CS151 Intro to Data Structures

Java Review, Inheritance, Generics

Announcements

- Join Piazza
- Join Gradescope
  - **GPBX65**

- TA Office Hours 7-9:30pm
  - Sun-Fri
- HW0 and Lab1 due Thursday

Outline

- Review: Exceptions and I/O

- Object Oriented Programming
- Inheritance
- Arrays

File I/O

- What Java object can we use to read from files?
  - Is this approach only for files?

Exceptions

code :)

Exceptions

1. Checked Exceptions
   a. `'error: unreported exception **FileNotFoundException;** must be caught or declared to be thrown'`

2. Unchecked Exceptions
   b. `ArrayIndexOutOfBoundsException`
   c. `NullPointerException`
   d. `ArithmeticException`

Exceptions

How do we deal with them?

a) in the caller

a) in the callee

Exceptions

- Exceptions are objects
  - can create with `new` keyword
- Inheritance
  - `NullPointerException` is a `RuntimeException` is an `Exception`
  - `FileNotFoundException` is a `IOException` is an `Exception`

Object Oriented Programming

Software Design Goals

- Robustness
  - software capable of error handling and recovery
- Adaptability
  - software able to evolve over time and changing conditions (without huge rewrites)
- Reusability
  - same code is usable as component of different systems in various applications

Object Oriented Programming aims to achieve these!

What benefits does a Class give us?

Abstraction and Encapsulation - hide internal details of how an Object works, while providing a well defined way to interact with it

Inheritance

- Enables a class to use the properties and behaviors of another class

- Establishes relationships between classes

Towards our goal of reusability!

Inheritance

Student example code

`super`

- `super` refers to the superclass object
- can also be used to reference methods defined in the superclass
  - `super(.....)` references the parent class constructor
- `super.getName()`

Inheritance - constructors

- Constructors are never inherited
- A subclass may invoke the superclass constructor via a call to `super` with the appropriate parameters
- If calling `super`, it must be in the first line of the subclass' constructor
- If no explicit call to `super`, then an implicit call to the zero-parameter `super()` will be made

Method Overriding

- Inherited methods from the superclass can be redefined/changed
  - signature stays the same

- **Let's override toString in our code**

Break for questions

`protected`

- access modifier
  - `public` – world
  - `private` – super class only
  - `protected` – super and subclasses
- subclass inherits all `public` and `protected` instance variable and methods
- What about `private` instance variables?

Type Hierarchy

- Every subclass object is an instance of its superclass
- A superclass object is NOT an instance of the subclass

```
class A {}
class B extends A {}
class C extends B {};
```

Homogeneous Type

- Array requires that the elements are of the same type

```
code :)
```

Object Casting

- Type conversion between super and subclasses
- A superclass is a wider type
- A subclass is a narrower type

code :)

Object Casting

- Down casting - casting an object of a **parent class** type to an object of a more specific **child class** type
  - Dangerous!!

```
B b2 = (B) a1; //ClassCastException!
```

Object Casting

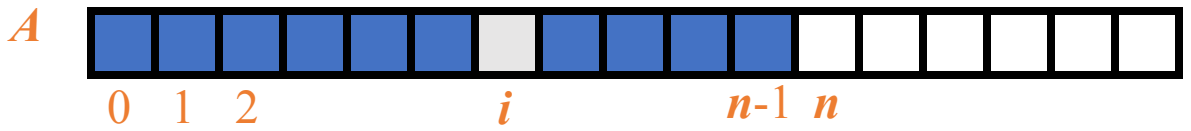
- Does downcasting always cause a ClassCastException?

```
A a2 = new C();
C c2 = (C) a2;
```

Arrays

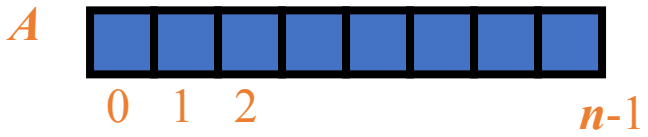
What is an Array?

- An array is a sequenced collection of homogenous variables (elements)
- Each element of an array has an index
- The length of an array is fixed and can not be changed
- Fast access – $O(1)$



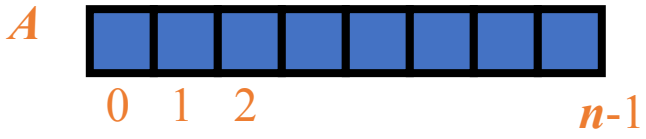
Let’s design an array that can change size!

