Bubble Sort

Pseudocode:

= 1 to n-1

Repeat n-1 times

For i from 0 to n-2

If numbers[i] and numbers[i+1] out of order

Swap them

So that means that you're doing $n-1$ things $n-1$ times.

I literally multiply how many times the outer loop is running by how many times the inner loop is running, which gives me sort of FOIL method $n-1$ squared, and I could multiply that whole thing out.

$n-1$ in the outer
$n-1$ on the inner

Let's go ahead and FOIL this:

$$(n-1) \times (n-1)$$

$$n^2 - 1n - 1n + 1$$

$$n^2 - 2n + 1$$

Which of these terms is clearly going to be dominant?

the n^2. Even though $-2n$ is a good thing, because it's subtracting off some of the time required, $+1$ is not a big thing, there's ~~no~~ such "drops in the bucket when n gets really large, like in the millions or billions certainly, that BUBBLE SORT 2 is on the order of n squared (n^2)

↓

Isn't the same exactly as selection sort?

But as n gets big, honestly, we're barely going to be able to notice the difference most likely.

If we consider now the lower bound of bubble sort's running time, here's where things get potentially interesting. What might you claim is the running time

of bubble sort in the best case? (when the numbers are already sorted).

But what if I add a bit of an optimization?
Where if I compare 2 people and I don't swap them, and I go all the way through the list comparing every pair of adjacent people, and I make no swaps, it would be kind of not just naive but stupid to do the same process again because if the humans haven't moved, I'm not going to make any ≠ decisions. I'm going to do nothing again.

↳ It would be stupid, very inefficient, to go back and forth.

So If I modify our pseudocode with just an additional IF condition, I bet we can speed this up. *

Inside of that same pseudocode, what if I say, hey, if no swaps, quit?

Like quit, prematurely before the loops are finished running. One of the loops has gone through per the indentation here. But if I do an loop from left to right and I have made no swaps, which you can think of as just being one other variable that's ++ as (i) go keeping, track of how many swaps, if I've made no swaps from left to right, I'm not going to make any swaps the next time around either.

So let's just QUIT at that point.

→ that is to say in the best case, if you will, when the list is already sorted, the Ω notation for bubble sort might indeed be omega of n ($\Omega(n)$) if you add that optimization, so as to short circuit all of that inefficient looping to do it only as many times as is necessary.

→ If the running time of selection sort and bubble sort are both in big O of n^2, but selection sort is in $\Omega(n^2)$ while bubble sort is in $\Omega(n)$, which sounds better — I think if I may, should we just always use bubble sort?

Yes, if we think that we might benefit our time from a lot of good case scenarios or best case scenarios.

↓

However, the good at hand in just a bit is going to be to do even better than both of these.

Visualization tool → (bunch of numbers, or array of numbers)

Show these things actually work and look at a faster rate than our humans are able to do here on stage.

short bars = small numbers
tall bars = big numbers