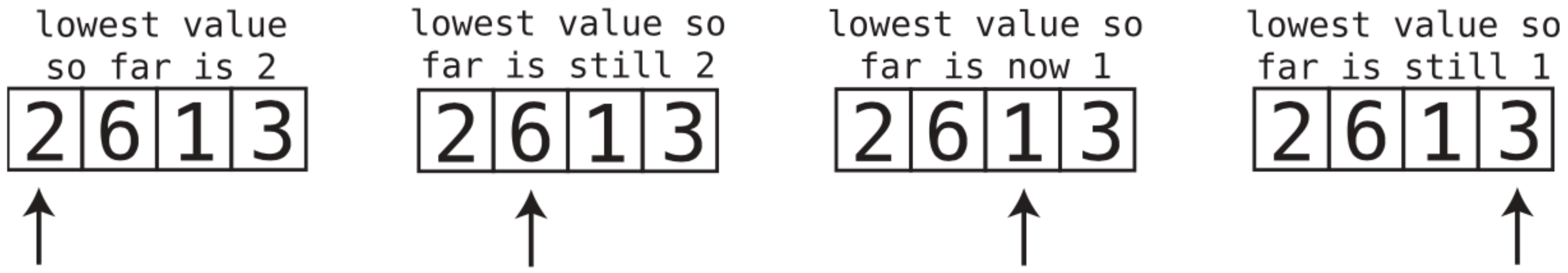
Selection Sort

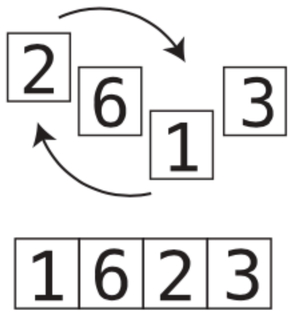
In the previous chapter, we explored a sorting algorithm known as Bubble Sort, which had an efficiency of O(N^2). We're now going to dig into another sorting algorithm called *Selection Sort*, and see how it measures up to Bubble Sort.

The steps of Selection Sort are as follows:

1. We check each cell of the array from left to right to determine which value is least. As we move from cell to cell, we keep track of the lowest value we've encountered so far. (We'll do this by storing its index in a variable.) If we encounter a cell that contains a value that is even lower than the one in our variable, we replace it so that the variable now points to the new index. See the following diagram:


lowest value so far is 2
2 6 1 3
lowest value so far is still 2
2 6 1 3
lowest value so far is now 1
2 6 1 3
lowest value so far is still 1
2 6 1 3

2. Once we've determined which index contains the lowest value, we swap its value with the value we began the pass-through with. This would be index 0 in the first pass-through, index 1 in the second pass-through, and so on. The diagram here illustrates making the swap of the first pass-through.



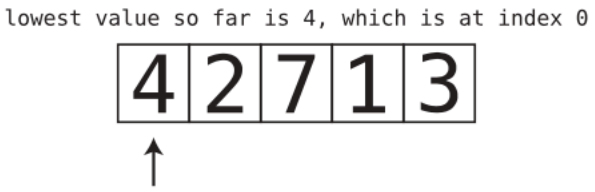
3. Each pass-through consists of Steps 1 and 2. We repeat the pass-throughs until we reach a pass-through that would start at the end of the array. By this point, the array will have been fully sorted.

Selection Sort in Action

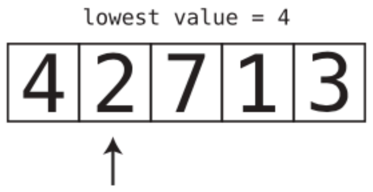
Let's walk through the steps of Selection Sort using the example array, `[4, 2, 7, 1, 3]`.

