CS151 Intro to Data Structures

ArrayList, Generics

Announcements

- Midterm: Wednesday 25th (Wednesday after Fall break)
  - Post on piazza if you want this to be pushed back
  - Lets resolve this by end of this week
  - No one posted on Piazza so I guess its ok

- Goldengate:
  - Don’t work directly on goldengate
  - From goldengate, ssh into another machine
  - List of available machines:
    - https://cs.brynmawr.edu/~gtowell/crp.html

Announcements

- Piazza:
  - Asynchronous communication
- Gradescope:
  - Submit all assignments
  - Can request re-grade requests

- If not on either, come to my office right after class

Announcements

- Lab02:
  - Inheritance
  - ExpandableArray
  - Generics

- HW01 due Tuesday (09/19)
  - Will be released later tonight
  - Will be using your ExpandableArray from today's lab

Outline

- Object Oriented Programming
- ExpandableArray
- Generics

Homogeneous Type

- Array requires that the elements are of the same type

```
int[] nums = {1, 2, 3};
```

- A subclass object is an instance of its superclass

```
A[] abcs = new A[3];
abcs[0] = new A();
abcs[1] = new B();
abcs[2] = new C();
```

Object Casting

- Type conversion between super and subclasses – like the primitive types

- A superclass is a wider type

- A subclass is a narrower type

- Explicit super to sub cast is dangerous

```
class A {}
class B extends A {}
class C extends B {};
B b1 = (B) new A();
C c1 = (C) new B();
C c2 = (C) new A();
A a1 = new B();
B b2 = (B) a1;
```

Object Casting

- Type conversion between super and subclasses – like the primitive types
- A superclass is a wider type
- A subclass is a narrower type
- Explicit super to sub cast is dangerous

```
class A {}
class B extends A {}
class C extends B {};
B b1 = (B) new A();
C c1 = (C) new B();
C c2 = (C) new A();
A a1 = new B();
B b2 = (B) a1;
```

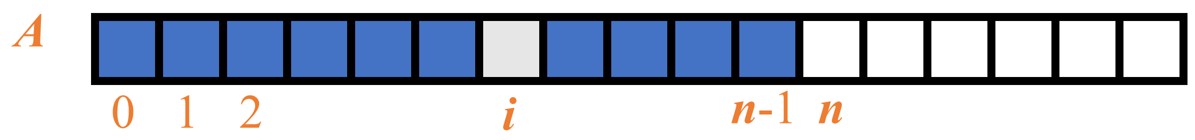
Super to Sub Cast

- Explicit super to sub cast is dangerous
- First use instanceof to make sure its possible

Arrays

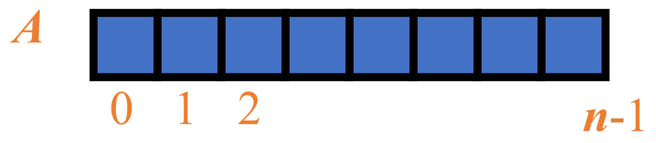
What is an Array?