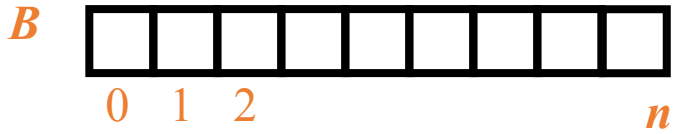
Imagine we have n items in our array



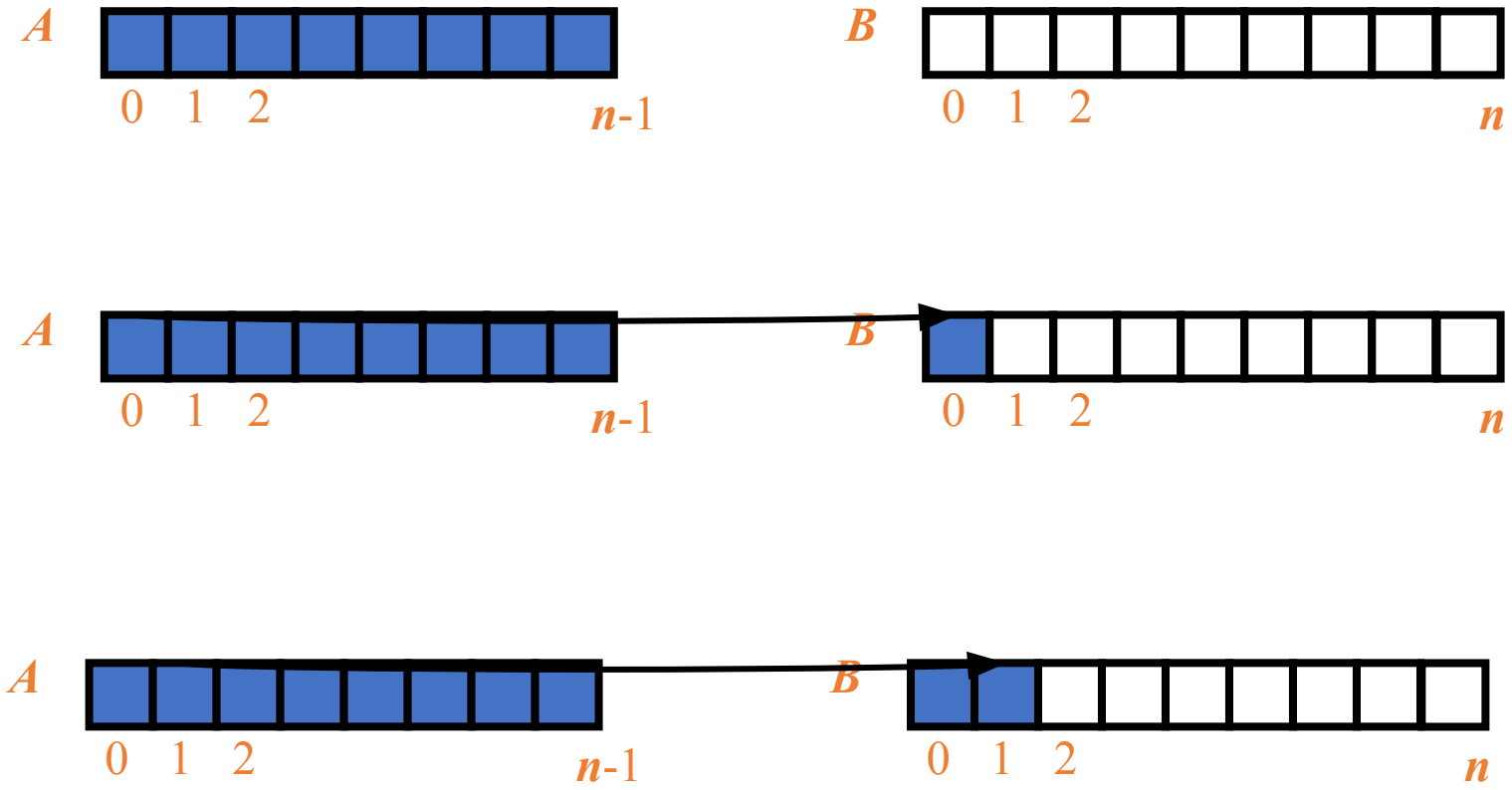
Say we want to add another item, are we stuck?

