Elementary Sorting Algorithms

Objectives

- Implement bubble sort
- Implement selection sort
- Implement insertion sort
- Understand why it is important to learn these simpler sorting algorithms

What is sorting?

Sorting is the process of rearranging the items in a collection (e.g. an array) so that the items are in some kind of order.

Examples

- Sorting numbers from smallest to largest
- Sorting names alphabetically
- Sorting movies based on release year
- Sorting movies based on revenue

Why do we need to learn this?

- Sorting is an incredibly common task, so it's good to know how it works
- There are many different ways to sort things, and different techniques have their own advantages and disadvantages
- Sorting sometimes has quirks, so it's good to understand how to navigate them

JavaScript has a sort method...

Yes, it does!

...but it doesn't always work the way you expect.

☺

```
[ "Steele", "Colt", "Data Structures", "Algorithms" ].sort();
// [ "Algorithms", "Colt", "Data Structures", "Steele" ]
```

☹

```
[ 6, 4, 15, 10 ].sort();
// [ 10, 15, 4, 6 ]
```

Telling JavaScript how to sort

- The built-in sort method accepts an optional *comparator* function
- You can use this comparator function to tell JavaScript how you want it to sort
- The comparator looks at pairs of elements (*a* and *b*), determines their sort order based on the return value
    - If it returns a negative number, *a* should come before *b*
    - If it returns a positive number, *a* should come after *b*,
    - If it returns 0, *a* and *b* are the same as far as the sort is concerned

Telling JavaScript how to sort

Examples

```
function numberCompare(num1, num2) {
  return num1 - num2;
}

[ 6, 4, 15, 10 ].sort(numberCompare);
// [ 4, 6, 10, 15 ]
```

```
function compareByLen(str1, str2) {
  return str1.length - str2.length;
}

[ "Steele", "Colt", "Data Structures", "Algorithms" ]
  .sort(compareByLen);
// [ "Colt", "Steele", "Algorithms", "Data Structures" ]
```

Before we sort, we must swap!

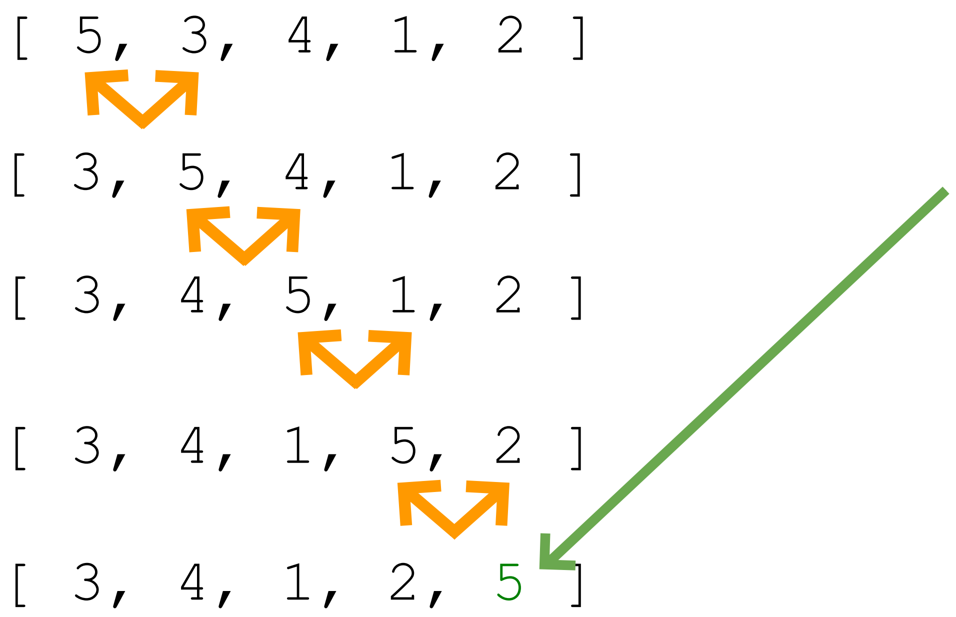
Many sorting algorithms involve some type of swapping functionality (e.g. swapping to numbers to put them in order)

```
// ES5
function swap(arr, idx1, idx2) {
  var temp = arr[idx1];
  arr[idx1] = arr[idx2];
  arr[idx2] = temp;
}

// ES2015
const swap = (arr, idx1, idx2) => {
  [arr[idx1],arr[idx2]] = [arr[idx2],arr[idx1]];
}
```

BubbleSort

A sorting algorithm where the largest values bubble up to the top!



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

[ 3, 5, 4, 1, 2 ]

[ 3, 4, 5, 1, 2 ]

[ 3, 4, 1, 5, 2 ]

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

5 is now in its sorted position!

BubbleSort Pseudocode

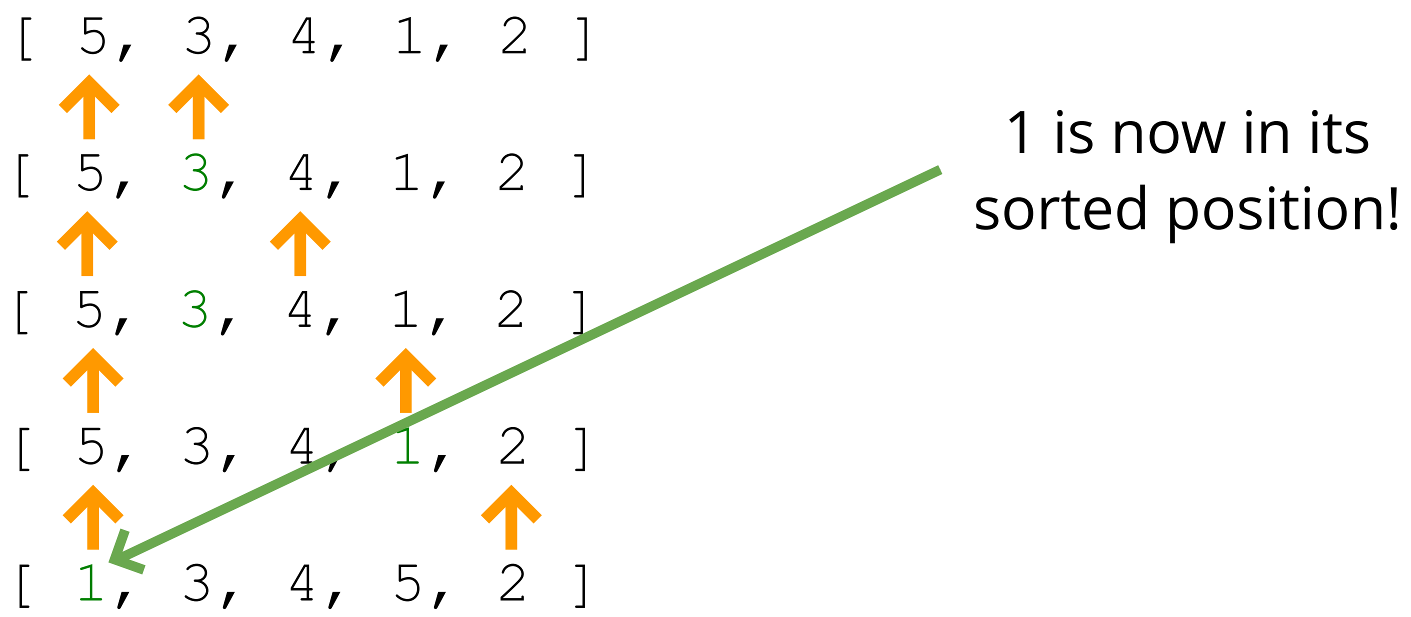
- Start looping from with a variable called i the end of the array towards the beginning
- Start an inner loop with a variable called j from the beginning until i - 1
- If arr[j] is greater than arr[j+1], swap those two values!
- Return the sorted array

Let's visualize this!

YOUR TURN

Selection Sort

Similar to bubble sort, but instead of first placing large values into sorted position, it places small values into sorted position



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

Selection Sort Pseudocode

- Store the first element as the smallest value you've seen so far.
- Compare this item to the next item in the array until you find a smaller number.
- If a smaller number is found, designate that smaller number to be the new "minimum" and continue until the end of the array.
- If the "minimum" is not the value (index) you initially began with, swap the two values.
- Repeat this with the next element until the array is sorted.

Let's visualize this!

YOUR
TURN

Insertion Sort

Builds up the sort by gradually creating a larger left half which is always sorted

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

[ 3, 5, 4, 1, 2 ]

[ 3, 4, 5, 1, 2 ]

[ 1, 3, 4, 5, 2 ]

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

Insertion Sort Pseudocode

- Start by picking the second element in the array
- Now compare the second element with the one before it and swap if necessary.
- Continue to the next element and if it is in the incorrect order, iterate through the sorted portion (i.e. the left side) to place the element in the correct place.
- Repeat until the array is sorted.

Let's visualize this!

YOUR

TURN

Big O of Sorting Algorithms

Algorithm	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)

Recap

- Sorting is *fun*damental!
- Bubble sort, selection sort, and insertion sort are all roughly equivalent
- All have average time complexities that are quadratic
- We can do better…but we need more complex algorithms!