- 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 – $O(1)$



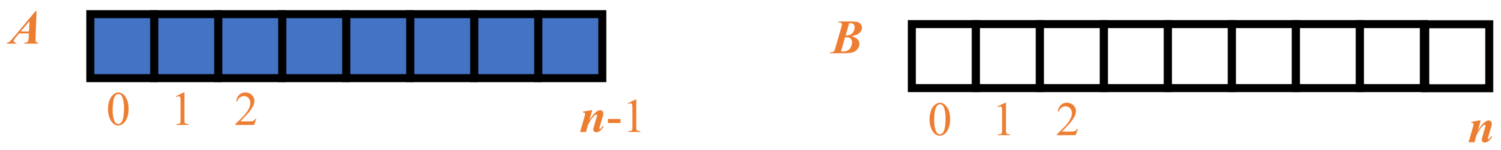
Array

Imagine we have n items in our array

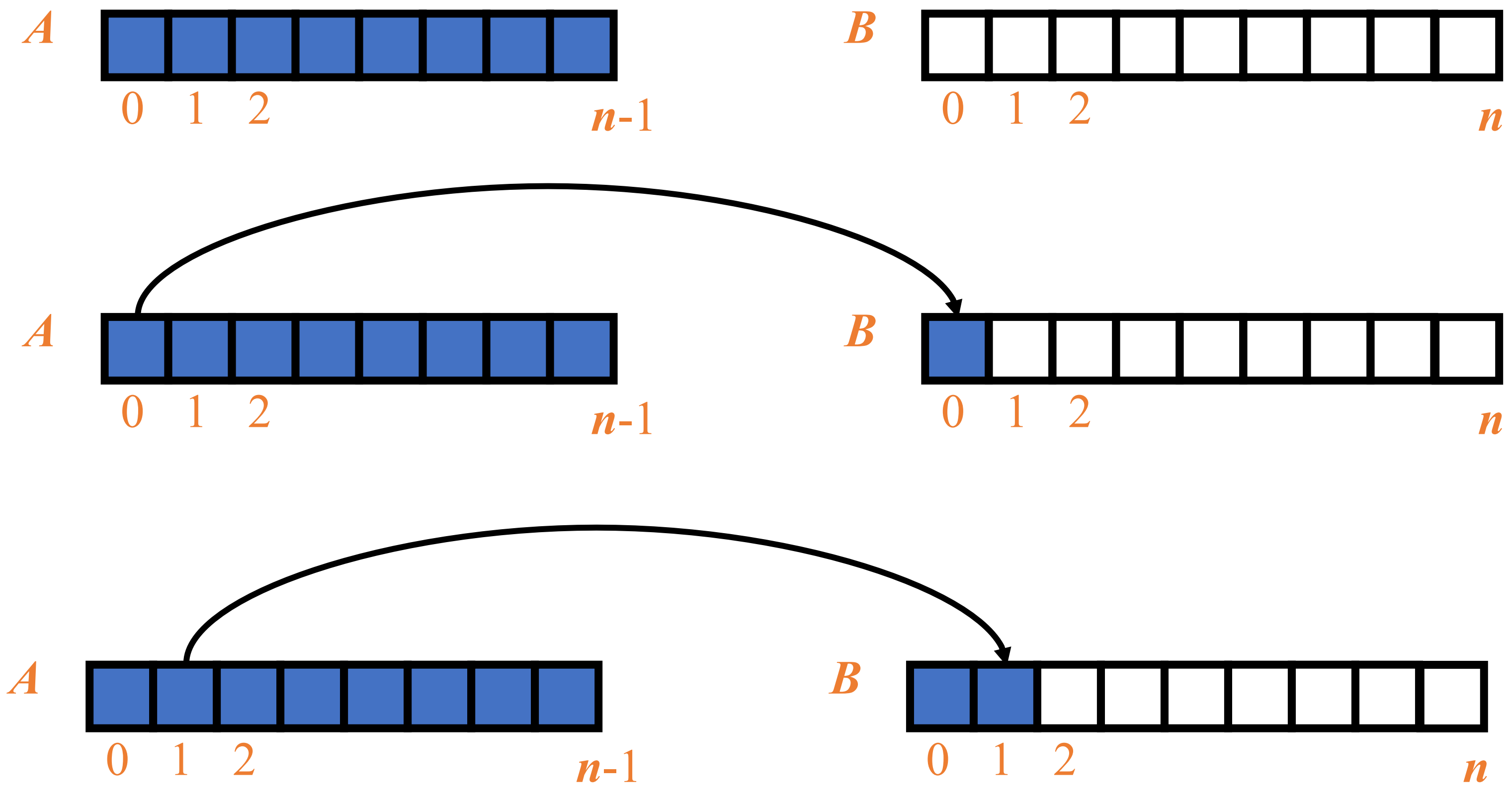


Say we want to add another item, are we stuck?