- No, make a new array and copy all the items over

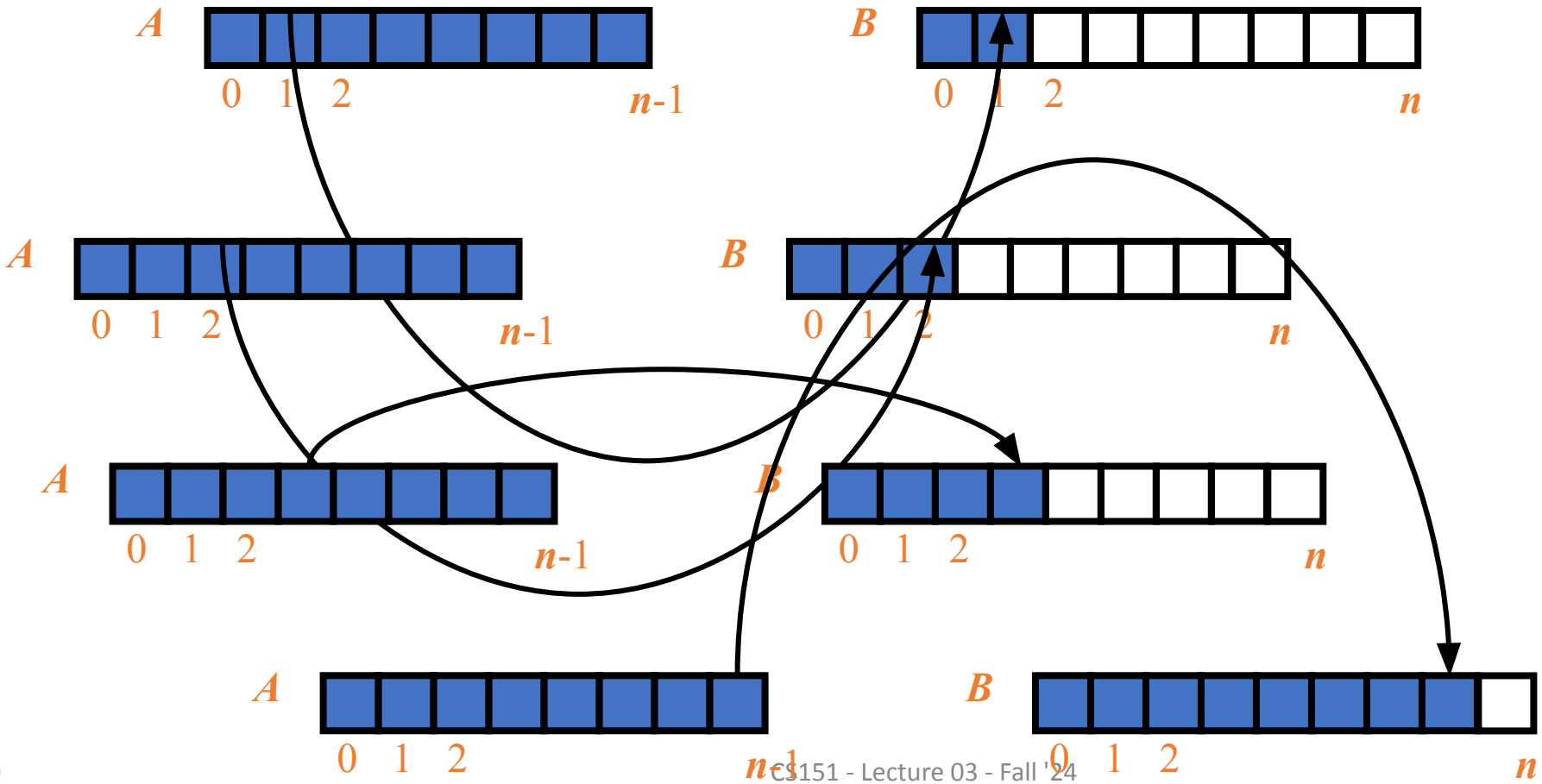




Array – Copying items over



Array – Copying items over



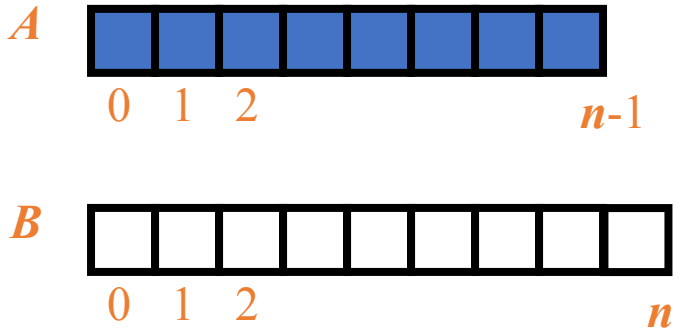
Array Copying

Computational complexity?

O(n)

How big should the new array be?

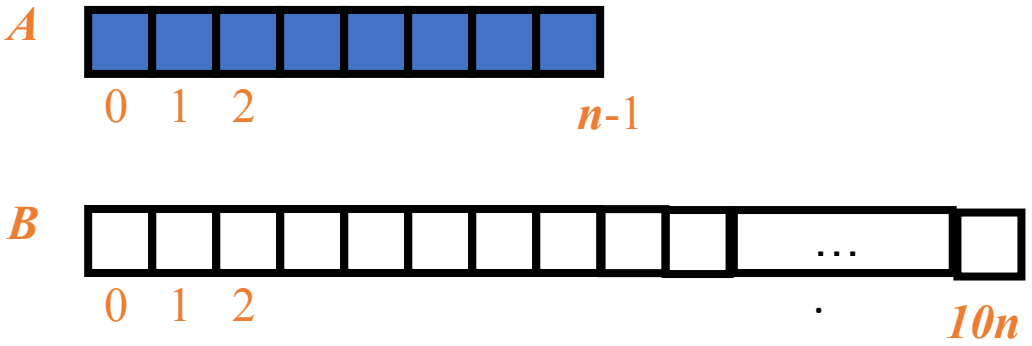
Just one more slot?



Pro: only use much space needed

Con: can lead to lots of copying over

10x the amount of slots?

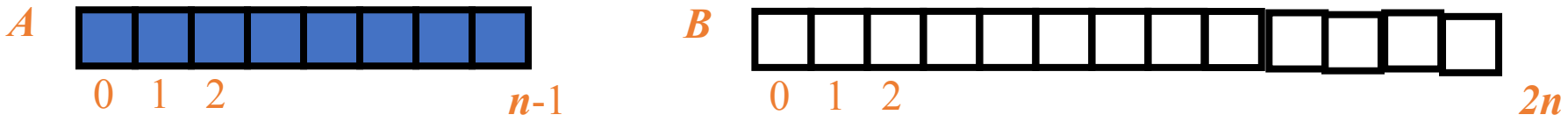


