Quiz 01 Review Session

COMP 210 / 2024 Summer Session I

Ajay Gandecha

Quiz 01 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!*

On Quiz 01

- Encapsulation
  - Marking fields as `private`, exposing get / set functionality as methods.
- Abstract Data Types
  - Using Java `interface`s, write classes *implementing* interfaces.
- Big-O Analysis
  - Analyzing code snippets for runtime analysis

→ Not on this quiz:
- Git / GitHub
- JUnit ✓
- BigO of recursion Qz2
  ↳ next time

Encapsulation

Idea that we want to *control* how our code interacts with objects' fields.

Key Points:

- Mark fields as `private`.
- Create *getter* and *setter* methods to access fields.

AA act = new AA("Ajay" —);

✗ act.creditCard Numbe

```
public class AmazonAccount{

    private
    ~~public~~ String name;

   private
    ~~public~~ String creditCardNumber;


    public AmazonAccount(String name, String ccn) {

        this.name = name;

        this.creditCardNumber = ccn;

    }

}
```

- Does the `AmazonAccount` class follow the principles of encapsulation?

```
public class AmazonAccount{

    public String name;

    public String creditCardNumber;



    public AmazonAccount(String name, String ccn) {

        this.name = name;

        this.creditCardNumber = ccn;

    }
    public void purchaseItem(Item i) {...}
}
```

- Does the **`AmazonAccount`** class follow the principles of encapsulation? No.
    - Fields are marked `public`.
    - There are no getter and setter methods.

Rewriting the `AmazonAccount` class:

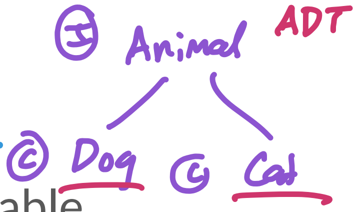
```
public class AmazonAccount {
  ...
   public String getName() {
          return this.name;
   }
                                     public void setName(String newName){
                                          this.name = newName;
                                     }

① No getter
② public String getCreditCardNum() {
                                                              or 5
      return this.creditCardNumber.substring(ccn.length-4, ccn.length)
                                             ------------  ----------
                                                  s             e
   }
}
```

.ss(2,3)

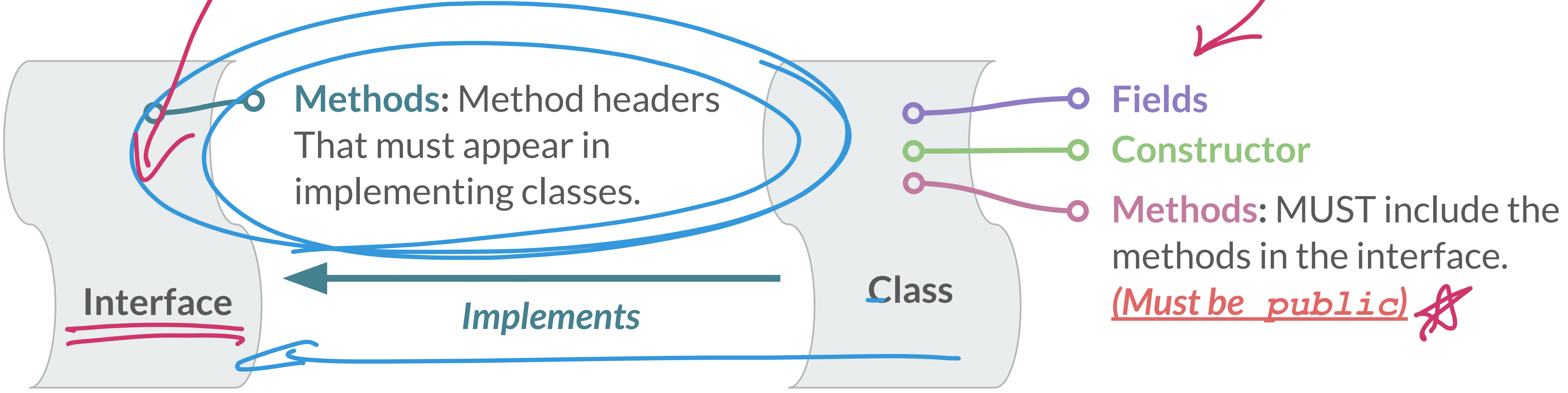
"Hello" → e

Abstract Data Types

- Idea that we want to define what a type can do *without worrying about the actual implementation.*
    - Expressed using the Java `interface`.



- Interfaces are like **contracts** that implementing classes ***must*** adhere to.
    - Indicates that certain `public` methods are **guaranteed** to be available.
    - One or more classes can implement an interface.

