Problem F: Inversion SwapSort

Thought process

It is always possible to swap the largest element with the last element to put the largest element at the last index, but swapping the largest element to the end first will not always work.

Consider the array [1, 8, 6, 1] with inversions (2, 3), (2, 4), and (3, 4). One correct answer is to swap (2, 3), then (2, 4), then (3, 4) to get [1, 1, 6, 8]. Another correct answer is to swap (3, 4), (2, 4), then (2, 3).

I then noticed that the sample cases and the example above had each swap decrease the number of inversions by one, and that each swap involves the element being swapped with the next largest element to its right.

However, this strategy doesn't work so well with [1, 6, 5, 6, 2, 3] which has 7 inversions:

- Swap (2, 3) to get [1, 5, 6, 6, 2, 3] – the next largest element of 6 at index 2 was 5 in index 3.
- Swap (2, 6) to get [1, 3, 6, 6, 2, 5] – the next largest element of 5 at index 2 was 3 in index 6.
- Swap (2, 5) to get [1, 2, 6, 6, 3, 5] – and so on…
- Swap (3, 6) to get [1, 2, 5, 6, 3, 6]
- Swap (3, 5) to get [1, 2, 3, 6, 5, 6] – we only have one inversion left?
- Swap (4, 6) to get [1, 2, 3, 6, 5, 6] – here is a swap of 2 items that don't form an inversion.
- Swap (4, 5) to get [1, 2, 3, 5, 6, 6] – and here we swap a 6 and a 5.

This means the correct answer doesn't necessarily involve swapping elements with the next largest at every step, and that this approach does not decrease the total number of inversions by 1 on each swap.

A new idea that I haven’t had time to implement yet: swap last element with next largest element until largest is at last index, then repeat for second to last element, and so on.