We begin our first pass-through:

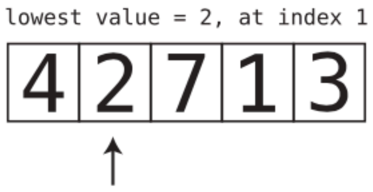
We set things up by inspecting the value at index 0. By definition, it's the lowest value in the array we've encountered so far (as it's the *only* value we've encountered so far), so we keep track of its index in a variable:



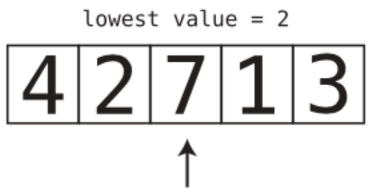
Step 1: We compare the 2 with the lowest value so far (which happens to be 4):



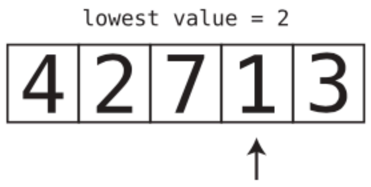
The 2 is even less than the 4, so it becomes the lowest value so far:



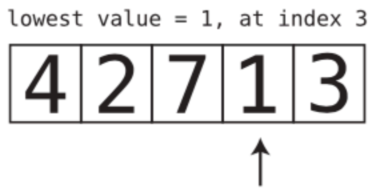
Step 2: We compare the next value—the 7—with the lowest value so far. The 7 is greater than the 2, so 2 remains our lowest value:



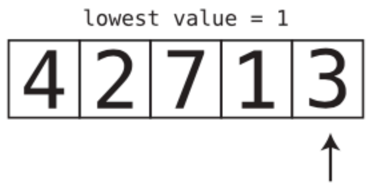
Step 3: We compare the 1 with the lowest value so far:



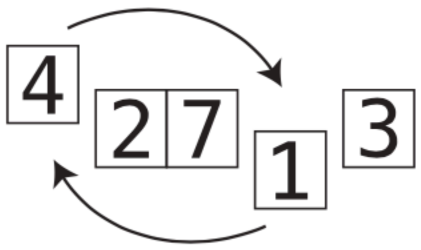
Because the 1 is even less than the 2, 1 becomes our new lowest value:



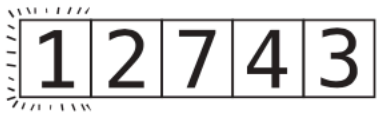
Step 4: We compare the 3 to the lowest value so far, which is the 1. We've reached the end of the array, and we've determined that 1 is the lowest value out of the entire array:



Step 5: Because 1 is the lowest value, we swap it with whatever value is at index 0—the index we began this pass-through with:

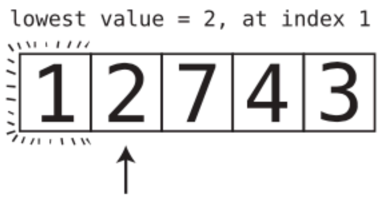


Since we've moved the lowest value to the beginning of the array, that means the lowest value is now in its correct spot:

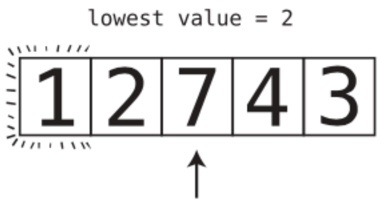


We are now ready to begin our second pass-through.

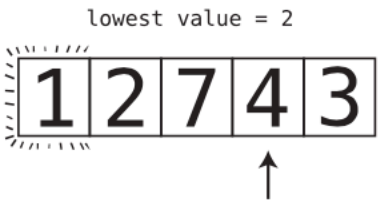
Setup: The first cell—index 0—is already sorted, so this pass-through begins at the next cell, which is index 1. The value at index 1 is the number 2, and it is the lowest value we've encountered in this pass-through so far:



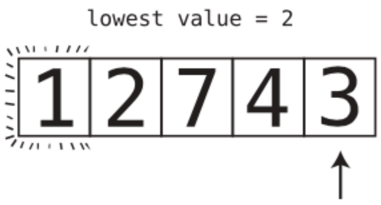
Step 6: We compare the 7 with the lowest value so far. The 2 is less than the 7, so 2 remains our lowest value:



