Your Next Week

Tuesday June 9	Wednesday June 10	Thursday June 11	Friday June 12
6:30 PM **— DUE Class 21 Lab** **— DUE Class 21 Code Challenge** **— DUE Class 22 Reading** — Class 22A	*6:30 PM* — Class 22B *MIDNIGHT* **— DUE Class 22 Learning Journal**	*6:30 PM* — Co-working	
Saturday June 13	**Sunday June 14**	**Monday June 15**	**Tuesday June 16**
9:00 AM **— DUE Class 22 Lab** **— DUE Class 22 Code Challenge** **— DUE Class 23 Reading** — Class 23 — Interview Prep *MIDNIGHT* **— DUE Class 23 Learning Journal**	*MIDNIGHT* **— DUE Career: Coffee Networking Report** **— DUE Class 22-23 Feedback**		*6:30 PM* **— DUE Class 23 Lab** **— DUE Class 23 Code Challenge** **— DUE Class 24 Reading** — Class 24A

What We've Covered

Module 01
Javascript Fundamentals and Data Models

C01 — Node Ecosystem, TDD, CI/CD
C02 — Classes, Inheritance, Functional Programming
C03 — Data Modeling & NoSQL Databases
C04 — Advanced Mongo/Mongoose
C05 — DSA: Linked Lists

Module 02
API Servers

C06 — HTTP and REST
C07 — Express
C08 — Express Routing & Connected API
C09 — API Server
C11 — DSA: Stacks and Queues

Module 03
Auth/Auth

C10 — Authentication
C12 — OAuth
C13 — Bearer Authorization
C14 — Access Control (ACL)
C15 — DSA: Trees

Module 04
Realtime

C16 — Event Driven Applications
C17 — TCP Server
C18 — Socket.io
C19 — Message Queues
C20 — Midterms Prep

Midterms

Module 05
React Basics

C21 — Component Based UI
C22 — React Testing and Deployment
C23 — Props and State
C24 — Routing and Component Composition
C25 — DSA: Sorting and HashTables

Module 06
Advanced React

C26 — Hooks API
C27 — Custom Hooks
C28 — Context API
C29 — Application State with Redux
C30 — DSA: Graphs

Module 07
Redux State Management

C31 — Combined Reducers
C32 — Asynchronous Actions
C33 — Additional Topics
C34 — React Native
C35 — DSA: Review

Module 08
UI Frameworks

C36 — Gatsby and Next
C37 — JavaScript Frameworks
C38 — Finals Prep

Finals

Code Challenge 21 Review

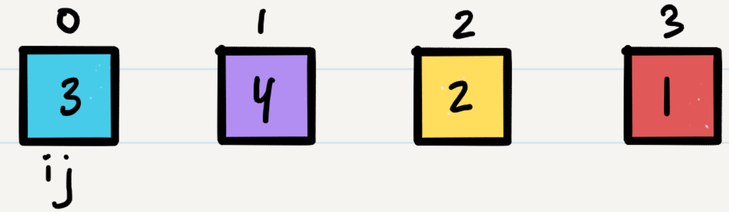
Sorting

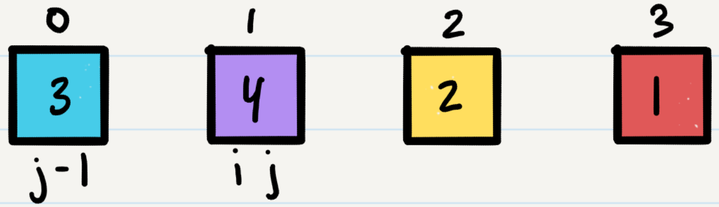
- A very common coding need
- What happens when the size of the list (n) gets larger and larger?
- Various algorithms have different time complexities
- We'll cover the simplest one this class

Insertion Sort

- Imagine sorting playing cards in your hands
    - Work left to right, index 0 to n-1
    - For each index, move the value left until it doesn’t need to go further

