3rd

Quiz 02 Review Session

COMP 210 / 2024 Summer Session I

Ajay Gandecha

Quiz 02 Format

- 30 minutes at the start of class.
- *On paper* - bring a pencil!
- **Question Types:**
  - Multiple choice, T/F, select all that apply, fill in the blank,
  - *No code writing on this quiz - but be able to trace given Java code!*

Possibly draw a diagram* (does not have to be extremely detailed)

Exercise Check-In Question

- … Specifics covered in lecture (5/28)

On Quiz 02

- Big-O Analysis
  - Analyzing code snippets for runtime analysis, including recursive code
- The `List` Abstract Data Type
  - Understand `ArrayList` and `LinkedList` on the heap
  - Explain trade-offs between both, justified using big-O notation

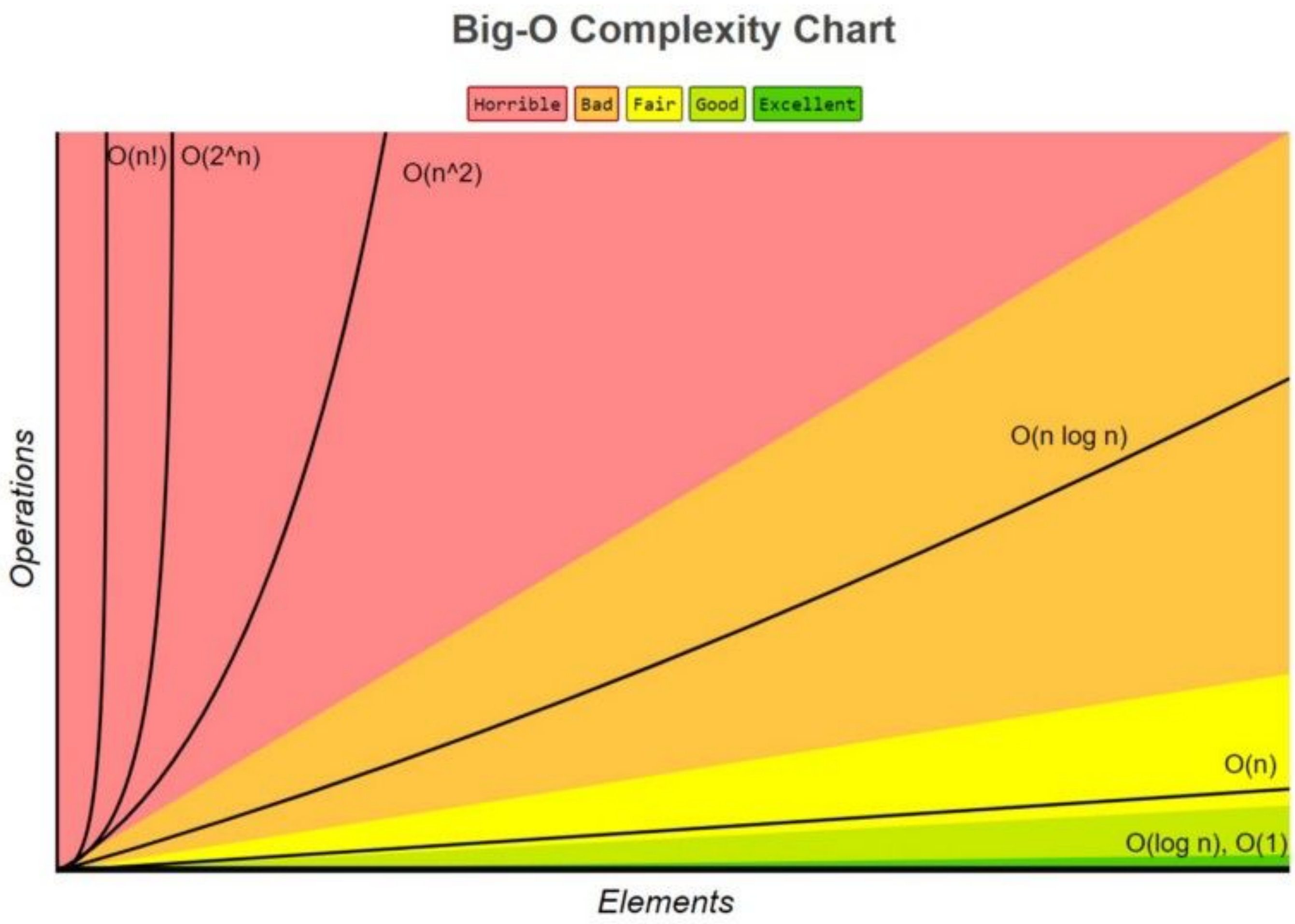
Review: Big-O Analysis

- **We need a way to determine *how efficiently* algorithms run.**
  - We need notation to be able to compare the *efficiency* of algorithms.
  - This is called Big-O Notation.