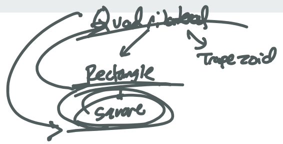


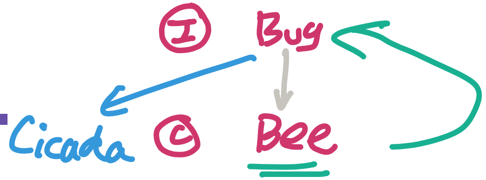
Methods: Method headers That must appear in implementing classes.

Fields

Constructor

Methods: MUST include the methods in the interface. *(Must be `public`)*





interface, ADT

- If class Bee implements the interface Bug:

  ↳ getHoney()

  - `Bee busyBee = new Bee(); // Valid`
  - `Bug busyBee = new Bee(); // Valid`
  - `Bee busyBee = new BugImpl(); // Invalid`
  - *why?*

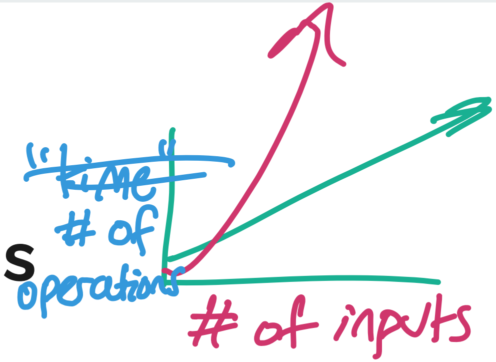
Bug b = new Bee() //Valid, but

b.getHoney() // Invalid

Bee b = new Bee()

b.getHoney() // valid

Big-O Analysis



O(∼)

O(N)

O(N^2)

- **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.

- **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*.**

Simple Example

$|a| = n$

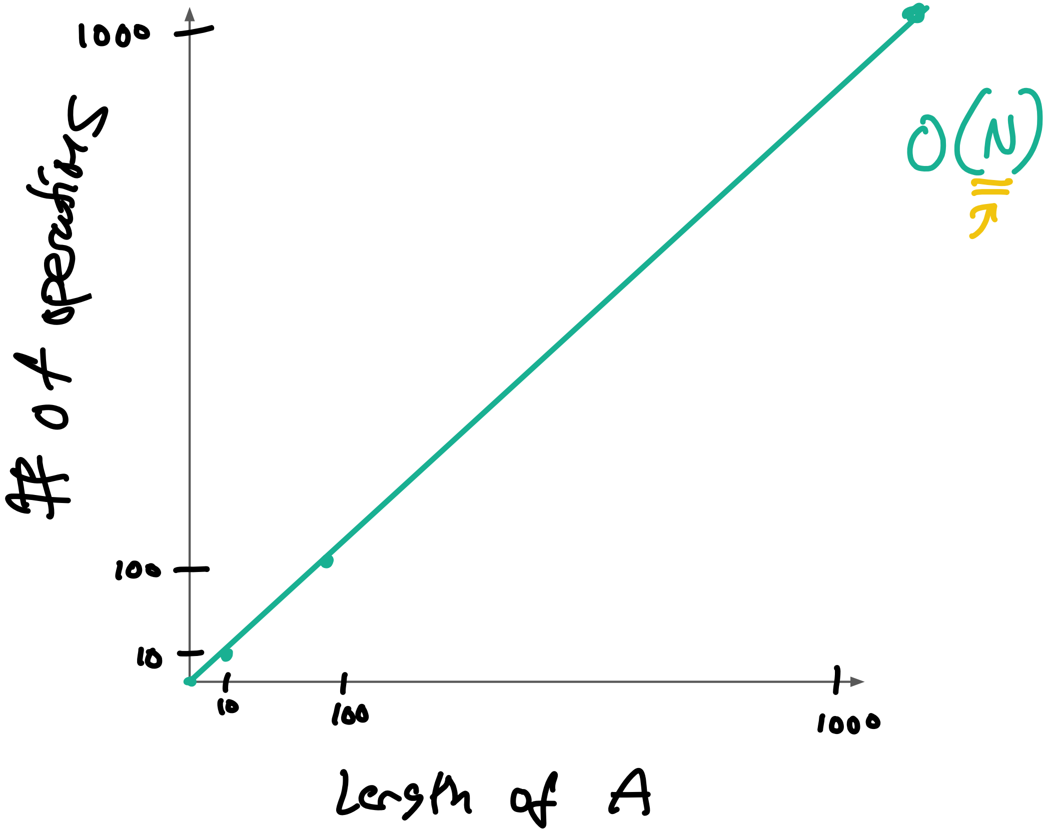
```
void example(int[] a) {
    for(int i = 0; i < a.length; i++) {   O(n)
        System.out.println(i);   O(1)
    }
}
```

