**Introduction to Sorting Algorithms**

As programmers, we should choose to organize the data in a way that is useful. A common way to organize data is to **sort** it. For example, words from a dictionary are sorted in alphabetical order. Sorted data is valuable because it can make "looking up" entries efficient. Can you imagine searching for entries in a dictionary that isn't sorted alphabetically?!?

In this section we'll learn an algorithm to sort numbers. However, these algorithms can also be applied to sort other data like strings. An [algorithm](https://en.wikipedia.org/wiki/Algorithm) is a sequence of actions to take. We'll be using sorting algorithms as sequences of steps we can follow to organize numbers in increasing (or decreasing) order. Our goal in this section will be to take an array of numbers and modify that array so that its elements are in increasing order.

**Bubble Sort Algorithm**

The Bubble Sort algorithm gets its name because it behaves as if the large numbers float to the top of the array like bubbles. See the wikipedia page on [Bubble Sort](https://en.wikipedia.org/wiki/Bubble_sort#Analysis) and check out the visualization of how it behaves!

**How does a pass of Bubble Sort work?**

Before we explore Bubble Sort in nitty-gritty Ruby detail, let's visualize it conceptually! Bubble sort works by performing multiple *passes* to move elements closer to their final positions. A single pass will iterate through the entire array once.

A pass works by scanning the array from left to right, two adjacent elements at a time, and checking if they are ordered correctly. To be ordered correctly the first element must be less than or equal to the second. If the two elements are not ordered properly, then we swap them to correct their relative order. Afterwards, we scan the next two numbers and continue to repeat this process until we have gone through the entire array.

Let's see one pass of bubble sort on the array [2, 8, 5, 2, 6]. On each step, the elements currently being scanned are in **bold**.

* [**2**, **8**, 5, 2, 6] - ordered, so leave them alone
* [2, **8**, **5**, 2, 6] - not ordered, so swap
* [2, 5, **8**, **2**, 6] - not ordered, so swap
* [2, 5, 2, **8**, **6**] - not ordered, so swap
* [2, 5, 2, 6, 8] - our first pass is complete

By the end of this pass we have correctly placed 8 by "bubbling it up" to the end of the array. From this point we would continue doing passes on the array until everything is in sorted order.

**How do we know when we are done Bubble Sorting?**

During Bubble Sort, we can tell if the array is in sorted order by checking if we made a swap during the previous pass performed. If a swap was not performed during the previous pass, then the array must be fully sorted and we can stop the algorithm.

You're probably confused by the last statement. Recall that a pass of Bubble Sort checks if any adjacent elements are **out of order** and swaps them if they are. If we don't make any swaps during a pass, then everything must be **in order**, so our job is done. Let that sink in for a bit.

**Swapping Elements**

Like we saw in the previous example, Bubble Sort manipulates the array by swapping the position of two elements. To implement Bubble Sort in Ruby, we'll need to perform this operation, so let's explore it in isolation. A key detail is that we may need an extra temporary variable to store one of the elements since we will be overwriting them in the array:

array = ["a", "b", "c", "d"] # let's swap "a" and "b"

temp = array[0]; # save a copy of the first ele

array[0] = array[1]; # overwrite the first ele with the second ele

array[1] = temp; # overwrite the second ele with the first ele copy

p array # => ["b", "a", "c", "d"]

It worked! But hmmm, using that temp variable was kind of messy. Is there an easier way to swap elements of an array? Yep! Ruby has many clean shortcuts:

array = ["a", "b", "c", "d"] # let's swap "a" and "b"

array[0], array[1] = array[1], array[0]

p array # => ["b", "a", "c", "d"]

As a side note: you can use this swapping mechanic for entire variables as well

food = "mom's spaghetti"

clothing = "sweater"

food, clothing = clothing, food

p food # => "sweater"

p clothing # => "mom's spaghetti"

**Bubble Sort Ruby Implementation**

Using swap and our newfound understanding of Bubble Sort, let's code! Take a look at the snippet below and try to understand how it corresponds to our conceptual understanding of the algorithm. Scroll down to the commented version when you get stuck.

def bubble\_sort(array)

sorted = false

while !sorted

sorted = true

(0...array.length - 1).each do |i|

if array[i] > array[i + 1]

array[i], array[i + 1] = array[i + 1], array[i]

sorted = false

end

end

end

array

end

def bubble\_sort(array)

sorted = false # when this var is false, that means the array is not fully sorted yet

while !sorted # while the array is not sorted...

sorted = true # reset the sorted flag to true

# the each below will perform a single 'pass' of bubble sort

(0...array.length - 1).each do |i|

if array[i] > array[i + 1] # if adjacent elements are out of order...

array[i], array[i + 1] = array[i + 1], array[i] # then swap their positions

sorted = false # since we just made a swap, we may need to perform

end # an additional 'pass', so set the flag to false

end

end

array

end