"time"

- **We can tell how efficient algorithms run by comparing *how many operations* an algorithm performs compared to the *number of inputs we supply to it*.**

Big-O Graph Comparisons



Recursive Example 1

```
void foo(int n) {
  if(n<=0) return 1;
  return 1 + foo(n-1);
}
```

bc

rc

f

5

C+ f(n-1)

O(N)

O(W) & O(D/H)

$O(1) + O(N) = O(N)$

f(5) = 1+f(4)

f(4) = 1+f(3)

f(3) = 1+f(2)

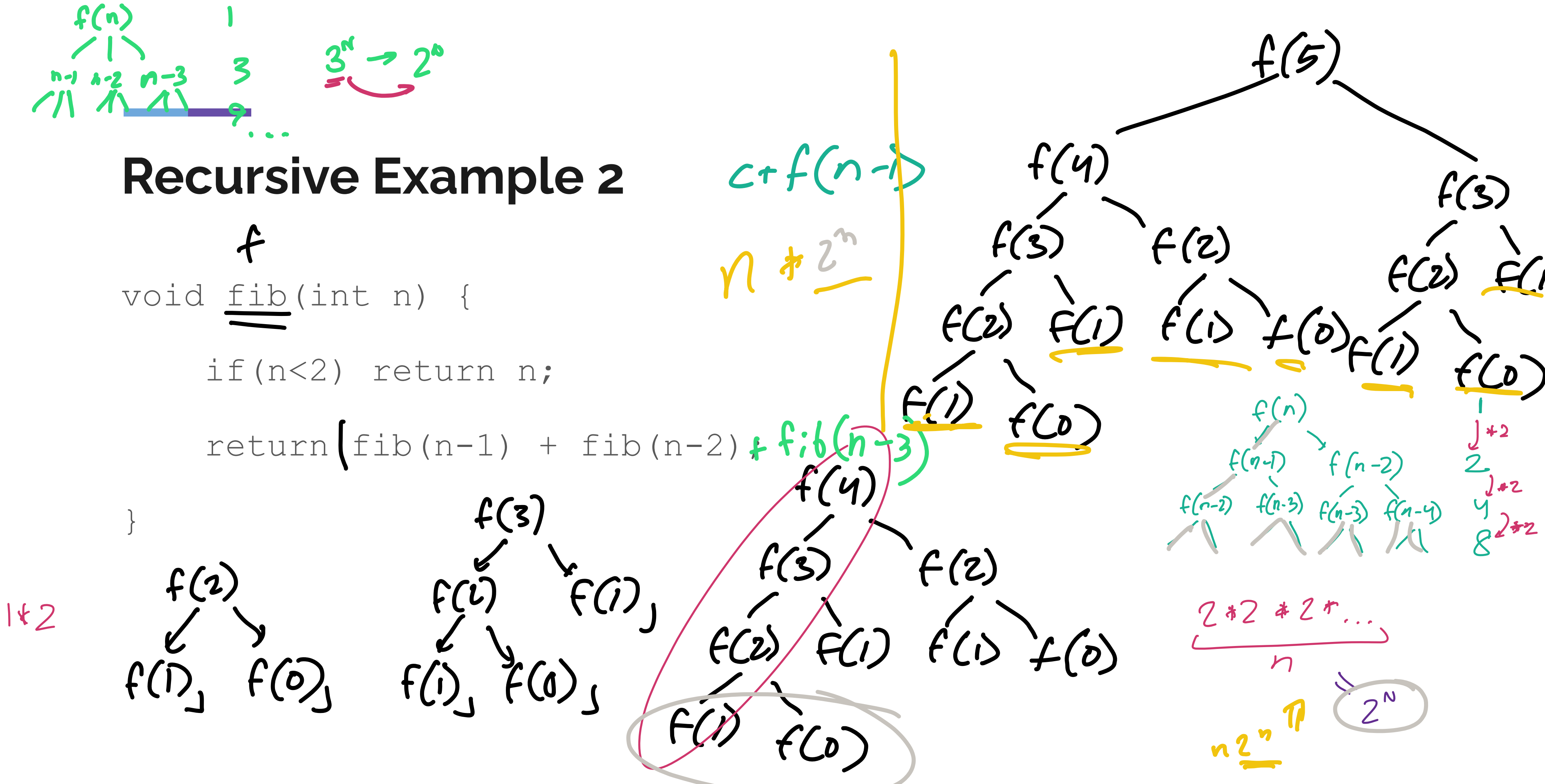
f(2)

f(1)

f(0)