Pro: don’t need to copy lots of times

Con: lots of unused space

How big should the new array be?

- 2 times the length of the full array



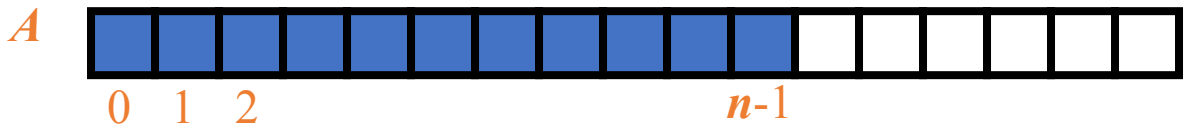
- Compromise between creating too much unnecessary space and having to expand the array too many times

- Runtime complexity?

Array Operations

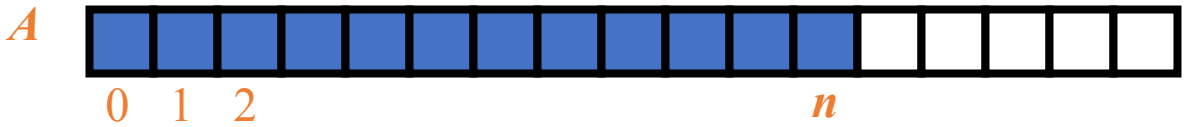
- Insertion
- Removal

Insertion



Where would be the easiest place to insert a new item?

The first open spot?