$O(N) * O(1) = O(N)$

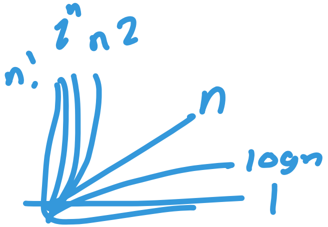
- How many times does the print statement run if a has 1 element? $= 1$
  - *What about 10 elements?* 10
  - *What about 100 elements?* 100
  - *What about 1,000 elements?* 1000

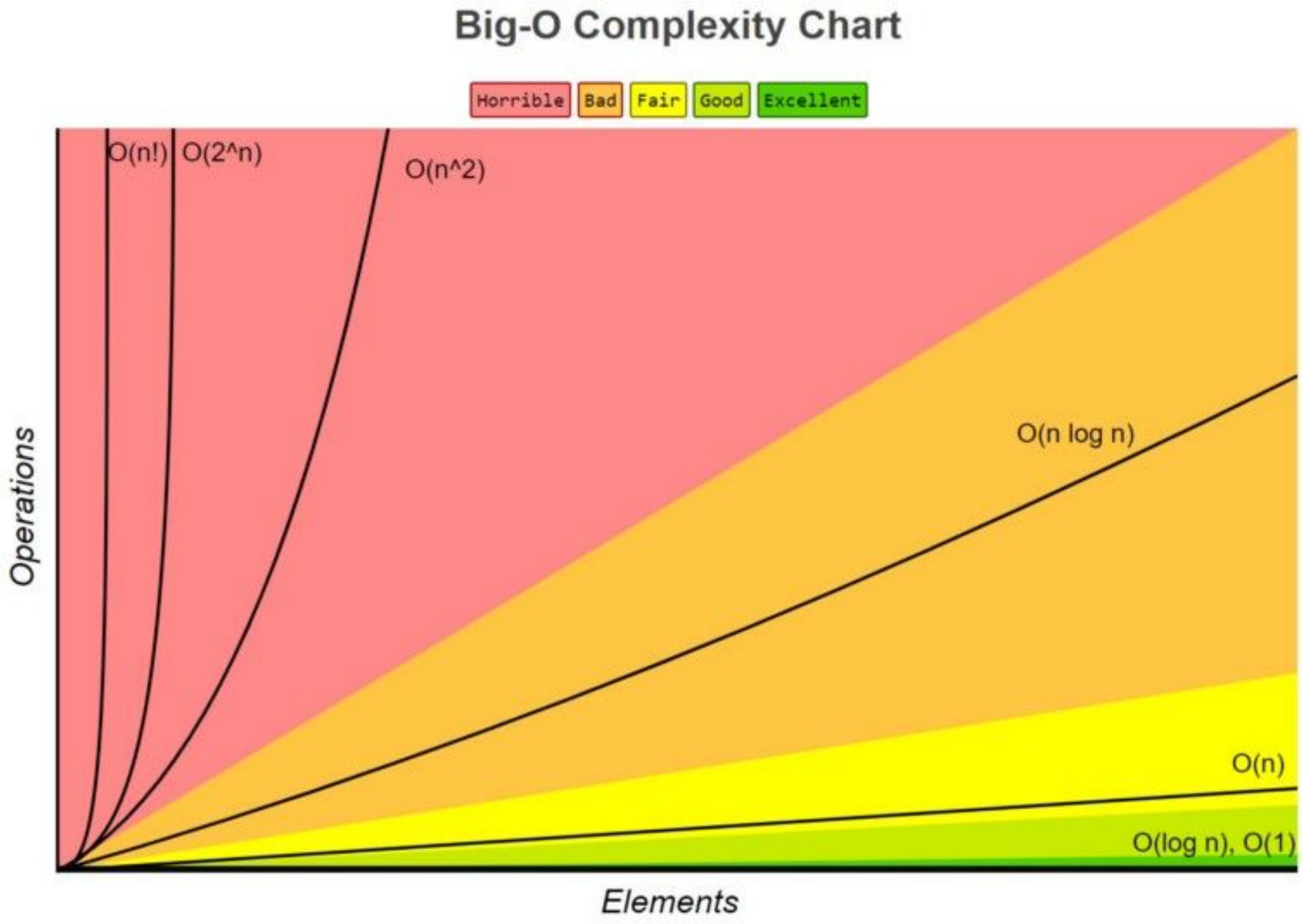
Simple Example

O(N)



Big-O Graph Comparisons





Intermediate Example

|a|=n

```
void example(int[] a) {
                          1000
    for(int i = 0; i < a.length; i++) {  O(N)
                               2000
n *     for(int j = 0; j < (a.length * 2); j++) {  O(2N) ~> O(N)
        2n*  System.out.println(i);  O(1)
        }
    }
}
```

O(2N)

2n*)