Recursive Example 2

```
void fib(int n) {
    if(n<2) return n;
    return fib(n-1) + fib(n-2);
}
```



Recursive Example 3

```
void fib(int n) {
    if(n<=1) return n;
    return fib(n/2) + fib(n/2);
}
```

f

16

$2^x = n$

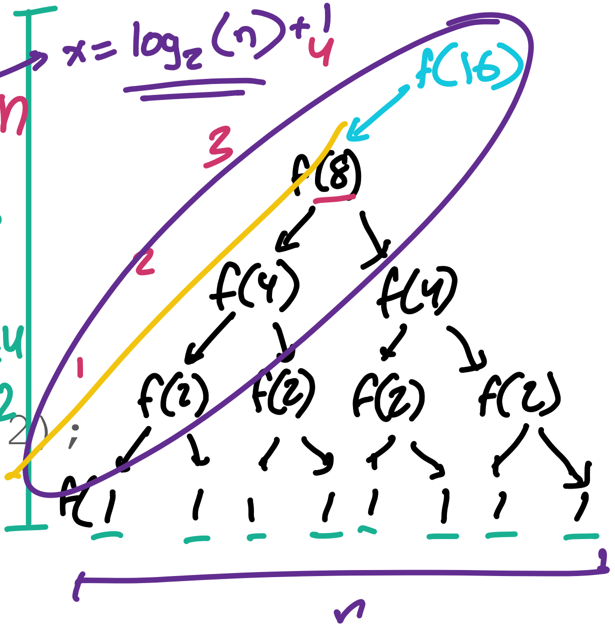
$2^4 = 16$

$2^3 = 8$

$2^2 = 4$

$2^1 = 2$

$2^0 = 1$



$O(n) + O(n)$

$O(n) * O(\log n) = O(N \log N)$

Recursion Big-O Guide

Given a recursive case:

one recursive call

① $c \pm f(n-\#) \Rightarrow O(N)$

constant   some #

input size decreases linearly (+/-)

② $f(n-\#_1) \overset{*}{+} f(n-\#_2) + \ldots \Rightarrow O(2^N)$

any operation

③ $f\left(\frac{n}{\#_1}\right) + f\left(\frac{n}{\#_1}\right) \Rightarrow O(N\log N)$

