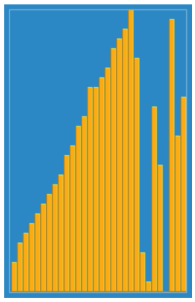
Insertion Sort



Let's talk about a sort that is one that you will use occasionally in certain contexts, insertion sort.

With insertion sort, you treat the first part of your list as sorted and the second part of your list as unsorted. Our algorithm will start by saying everything the 1 index (so just index 0, the first element) is sorted and everything after unsorted. By definition a list of one is already sorted. From there, we start with the next element in the list (in this case, the 1 index, the second element) and loop backwards over our sorted list, asking "is the element that I'm looking to insert larger than what's here? If not, you work your way to the back of the array. If you land at the first element of the sorted part of the list, what you have is smaller than everything else and you put it at the start. You then repeat this until you've done it over the whole list!

The mechanism by which we'll do this is that we'll keep moving bigger elements to the right by swapping items in the array as we move across the element. When we come to the place where we're going to insert, we'll stop doing those swaps/

Let's do an example.

```
[3, 2, 5, 4, 1]

   ↓
[3, 2, 5, 4, 1] // the ↓ is the number we're looking to insert, everything before

Is 2 larger than 3? No. Move 3 to the right.
Beginning of list, insert 2 at index 0.

      ↓
[2, 3, 5, 4, 1]

Is 5 larger than 3? Yes.
Insert after 3 (it's already there so it doesn't move)

         ↓
[2, 3, 5, 4, 1]

Is 4 larger than 5? No. Move 5 to the right.
Is 4 larger than 3? Yes.
Insert after 3 at index 2.

            ↓
[2, 3, 4, 5, 1]

Is 1 larger than 5? No. Move 5 to the right.
Is 1 larger than 4? No. Move 4 to the right.
Is 1 larger than 3? No. Move 3 to the right.
Is 1 larger than 2? No. Move 2 to the right.
Beginning of list, insert 1 at index 0

[1, 2, 3, 4, 5]

Reached end of list, list is sorted.
```

Big O

What's the worst case scenario for this sort? A reverse sorted list. You'd have to make every comparison possible since the numbers would be moving from the end of the list to the beginning of the list every single iteration and have to move an item every single iteration. This would be $O(n^2)$

What's the best case scenario? An already-sorted list. In this case, it would only make ask the question "is x the larger than y? Yes." It would never do any extra comparisons and it would never need to make any swaps. This would be O(n). This is when you would consider using insertion sort over something like quick sort or merge sort: if the arrays you're going to be sorting are likely to be sorted already or very close to it (a few swaps is cheap.) This will make it faster than quick sort and merge sort.

What's the average case of a randomly shuffled array? Well, it'll make a lot of comparisons and swaps, and those just grow exponentially as the array grows, so it'll still be $O(n^2)$. Not great for average-case use, only nearly-sorted situations.

What about spatial complexity? O(1)! We don't create any additional items for this sort.

The sort *is* destructive since we work on the array itself and the sort can be stable as long as you program it so that they stay in order during the insertions.