O(N) *n

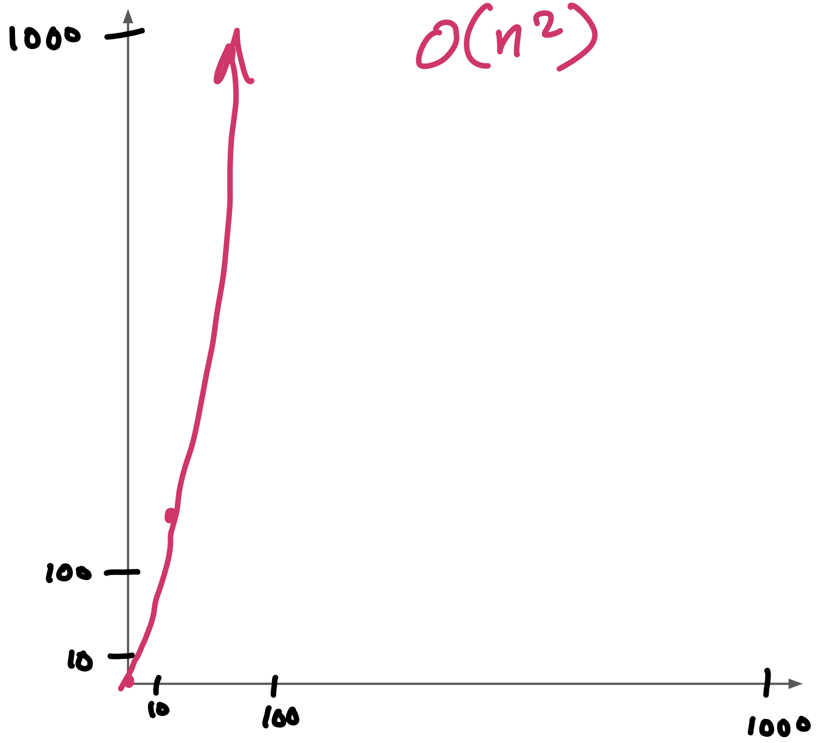
O(2N²)

O(2n * n)

$O(N^2)$

- How many times does the print statement run if a has 1 element? → 2
  - *What about 10 elements?* → 200
  - *What about 100 elements?* 20000
  - *What about 1,000 elements?* 2000,000

Another Simple Example



More Complicated Example

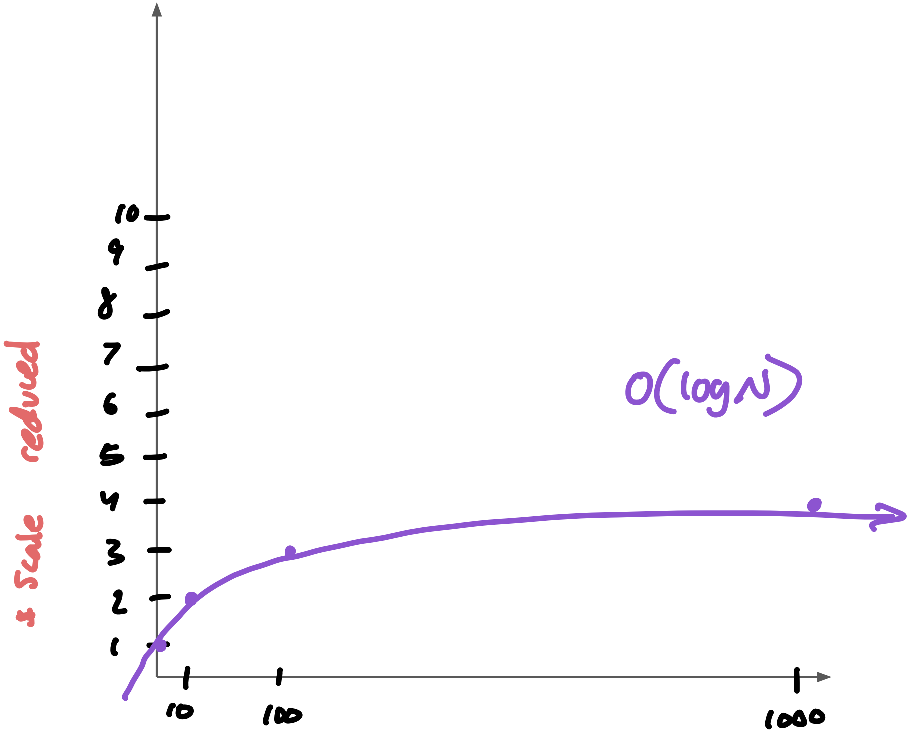
|a| =1000

1000

```
void example(int[] a) {
    for(int i =1; i <=a.length; i*=10) {
        System.out.println(i);
    }
}
```

- How many times does the print statement run if a has 1 element? 1
  - *What about 10 elements?* 2
  - *What about 100 elements?* 3
  - *What about 1,000 elements?* 4

More Complicated Example


O(logN)
* Scale reduced