④ $c \pm f\left(\frac{n}{\#_1}\right) \Rightarrow O(\log N)$

$\log N$

$f(8)$

$\downarrow$

$f(4)$

$\downarrow$

$f(2)$

$\downarrow$

$f(1)$

1

`ArrayList` Representation

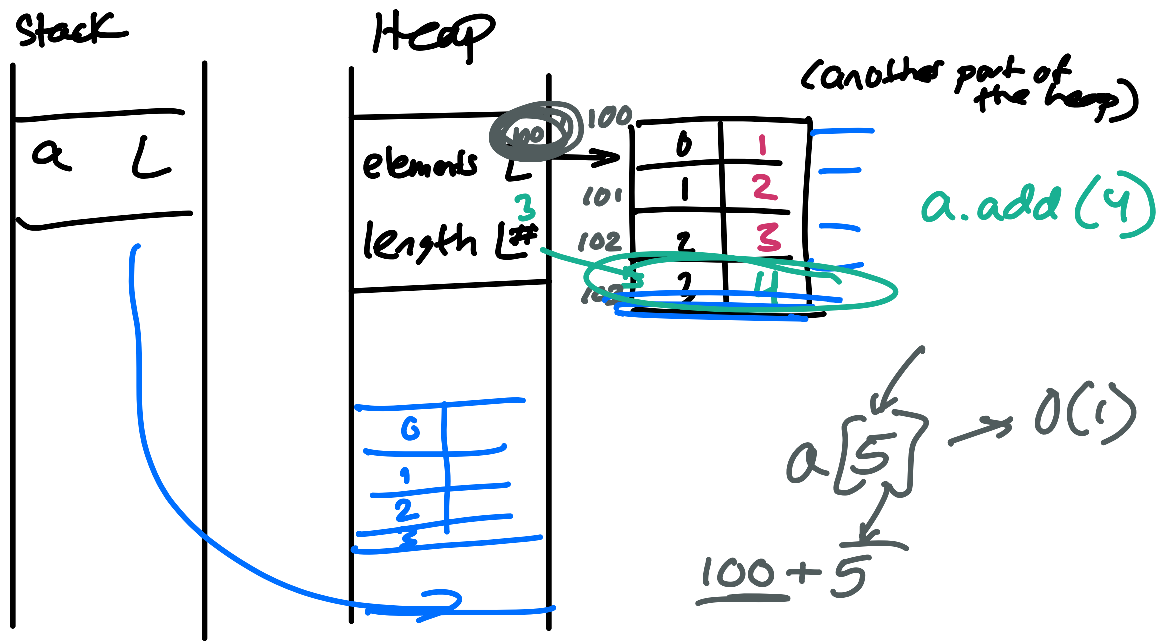
I List

C ArrayList   C LinkedList

- Recall that `List` is an abstract data type.
- **`ArrayList`** is one implementation of the `List` interface.

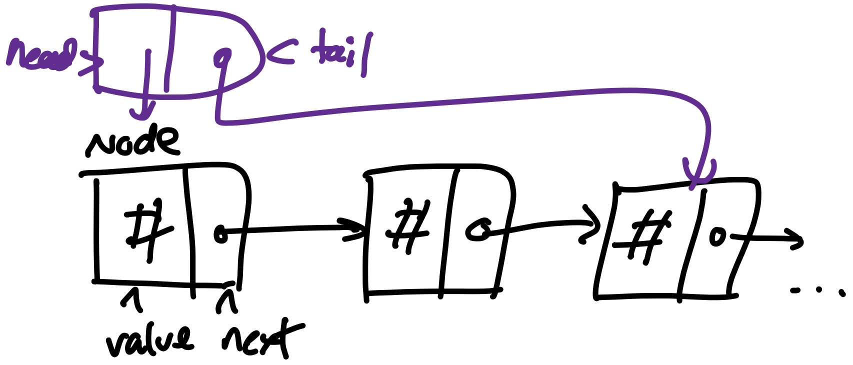
`ArrayList` Representation

a = [1, 2, 3]

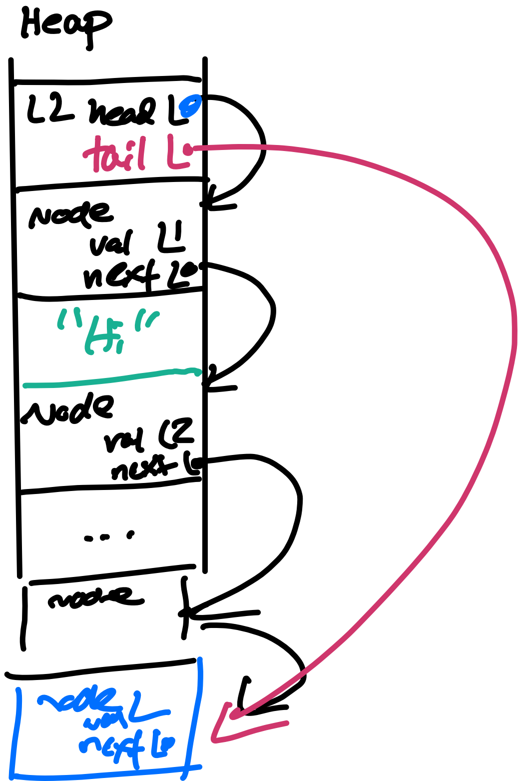


LinkedList Representation

- **LinkedList** is another implementation of the List interface.



`LinkedList` Representation



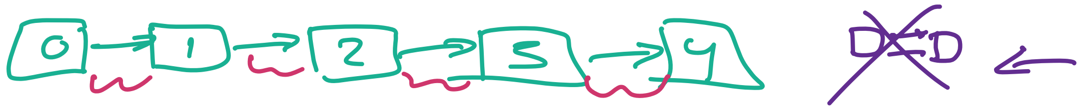
Deriving `List` Time Complexities

★ = ammortized

add()

	get(0)	get(i)	get(n)	insert(0)	insert(i)	insert(n)	remove(0)	remove(i)
ArrayList	O(1)	O(1)	O(1)	O(N) Avg = O(N)	O(N) Avg = O(N)	O(N)★ Avg = O(1)	O(N)	O(N)
LinkedList (Head only)	O(1)	O(N)	O(N)	O(1)	O(N)	O(n)	O(1)	O(N)
LinkedList (Head and Tail)	O(1)	O(N)	O(1)	O(1)	O(N)	O(1)	O(1)	O(N)

O(N) for remove(n) also ★


0
1
2
3
4