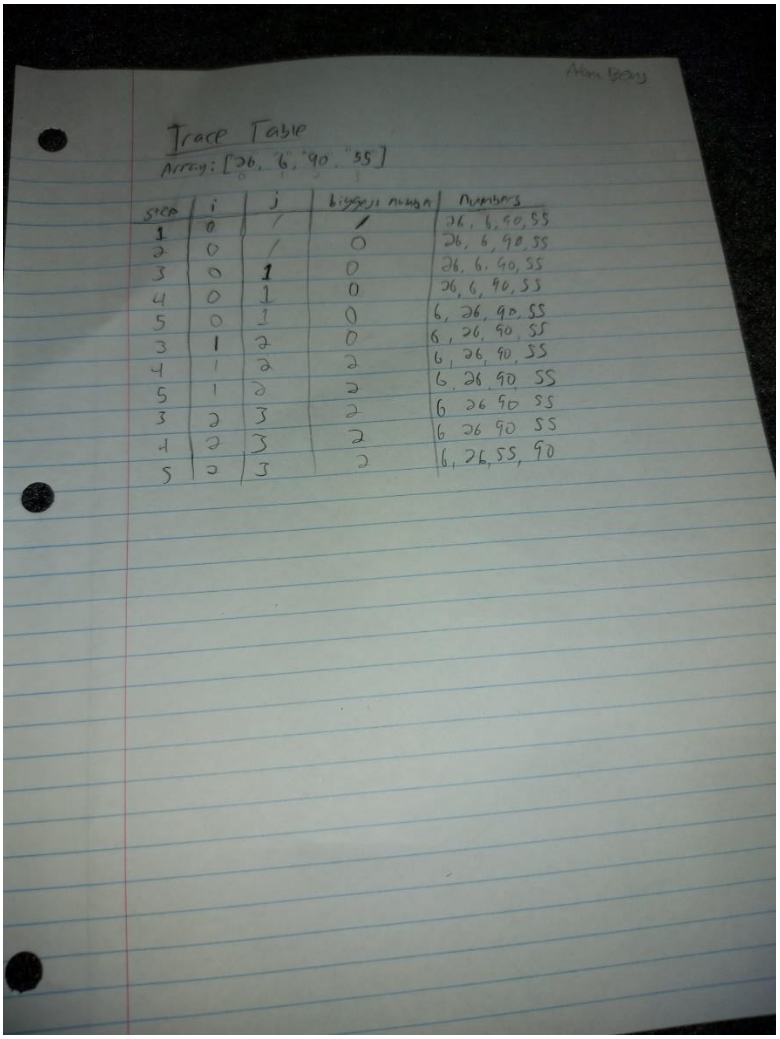
My solution is short and clean, and would work with minor changes. For example, the j for-loop needs to be (i + 1, len(numbers)), which copilot does correctly. This is the main error, and once this is fixed the code should work correctly. Copilots' solution is great and seems to maximize efficiency. The syntax for the swap that it uses is different, but would still work and looks more like actual code. The way I swapped them is with a simple swap function that I think would be easier if you are working with multiple languages. Both versions of the pseudocode match the algorithm I wrote in step 1.

Step 6: Updated Pseudocode

```
numbers = [23, 19, 11, 37, 29, 13, 17, 31]
For i in range(len(numbers))):                              //1
    biggest_number = i                                      //2
    for j in range(i + 1, len(numbers)):                    //3
```

```
            if numbers[j] > numbers[biggest_number]:
                biggest_number = j                        //4
        swap(numbers[i], numbers[biggest_number])         //5
        function swap(i, j):
            temp = i
            i = j
            j = temp
```

Step 7: Trace



Adam Beus

Trace Table

Array: [26, 6, 90, 55]

0 1 2 3

step	i	j	biggest number	numbers
1	0	/	/	26, 6, 90, 55
2	0	/	0	26, 6, 90, 55
3	0	1	0	26, 6, 90, 55
4	0	1	0	26, 6, 90, 55
5	0	1	0	6, 26, 90, 55
3	1	2	0	6, 26, 90, 55
4	1	2	2	6, 26, 90, 55
5	1	2	2	6, 26, 90, 55
3	2	3	2	6 26 90 55
4	2	3	2	6 26 90 55
5	2	3	2	6, 26, 55, 90

Adam Beus

Step 8: Efficiency

```
numbers = [23, 19, 11, 37, 29, 13, 17, 31]
For i in range(len(numbers))):                              O(n)
     biggest_number = i                                     O(1)
     for j in range(i + 1, len(numbers)):                   O(n)
           if numbers[j] > numbers[biggest_number]:         O(1)
                biggest_number = j                          O(1)
     swap(numbers[i], numbers[biggest_number])              O(1)
     function swap(i, j):                                   O(1)
         temp = i
         i = j
         j = temp
```

This is an O(n) efficiency algorithm.