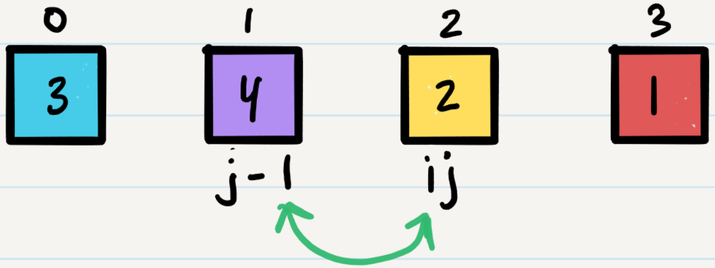
```
let i = 0;

while(i < arr.length) {
    let j = i;

    while(j > 0 && arr[j - 1] > arr[j]) {
        let temp = arr[j - 1];
        arr[j - 1] = arr[j];
        arr[j] = temp;
        j--;
    }

    i++;
}
```





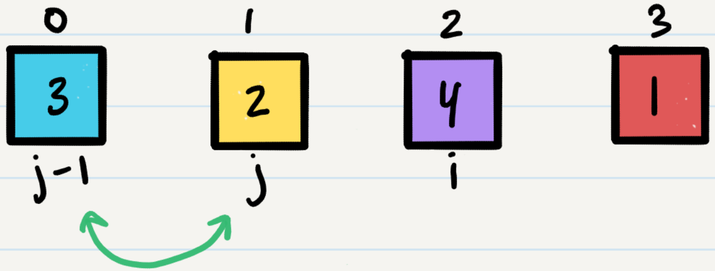
arr[j-1] < arr[j]

no swap



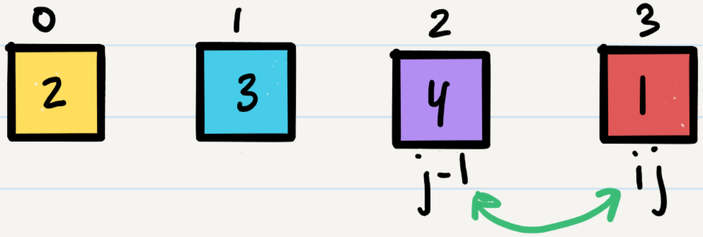
arr[j-1] > arr[j]

swap!



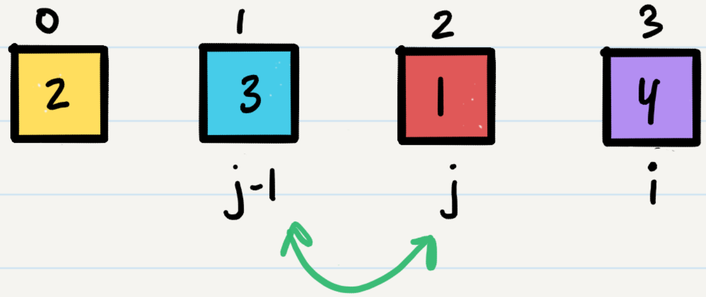
arr[j-1] > arr[j]

swap!

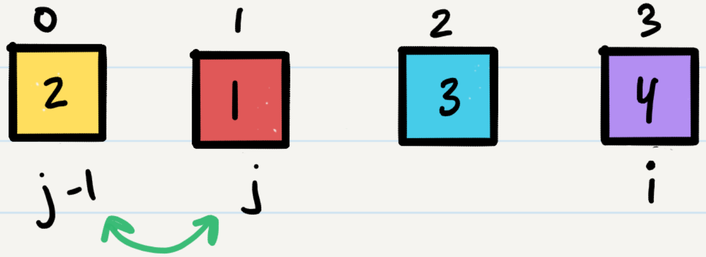


arr[j-1] > arr[j]

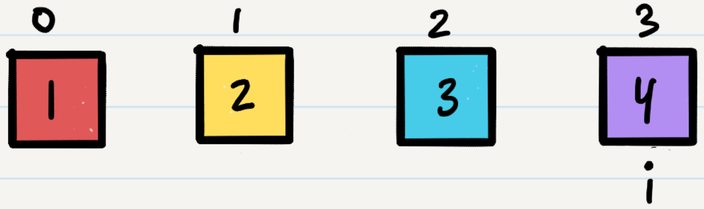
swap!



3 1
arr[j-1] > arr[j]
swap!



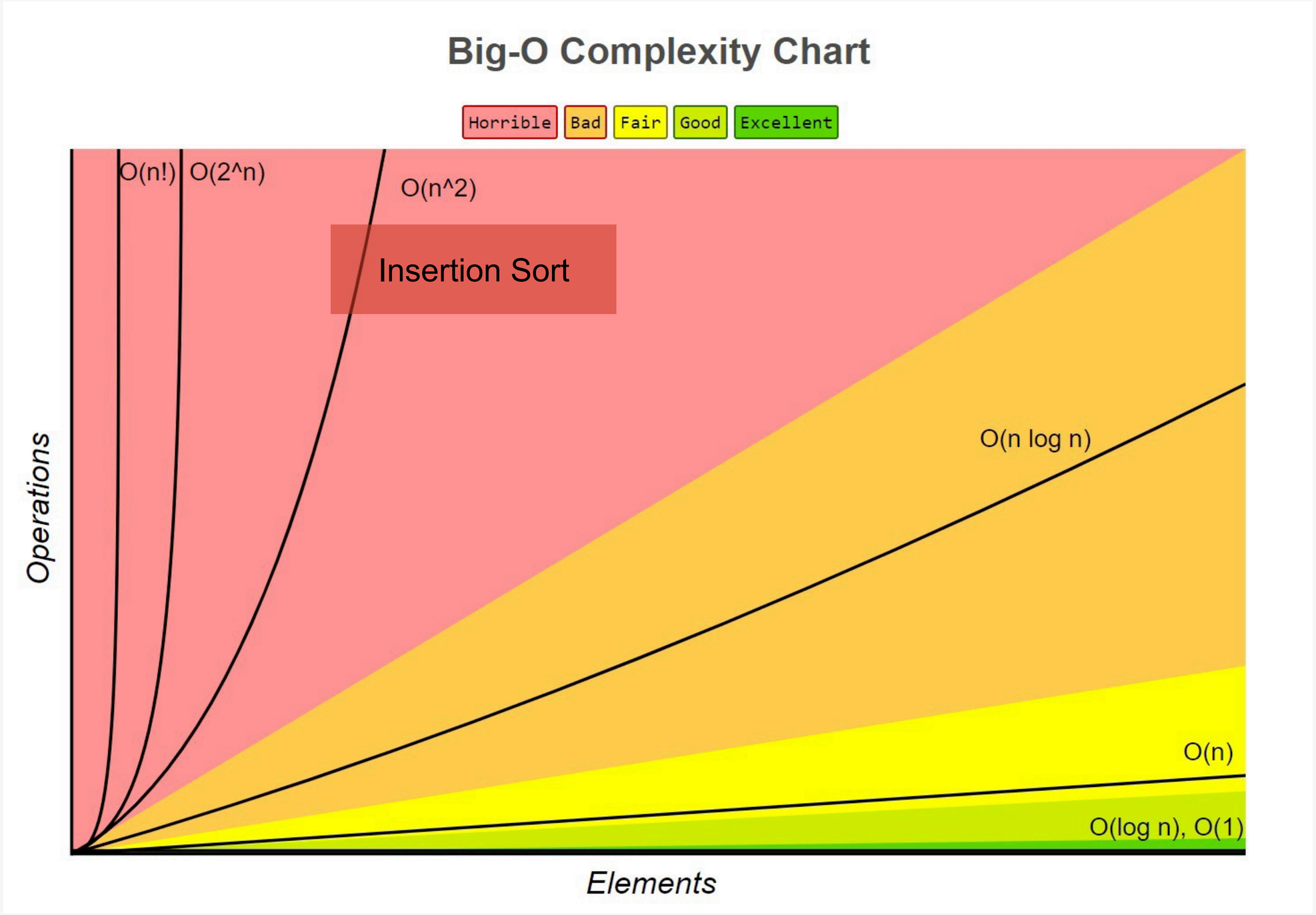
2 1
arr[j-1] > arr[j]
swap!



sorted!

Insertion Sort

- Two while loops, so $O(n^2)$
- Not very efficient!
- Avoid if you can, though it’s the simplest sort to implement


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
Insertion Sort
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Lab 21 Review