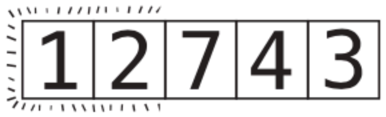
Step 7: We compare the 4 with the lowest value so far. The 2 is less than the 4, so 2 remains our lowest value:



Step 8: We compare the 3 with the lowest value so far. The 2 is less than the 3, so 2 remains our lowest value:

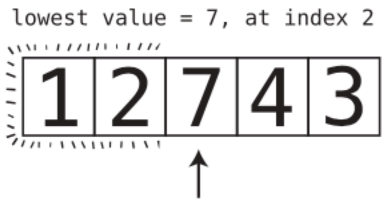


We've reached the end of the array. Since the lowest value from this pass-through was already in its correct spot, we don't need to perform a swap. This ends our second pass-through, leaving us with:

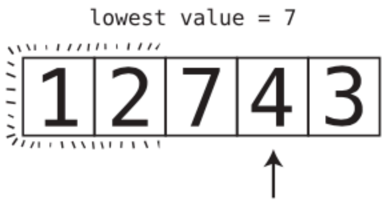


We now begin the third pass-through.

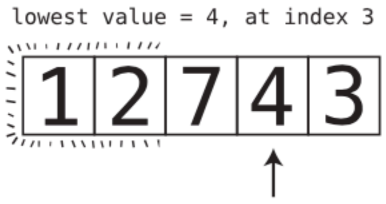
Setup: We begin at index 2, which contains the value 7. The 7 is the lowest value we've encountered so far in this pass-through:



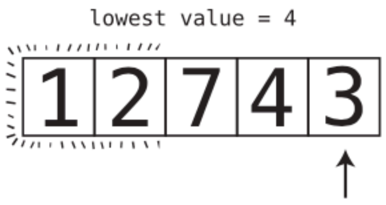
Step 9: We compare the 4 with the 7:



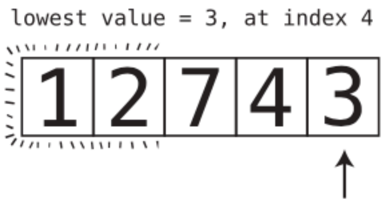
We note that 4 is our new lowest value:



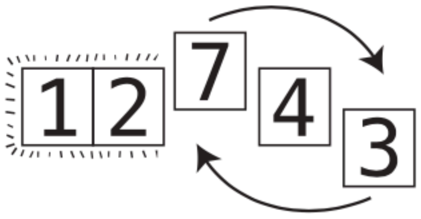
Step 10: We encounter the 3, which is even lower than the 4:



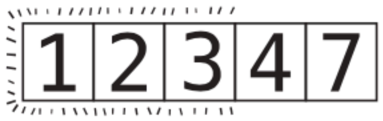
The 3 becomes our new lowest value:



Step 11: We've reached the end of the array, so we swap the 3 with the value we started our pass-through with, which is the 7:

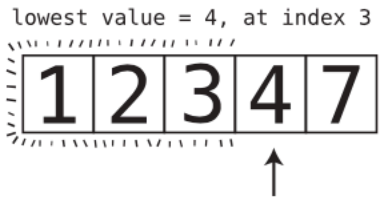


We now know that the 3 is in the correct place within the array:

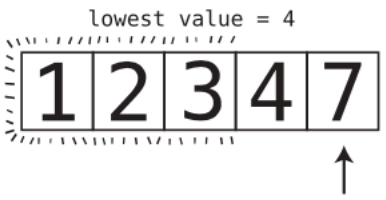


While you and I can both see that the entire array is correctly sorted at this point, the *computer* does not know this yet, so it must begin a fourth pass-through.

Setup: We begin the pass-through with index 3. The 4 is the lowest value so far:



Step 12: We compare the 7 with the 4:



The 4 remains the lowest value we've encountered in this pass-through so far, so we don't need to swap it, since it's already in the correct place.

Because all the cells besides the last one are correctly sorted, that must mean the last cell is also in the correct order, and our entire array is properly sorted: