Code Fellows

Class 05 + DSA 01

Linked Lists and Data Modeling with NoSQL

seattle-javascript-401n14

Lab 04 Review

Code Challenge 04 Review

Vocab Review!

What is a database?

What is a database?

A database is a structured collection of data, usually stored outside our applications altogether. Databases want to be secure, so they limit how our applications can access them.

What is CRUD?

What is CRUD?

CRUD stands for Create, Read, Update and Delete. CRUD usually refers to the standard operations you can do upon a database. You can create data, read data, update data contents and delete data.

What is data modeling?

What is data modeling?

Data modeling is a way to define the types of data in your application (usually using a schema), as well as the relationship between the different types of data.

What is a schema?

What is a schema?

A schema is a blueprint or structure definition for a piece of data / object. In JavaScript, we usually define a schema as an object with the same keys as our data, and with each key-value being an object containing a `required` boolean and a `type` definition.

What is a uuid?

What is a uuid?

UUID stands for “Universally Unique Identifier”, and it is usually a 128-bit number (in hexadecimal). We use UUIDs as a unique way to refer to pieces of our data. Think of it like a social security number for our data objects; it’s a secure and unique way to reference a piece of data.

What is SQL?

What is SQL?

SQL stands for “Structured Query Language”, and it is used to access data stored in tables. If our database is stored in a table format, SQL lets us do operations such as `SELECT id FROM table WHERE id = “”`

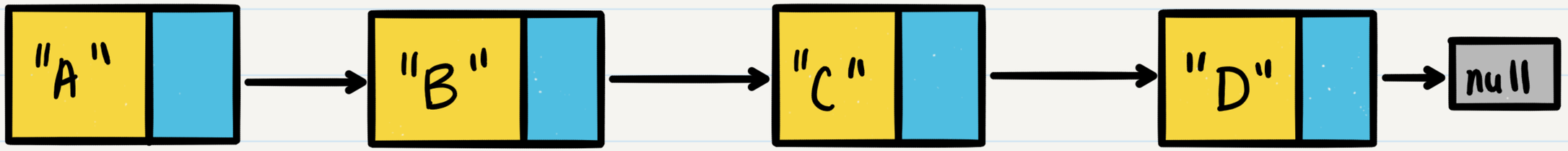
What is a Linked List?

What is a Linked List?

A linked list is an ordered collection of data, where each item in the list contains a **reference** or **pointer** to the next item in the list. Unlike an array which has a set length and uses an index to navigate, linked lists have no set length and are only navigated by individual items pointing to the next item.

Why Linked Lists?

- In languages other than JavaScript, it’s usually difficult to change the size of an array (we give the array a size at init)
    - Takes time and is usually a larger copy of the original
- When arrays are defined in those languages, we define their size and that much space is reserved in memory
- Linked lists instead is just a collection of independent objects that say where to go **next**


"A"
"B"
"C"
"D"
null

```
12
13      if (!this.head) {
14        this.head = node;
15        return this;
16      }
17
18      let current = this.head;
19
20      while (current.next)
21        current = current.next;
22
23      current.next = node;
24      return this;
25    }
26  }
```

```
Head:  null
New Head:  Apple -> null
Done Appending
======================

Head/Current:  Apple -> null
Setting current.next:  Banana
Done Appending
======================

Head/Current:  Apple -> Banana
```

Demo

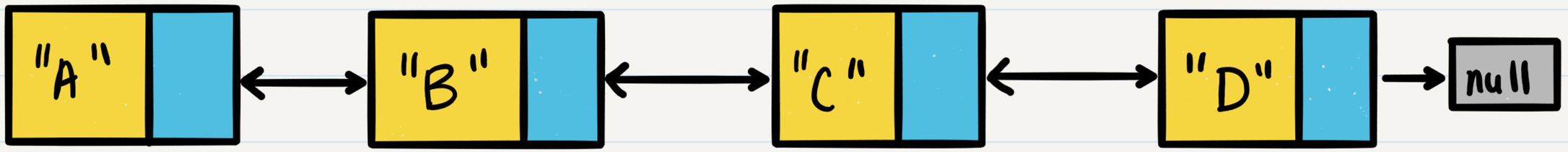
cc-05/demo/ linked-lists

Linked lists are all about using references to find the next (or even previous!) item in the list.

Unlike arrays, because linked lists are all reference based, it's very easy to reorganize or grow the list.

Singly vs. Doubly Linked List

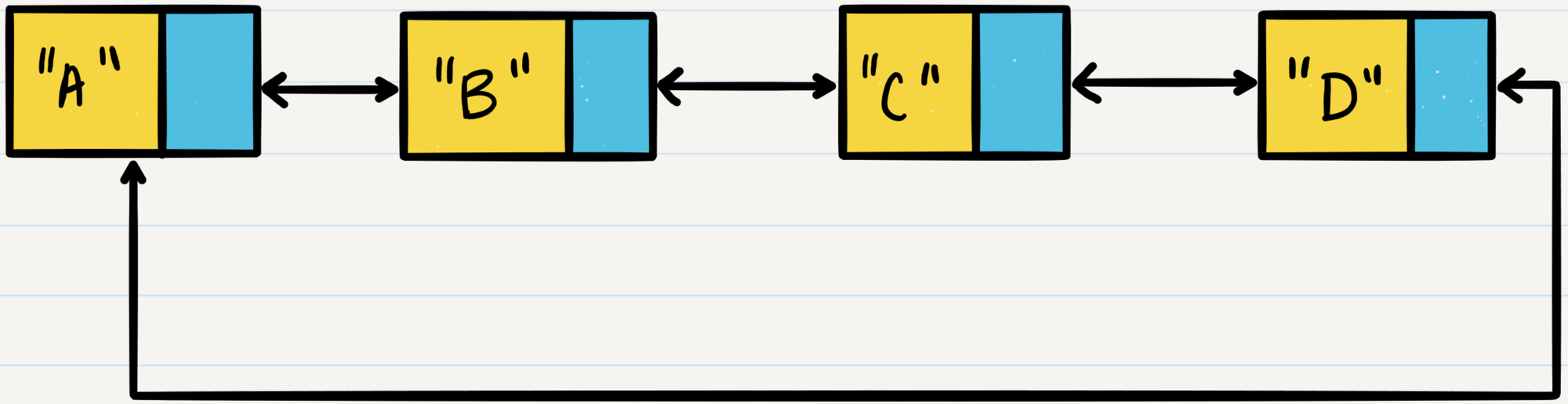
- **Singly**:
  - Each node only has a `next` pointer (last node has `next` set to `null`)
  - There's only one **direction to traverse** the list: `head` >> `nodes…` >> `null`
- **Doubly**:
  - Each node has a `next` and `prev` pointer (`head` has `prev` set to null)
  - You can traverse forwards and backwards: `head` <> `nodes…` <> `tail` > `null`


"A"
"B"
"C"
"D"
null

Circular Linked List

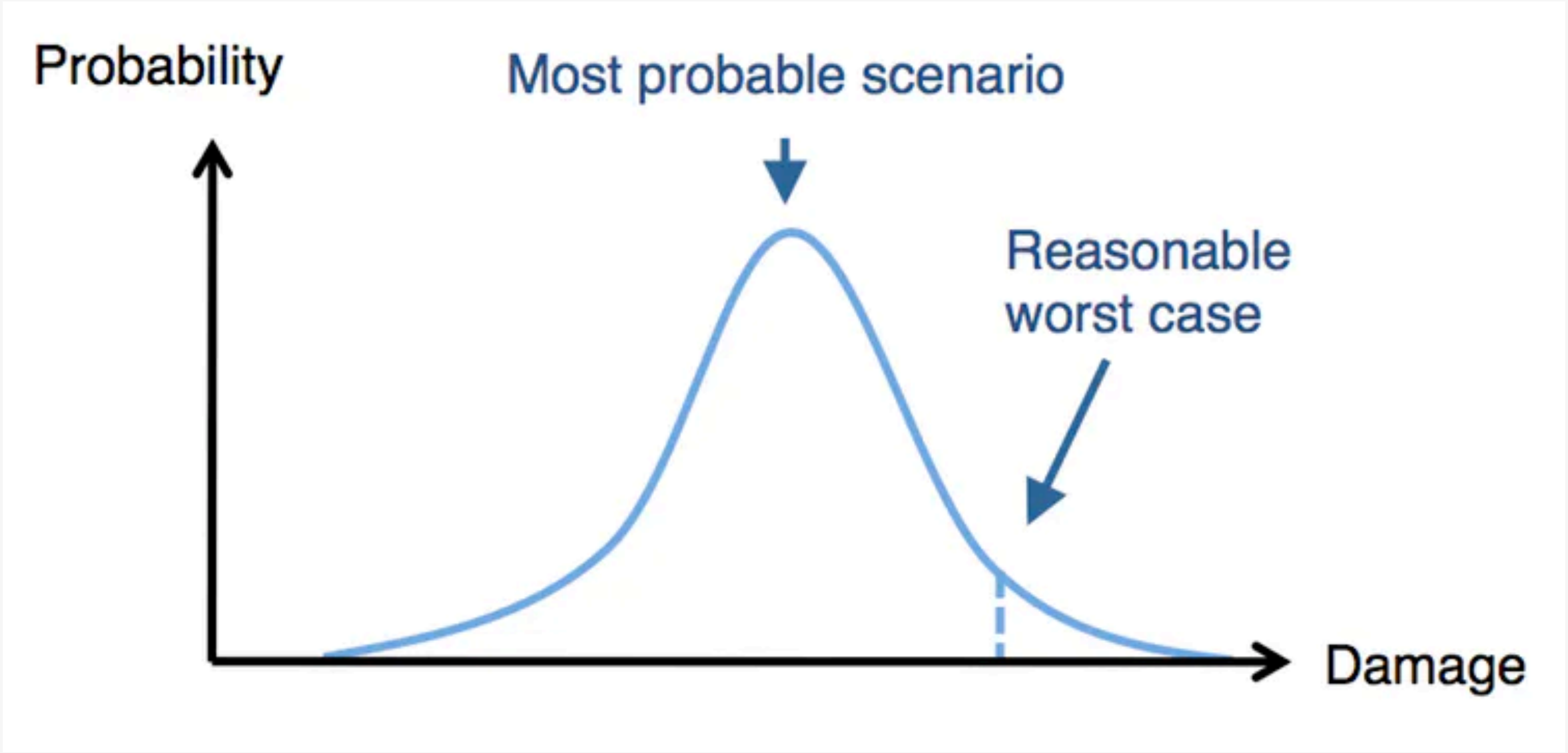
- For both Singly and Doubly linked list, to get from the `head` > `tail` you have to look at `n` items, where `n` = number of nodes
- In a **circular linked list**, `tail.next` = `head` and `head.prev` = `tail` so you can get from the `head` to the `tail` by just going to `head.prev`.

  We've reduced our search from `n` actions to `1` action


"A"
"B"
"C"
"D"

Worst Case Scenario

- When we try to decide what kind of linked list to use, we often want to think of time and space
    - We want to focus on the **worst case scenario** - what is the longest time / largest space we should account for?
- What if `n` (the number of items in our linked list) was a large number? What if it was almost infinity?


Probability
Most probable scenario
Reasonable
worst case
Damage

So What is Big O?

- When we try to figure out the worst case scenario in time and space for an algorithm, we use **big-o notation** to signify this
- We also use `n` in our big-o notation; it signifies the size of an input (think of arrays with a length; `n` is equal to the length)
- `O(n)` means that in the worst case, this function will take as long as the length of the input
- We reduce any constant numbers (not dependent on input) to 1