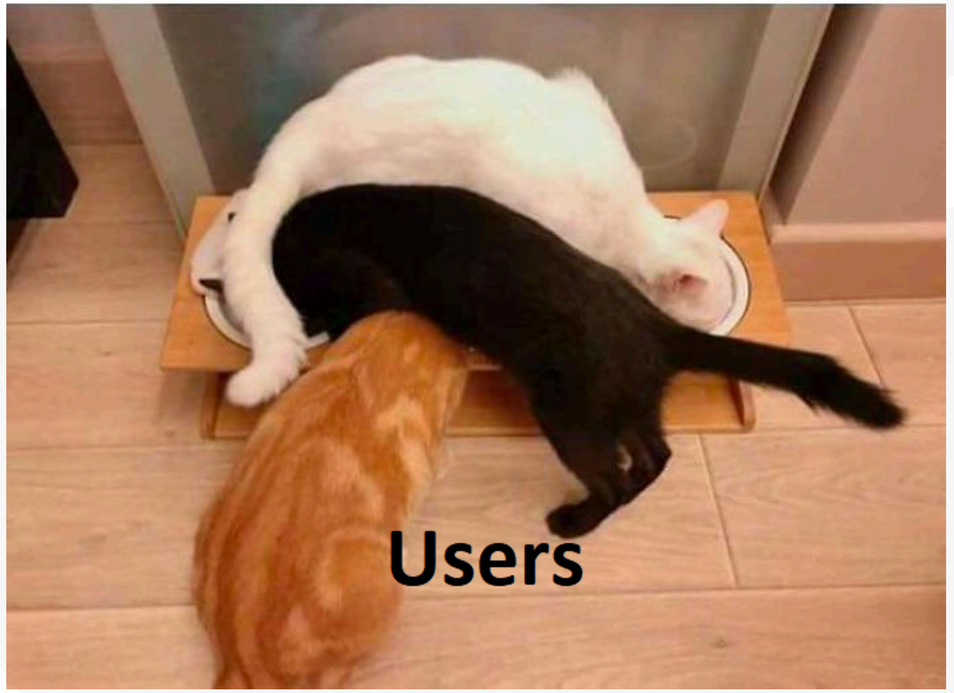
Class 22

React Testing and Deployment

seattle-javascript-401n16

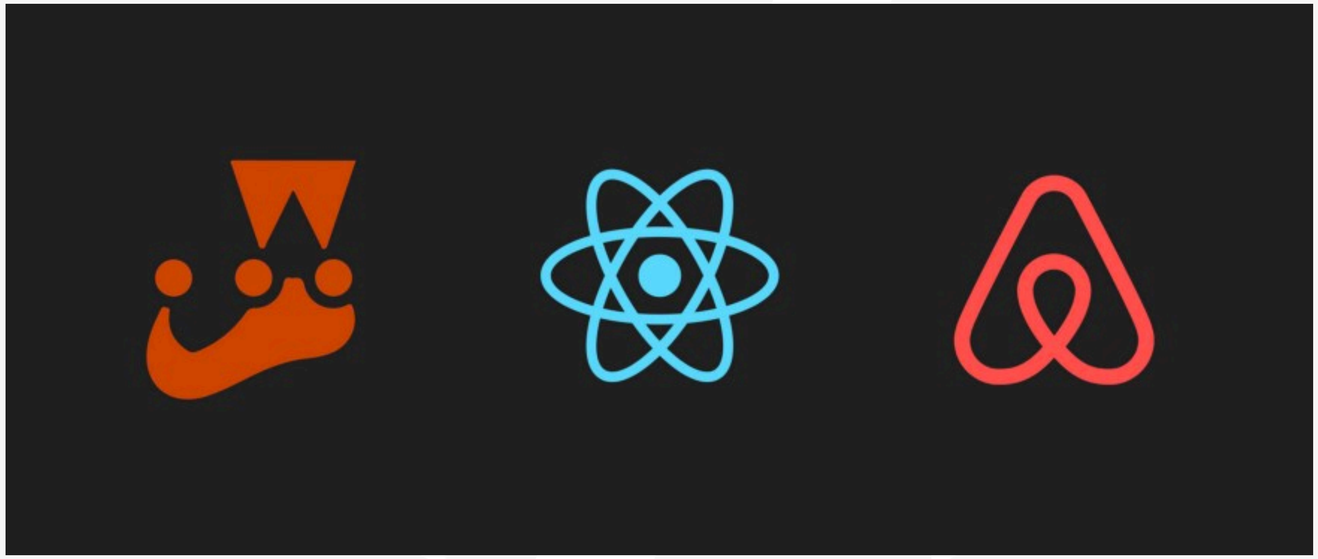
Testing UI

- We can check either
  - That the rendered HTML matches our expectations
    - Can be tricky with TDD
  - If the state and props match our expectations during user interaction



Enzyme

- Another package we use in addition to Jest
- Allows us to search HTML using selectors (like JQuery!)
- We can fake user action such as button click or form input
- We can access component state



Lab 22 Overview