

Exercise 4-1: sort_bubble.c

Maximilian Fernaldy - C2TB1702

Exercise 4-1: Visualization of bubble sort

5

- Modify sort_bubble.c and create a program sort_bubble1.c that visualizes bubble sort.
- Also, count and display the number of comparisons and the number of replacements.

- ex)
 - *** Before the reference element
 - ">" Before the element to be compared
 - [Comparison count] and [Replacement count] are displayed at the beginning of the array.

```
[ 1][ 0] *8 >2 7 4 5 6 9 0 1 3
[ 2][ 1] 2 *8 >7 4 5 6 9 0 1 3
[ 3][ 2] 2 7 *8 >4 5 6 9 0 1 3
[ 4][ 3] 2 7 4 *8 >5 6 9 0 1 3
[ 5][ 4] 2 7 4 5 *8 >6 9 0 1 3
[ 6][ 5] 2 7 4 5 6 *8 >9 0 1 3
[ 7][ 6] 2 7 4 5 6 9 *9 >0 1 3
[ 8][ 7] 2 7 4 5 6 9 0 *9 >1 3
[ 9][ 8] 2 7 4 5 6 9 0 1 *9 >3
```

....

```
[38][24] 2 0 *4 >1 3 5 6 7 8 9
[39][25] 2 0 1 *4 >3 5 6 7 8 9
[40][26] *2 >0 1 3 5 6 7 8 9
[41][27] 0 *2 >1 3 4 5 6 7 8 9
[42][28] 0 1 *2 >3 4 5 6 7 8 9
[43][28] *5 >1 2 3 4 5 6 7 8 9
[44][28] 0 *1 >2 3 4 5 6 7 8 9
[45][28] *0 >1 2 3 4 5 6 7 8 9
```

- The origin of "bubble" is from the appearance that the larger values move (emerge) to the edge in order.
- Bubble sort always makes $n(n-1)/2$ comparisons

To visualize the comparisons, we need to modify the `j` level `for` loop of the program to print the current condition of the array for each comparison ran. An asterisk should be printed whenever our `for` loop encounters the element which is currently being referenced. Since the reference element is always compared to the element right next to it, we can just use `j+1` for its index. Then, if the `for` loop encounters the `j+1`-th element, it should print a ">" before printing the value of the element.

```
for (j = 0; j < NUM-i-1; j++) {
    comparisons++;
    printf("[%d][%d] ", comparisons, swaps); // Display the number of comparisons and swaps done
    for (int k = 0; k < NUM; k++) {
        if (k == j) {
            printf(" *%d ", A[k]);
        } else if (k == j + 1) {
            printf(" >%d ", A[k]);
        } else {
            printf("%d ", A[k]);
        }

        // Insert new line if k reaches the end of the array
        if (k == NUM - 1) {
            printf("\n");
        }
    }

    // If compared number is smaller than reference number, swap them
    if (A[j] > A[j+1]) {
        temp = A[j];
        A[j] = A[j+1];
        A[j+1] = temp;

        swaps++;
    }
}
```

To print the array, we add a lower level `for` loop with iterator `k`, which goes from 0 to `NUM-1` to print the elements one by one, terminating before it reaches `NUM`. When `k` is equal to `j`, that means the program is trying to print the element on the array that is being referenced, so it should print an asterisk before the integer. When `k` is equal to `j+1`, the program is trying to print the element it is being compared to, so it should print an exclamation mark before the integer.

Here, it's not critical to use `else if` instead of `if`, because an element cannot be indexed by `j` and `j+1` at the same time. However, it is still good practice to use `else if` so that the program can skip over this condition if it has already executed the block above it (which is

```
if (k == j) )
```

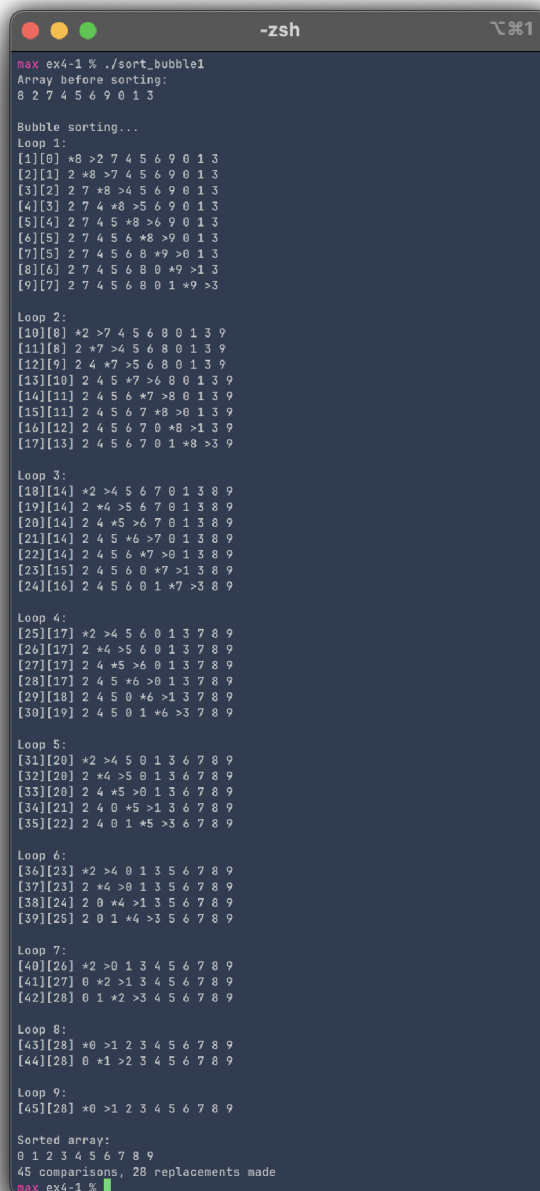
To count the number of comparisons and replacements made by the program, integers `comparisons` and `swaps` are initialized as 0, and every time a new comparison is made (the beginning of the `j` level `for` loop), `comparisons` is increased by 1:

```
for (j = 0; j < NUM-i-1; j++) {  
    comparisons++;  
    ...  
}
```