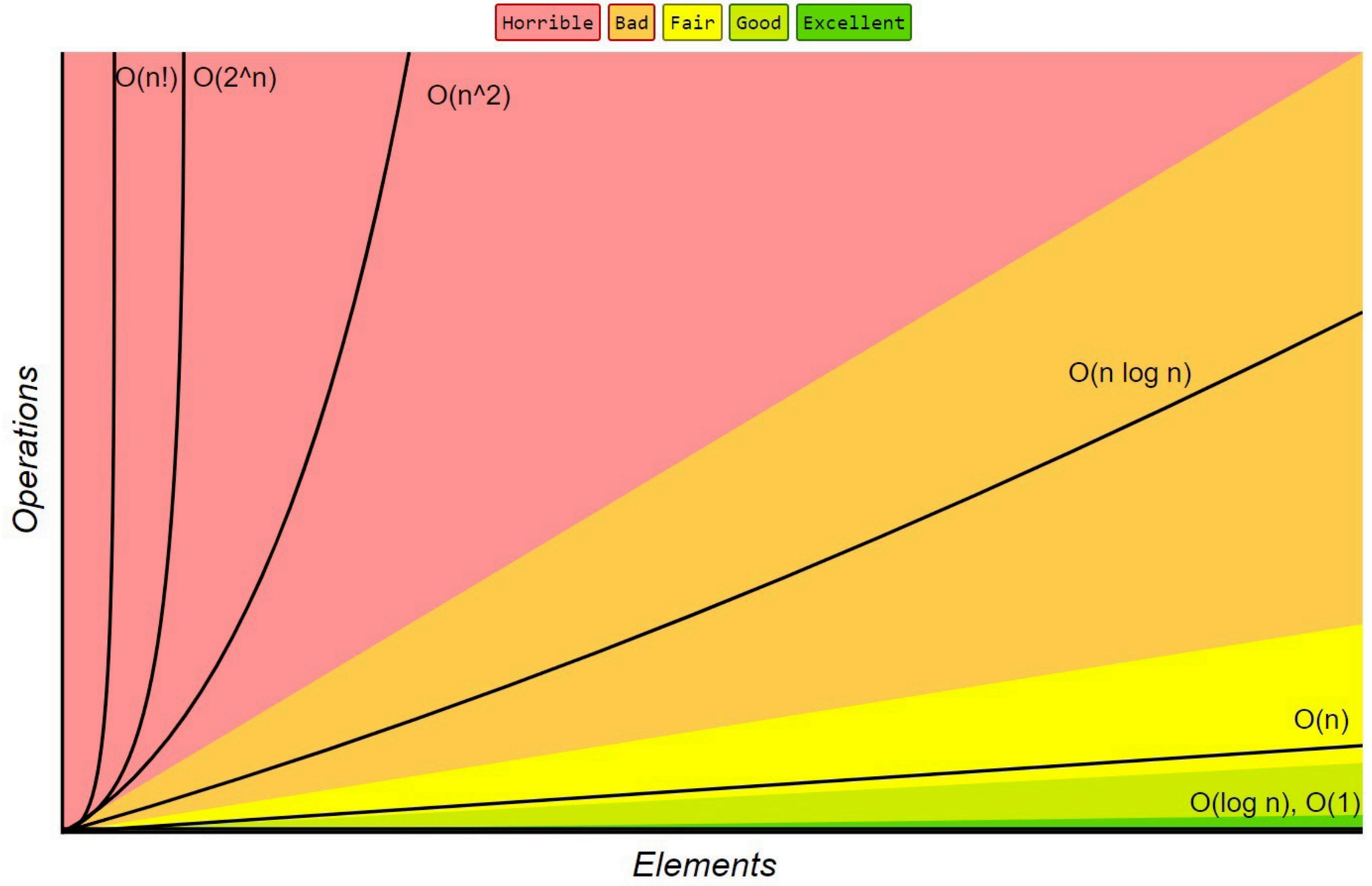
Big O to Know

- There are a couple of common big-o values for algorithms:
  - `O(1)` - this function takes a constant time/space, not dependent on input
  - `O(n)` - this function grows in time/space in a 1:1 ratio with the input
  - `O(n²)` - this function grows exponentially when n grows, meaning the larger the input, the even larger the time/space (`n * n`)
  - `O(log n)` - Anyone remember log functions from math class?

Big O to Know

- There are a couple of common big-o values for algorithms:
  - `O(n log n)` - slightly worse than `O(n)`
  - `O(2^n)` - this is a weird one, but usually happens when we call two functions **recursively**
  - `O(n!)` - the exclamation point stands for factorial, which means `n * n-1 * n-2 * n-3 * … until n = 1`

Big-O Complexity Chart



```
function fibonacci(n) {
  if (n <= 1) return n;
  else return fibonacci(n - 1) + fibonacci(n - 2);
}

console.log(fibonacci(12));
```

```
Not there yet: 12
Not there yet: 11
Not there yet: 10
Not there yet: 9
Not there yet: 8
Not there yet: 7
Not there yet: 6
Not there yet: 5
Not there yet: 4
Not there yet: 3
Not there yet: 2
Base Case: n = 1

Base Case: n = 0

Base Case: n = 1

Not there yet: 2
Base Case: n = 1
```

Demo

cc-05/demo/bigo

Big O often refers to how many times a function / piece of code is called based on an input (time) or how many items are created based on an input (space)

12:06:57 0.07 K/s 9%

Thread

Sunjay liked

jwcarroll
@jwcarroll

Alternative Big O notation:

O(1) = O(yeah)
O(log n) = O(nice)
O(n) = O(ok)
$O(n^2)$ = O(my)
$O(2^n)$ = O(no)
O(n!) = O(mg!)

8:10 PM · 06 Apr 19 · Twitter for Android

3,665 Retweets **9,265** Likes

jwcarroll @jwcarroll · 8h

Tweet your reply

O(1)	😎	Speed doesn't depend on the size of the dataset
O(log n)	😁	10x the data means 2x more time
O(n)	😕	10x the data means 10x more time
O(n log n)	😖	10x the data means about 20x more time
$O(n^{2})$	😫	10x the data take 100x more time
$O(2^{n})$	😱	The dilithium crystals are breaking up!

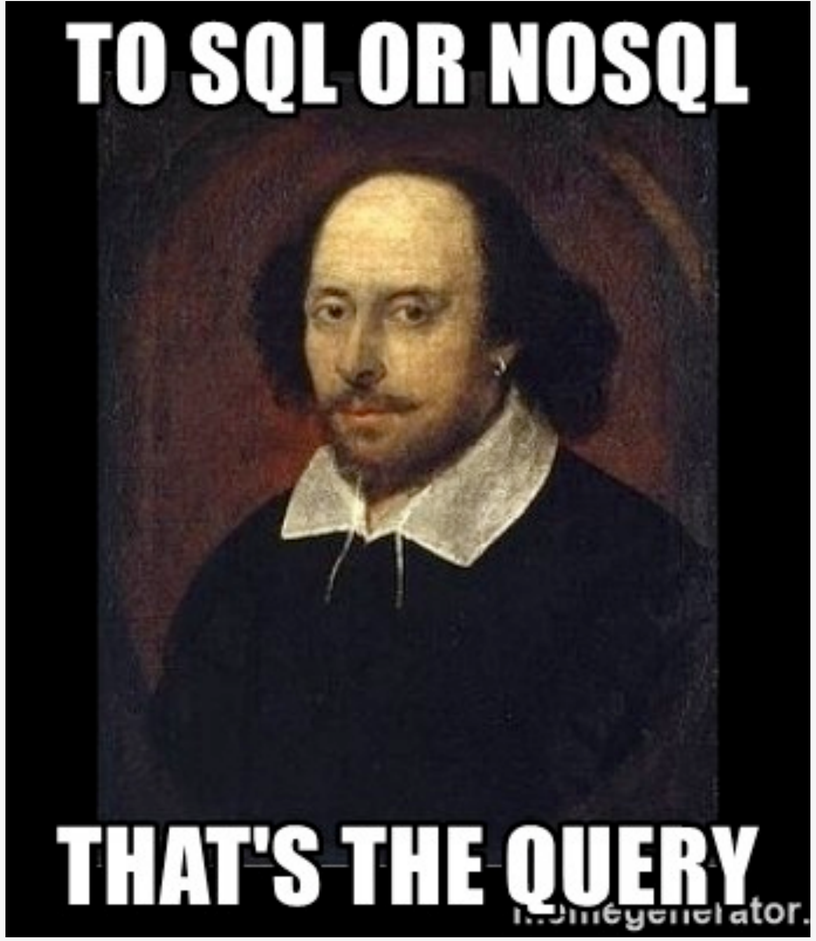
Big O and Linked Lists

- How long does it take to search for something in a singly or doubly linked list?
    - What we're searching for might be at the end of the list (**worst case**) so we would have to search each item. If there are `n` items, that would be `O(n)`
- What about a circularly doubly linked list?
    - Because we can go from `head` >> `tail`, the longest gap is actually `head` -> `midpoint`, so it's kind of like `O(0.5n)`
    - We always turn constants into `1`, so in the end it still is `O(n)`

Okay, back to Data Modeling

SQL vs. NoSQL

- You did some **SQL** (hopefully) in 301
    - `SELECT id FROM table WHERE id = ###`
- **SQL databases** store data in tables, and accessing data requires operations on the those tables (combine, search, sort)
- **NoSQL databases** are all about storing data in files, and overall NoSQL databases allow for more flexibility in storing data (doesn't have to follow a strict schema)
    - Didn't we do something like this in Lab 04?



MongoDB

- This is a very popular NoSQL database system
- Remember reading and writing to files in Lab 04?
    - **MongoDB** will now do that for us
    - MongoDB will let us have simpler CRUD functions in our `model.js`
    - The MongoDB CRUD functions **will also return Promises**!
- Dependency: `npm install mongodb`

Get Validator Outta Here!

- Because NoSQL databases like MongoDB don't really enforce a schema, we need to do some of that on our own
    - We originally did this with a sanitize function (which was really just validator)
    - Now we're going to do this with a package called **mongoose**
- Dependency: `npm install mongoose`

Finally - How to Run a DB?

- Databases are usually run on a **server**
  - A server is a program / service that runs outside of our application, with the goal to serve our application data
  - Every website is a loaded from a server (`www`, `localhost`)
  - We can start our MongoDB server using:

    ```
    mongod --dbpath=/PATH/TO/A/DATABASE FOLDER
    ```

Your database is independent from your application

Accessing Your Database

- Once you've started your database server, you can use the command "`mongo`" in your terminal to hook directly into the database
- Using unique commands, you can navigate around the database and do any CRUD operation plus so much more!
    - MongoDB Documentation
- `db` = database, `collection` = a type of object in that database (`peoples`, `teams`)

Useful Mongo Commands

Handy Cheatsheet!

`mongo`	Start your mongo command line (lets you run commands on your open mongo database)
`show dbs`	Show all the databases on your current mongo server (yes you can have multiple databases on a server!)
`show collections`	Show all the collections (aka: data models) on a database
`use <database name>`	Focus your operations on a specific database
`db.<collection name>.find()`	List all the items in the current database collection
`cls`	Clear screen

Useful Mongo Commands++

Handy Cheatsheet!

`db.<collection name>.deleteOne(query)`	Delete an item that matches the query
`db.<collection name>.insert(record)`	Insert items into a collection
`db.<collection name>.count(query)`	Counts the items that match the query

```
const mongoose = require('mongoose');
const db = 'mongodb://127.0.0.1:27017/app';
const config = {
  useUnifiedTopology: true,
  useNewUrlParser: true
};

mongoose.connect(db, config);

const Teams = require('./models/teams.js');
const People = require('./models/people.js');

let teams = new Teams();
let people = new People();

teams.getByField({ name: 'Red Heron' }).then(data => console.log(data));
```

```
{ _id: 5da1d8eb2731d3aac536e602, name: 'Red Heron', __v: 0 }
```

Demo

class-05/ demo/mongo

There's a lot to learn with MongoDB! Practice what we did here in this demo until it becomes more familiar; it's okay to not get all of it the first time round!

How to Test??

- Testing a database can be tough because you want to hook into it without editing it
- We use a plugin called `supertest`; from your end you shouldn't need to do much other than add this file!
- New dependencies:
  - `npm install supertest`
  - `npm install mongodb-memory-server`

What's Next:

- Due by Midnight tonight: **Learning Journal 05** + **Partner Power Hour Report #1**
- Due by Midnight Sunday: **Feedback Survey Week 3** + **Career Coaching (2 assignments)**
- Due by Midnight Monday: **Code Challenge 05**
- Due by 6:30pm Tuesday:
    - **Lab 05**
    - **Read: Class 06**
- Next Class:
    - **Class 06 - HTTP and REST**

Questions?