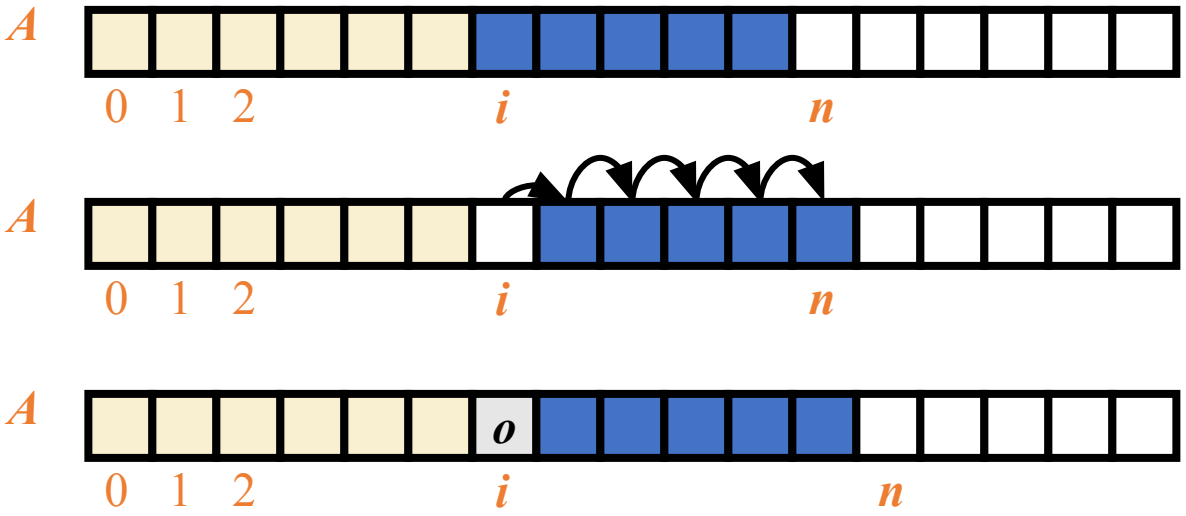


beginning of the array?

If we are going to search for that item a bunch

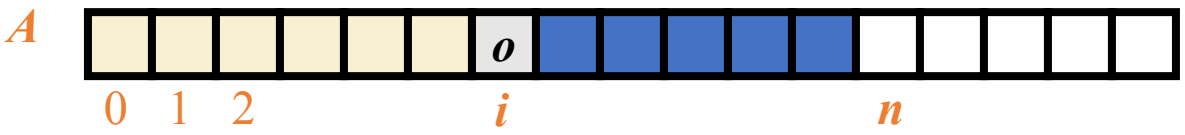
Insertion

- In an operation `insert(i, o)`, we make room for the new element `o` by shifting forward the elements `A[i], …, A[n - 1]`



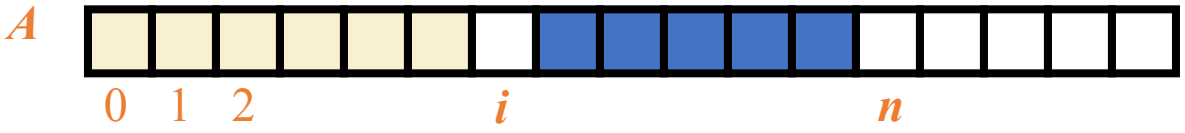
Removal

Say we want to remove the item at index `i`?

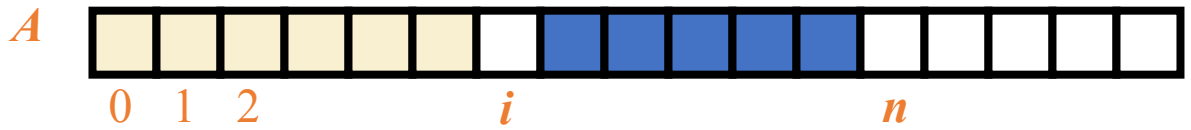


What's the simplest approach?

Just remove it, leaving an empty index



What is wrong with this setup?