similarly, we modify the code that swaps the numbers to increment `swaps` by 1 to keep track of how many swaps has been made:

```
if (A[j] > A[j+1]) {  
    temp = A[j];  
    A[j] = A[j+1];  
    A[j+1] = temp;  
  
    swaps++;  
}
```

Compiling and running the program gives the following output:



```
max ex4-1 % ./sort_bubble1  
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][1] 2 *8 > 7 4 5 6 9 0 1 3  
[3][2] 2 7 *8 > 4 5 6 9 0 1 3  
[4][3] 2 7 4 *8 > 5 6 9 0 1 3  
[5][4] 2 7 4 5 *8 > 6 9 0 1 3  
[6][5] 2 7 4 5 6 *8 > 9 0 1 3  
[7][5] 2 7 4 5 6 8 *9 > 0 1 3  
[8][6] 2 7 4 5 6 8 0 *9 > 1 3  
[9][7] 2 7 4 5 6 8 0 1 *9 > 3  
  
Loop 2:  
[10][8] *2 > 7 4 5 6 8 0 1 3 9  
[11][8] 2 *7 > 4 5 6 8 0 1 3 9  
[12][9] 2 4 *7 > 5 6 8 0 1 3 9  
[13][10] 2 4 5 *7 > 6 8 0 1 3 9  
[14][11] 2 4 5 6 *7 > 8 0 1 3 9  
[15][11] 2 4 5 6 7 *8 > 0 1 3 9  
[16][12] 2 4 5 6 7 0 *8 > 1 3 9  
[17][13] 2 4 5 6 7 0 1 *8 > 3 9  
  
Loop 3:  
[18][14] *2 > 4 5 6 7 0 1 3 8 9  
[19][14] 2 *4 > 5 6 7 0 1 3 8 9  
[20][14] 2 4 *5 > 6 7 0 1 3 8 9  
[21][14] 2 4 5 *6 > 7 0 1 3 8 9  
[22][14] 2 4 5 6 *7 > 0 1 3 8 9  
[23][15] 2 4 5 6 0 *7 > 1 3 8 9  
[24][16] 2 4 5 6 0 1 *7 > 3 8 9  
  
Loop 4:  
[25][17] *2 > 4 5 6 0 1 3 7 8 9  
[26][17] 2 *4 > 5 6 0 1 3 7 8 9  
[27][17] 2 4 *5 > 6 0 1 3 7 8 9  
[28][17] 2 4 5 *6 > 0 1 3 7 8 9  
[29][18] 2 4 5 0 *6 > 1 3 7 8 9  
[30][19] 2 4 5 0 1 *6 > 3 7 8 9  
  
Loop 5:  
[31][20] *2 > 4 5 0 1 3 6 7 8 9  
[32][20] 2 *4 > 5 0 1 3 6 7 8 9  
[33][20] 2 4 *5 > 0 1 3 6 7 8 9  
[34][21] 2 4 0 *5 > 1 3 6 7 8 9  
[35][22] 2 4 0 1 *5 > 3 6 7 8 9  
  
Loop 6:  
[36][23] *2 > 4 0 1 3 5 6 7 8 9  
[37][23] 2 *4 > 0 1 3 5 6 7 8 9  
[38][24] 2 0 *4 > 1 3 5 6 7 8 9  
[39][25] 2 0 1 *4 > 3 5 6 7 8 9  
  
Loop 7:  
[40][26] *2 > 0 1 3 4 5 6 7 8 9  
[41][27] 0 *2 > 1 3 4 5 6 7 8 9  
[42][28] 0 1 *2 > 3 4 5 6 7 8 9  
  
Loop 8:  
[43][28] *0 > 1 2 3 4 5 6 7 8 9  
[44][28] 0 *1 > 2 3 4 5 6 7 8 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 ex4-1 %
```