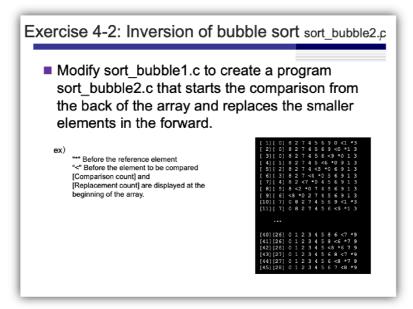# Exercise 4-2:

Maximilian Fernaldy - C2TB1702

To start the sort from the end of the array, we only need to replace the values used in the loops, and reuse almost all the code from exercise 4-1. We need to modify these lines in particular:

First, we modify the j-th level `for` loop to start from the last element on the list with index `NUM-1`. Then, we use `j > i` for the looping condition, which means it will first go through the last element to the second element, then from the last to the third, and so on until it reaches the last comparison, which compares the last element with the second element. Finally we *decrement* `j` after every iteration instead of incrementing it so that we move backwards in the array.

```
for (j = NUM-1; j > i; j--) {
    ...
}
```

Next, we need to modify the `for` loop that prints the array. The element that is being referenced is still the `j`-th element, so we don't need to change that part. However, the element it is being compared to is now the `j-1`-th element, as opposed to `j+1` from earlier. Therefore, we should change the condition of the `else if` operator to execute the block is `k` matches `j-1` instead. We also change the symbol shown beside the number to "<" instead of ">".

```
} else if (k == j - 1) {
    printf("<%d ", A[k]);
}
```

Finally, we of course need to change the code that swaps the numbers around. If the compared number is larger than the referenced one, they should be swapped around.

```
if (A[j-1] > A[j]) {
    temp = A[j];
    A[j] = A[j-1];
    A[j-1] = temp;

    swaps++;
}
```

Running the compiled program gives:

```
max ~ % cdpip && cd lec04/ex4-2 && ./sort_bubble2
Array before sorting:
8 2 7 4 5 6 9 0 1 3

Bubble sorting...
Loop 1:
[1][0] 8 2 7 4 5 6 9 0 <1 *3
[2][0] 8 2 7 4 5 6 9 <0 *1 3
[3][0] 8 2 7 4 5 6 <9 *0 1 3
[4][1] 8 2 7 4 5 <6 *0 9 1 3
[5][2] 8 2 7 4 <5 *0 6 9 1 3
[6][3] 8 2 7 <4 *0 5 6 9 1 3
[7][4] 8 2 <7 *0 4 5 6 9 1 3
[8][5] 8 <2 *0 7 4 5 6 9 1 3
[9][6] <8 *0 2 7 4 5 6 9 1 3

Loop 2:
[10][7] 0 8 2 7 4 5 6 9 <1 *3
[11][7] 0 8 2 7 4 5 6 <9 *1 3
[12][8] 0 8 2 7 4 5 <6 *1 9 3
[13][9] 0 8 2 7 4 <5 *1 6 9 3
[14][10] 0 8 2 7 <4 *1 5 6 9 3
[15][11] 0 8 2 <7 *1 4 5 6 9 3
[16][12] 0 8 <2 *1 7 4 5 6 9 3
[17][13] 0 <8 *1 2 7 4 5 6 9 3

Loop 3:
[18][14] 0 1 8 2 7 4 5 6 <9 *3
[19][15] 0 1 8 2 7 4 5 <6 *3 9
[20][16] 0 1 8 2 7 4 <5 *3 6 9
[21][17] 0 1 8 2 7 <4 *3 5 6 9
[22][18] 0 1 8 2 <7 *3 4 5 6 9
[23][19] 0 1 8 <2 *3 7 4 5 6 9
[24][19] 0 1 <8 *2 3 7 4 5 6 9

Loop 4:
[25][20] 0 1 2 8 3 7 4 5 <6 *9
[26][20] 0 1 2 8 3 7 4 <5 *6 9
[27][20] 0 1 2 8 3 7 <4 *5 6 9
[28][20] 0 1 2 8 3 <7 *4 5 6 9
[29][21] 0 1 2 8 <3 *4 7 5 6 9
[30][21] 0 1 2 <8 *3 4 7 5 6 9

Loop 5:
[31][22] 0 1 2 3 8 4 7 5 <6 *9
[32][22] 0 1 2 3 8 4 7 <5 *6 9
[33][22] 0 1 2 3 8 4 <7 *5 6 9
[34][23] 0 1 2 3 8 <4 *5 7 6 9
[35][23] 0 1 2 3 <8 *4 5 7 6 9

Loop 6:
[36][24] 0 1 2 3 4 8 5 7 <6 *9
[37][24] 0 1 2 3 4 8 5 <7 *6 9
[38][25] 0 1 2 3 4 8 <5 *6 7 9
[39][25] 0 1 2 3 4 <8 *5 6 7 9

Loop 7:
[40][26] 0 1 2 3 4 5 8 6 <7 *9
[41][26] 0 1 2 3 4 5 8 <6 *7 9
[42][26] 0 1 2 3 4 5 <8 *6 7 9

Loop 8:
[43][27] 0 1 2 3 4 5 6 8 <7 *9
[44][27] 0 1 2 3 4 5 6 <8 *7 9

Loop 9:
[45][28] 0 1 2 3 4 5 6 7 <8 *9

Sorted array:
0 1 2 3 4 5 6 7 8 9
45 comparisons, 28 replacements made
max ex4-2 %
```