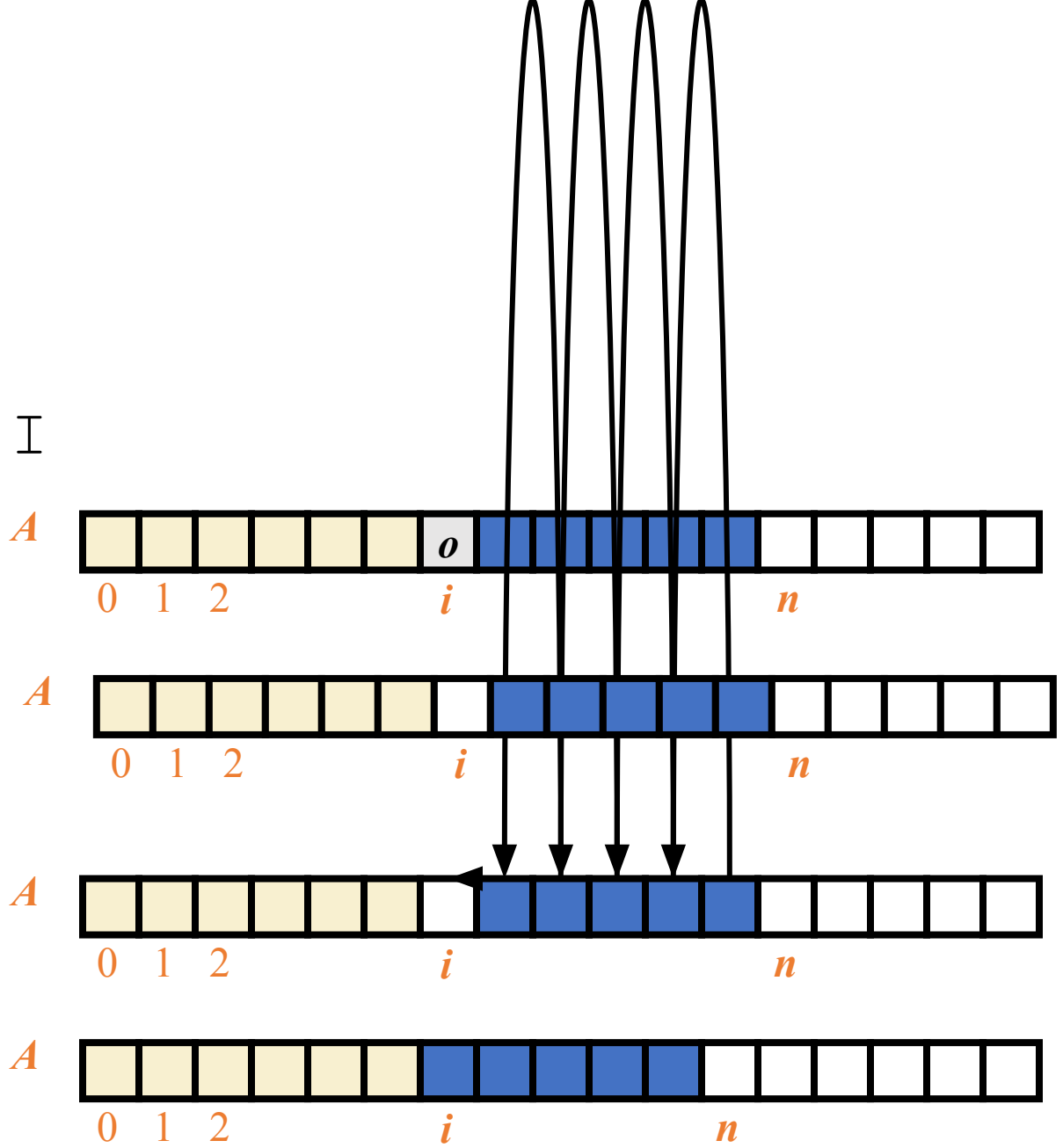
Why is having an empty slot in the middle of the array not ideal? What issues might arise?

- Makes inserting complicated
  - Where would we put a new item? At the end, or fill the spot?
- Makes looping through the array complicated
  - Need to check for null spots

Removing

In an operation `remove(i)`, we

- remove the element at location `I`
- then fill the hole by shifting backwards elements `A[i+1], …, A[n-1]`



Summary

Computational complexity of:

- Array lookup?
  - O(1)
- Array expansion?
  - O(n) or O(1) amortized
- Array insertion?
  - O(n)
- Array Removal?
  - O(n)

ExpandableArray

We just created an Expandable Array

- Dynamic size: grows and shrinks
- No empty slots between filled slots

- Supports:
  - Inserting in a specific location
  - Removing from a specific location

Summary

When would we want to use an array?

When would we might not want to?

Homework due Thursday