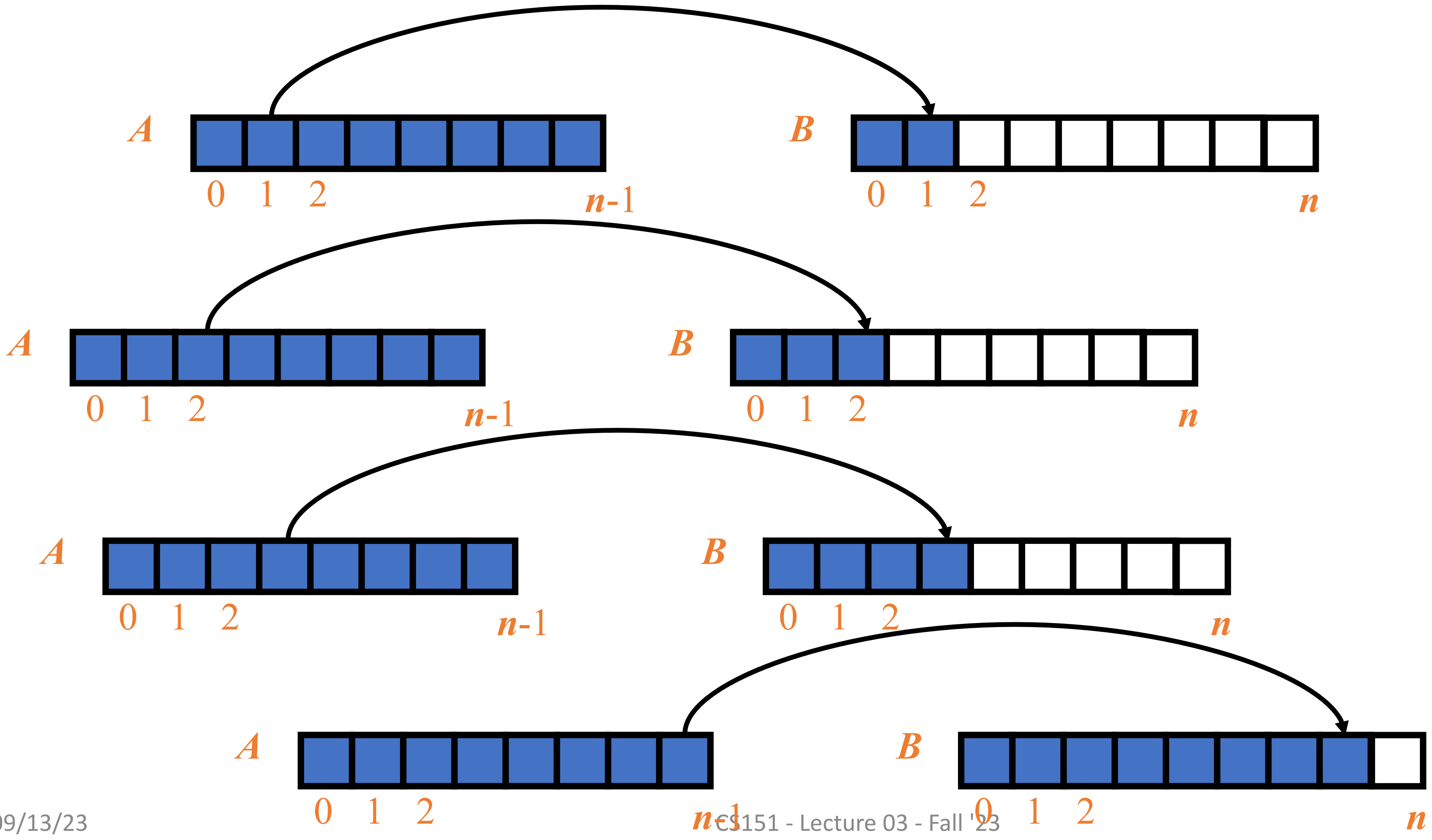
- No, make a new array and copy all the items over



Array – Copying items over

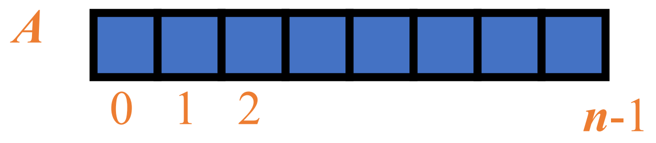


Array – Copying items over



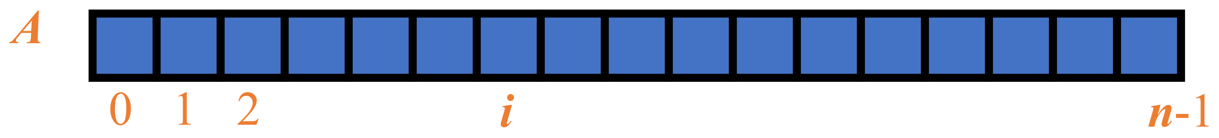
Array – Copy items over

Imagine we have n items in our array



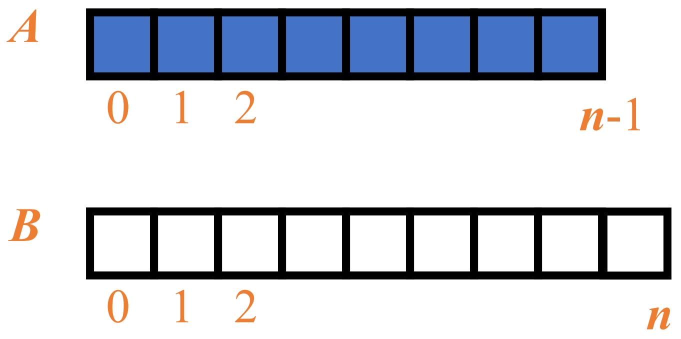
Say we want to add another item, are we stuck?

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



How big should the new array be?

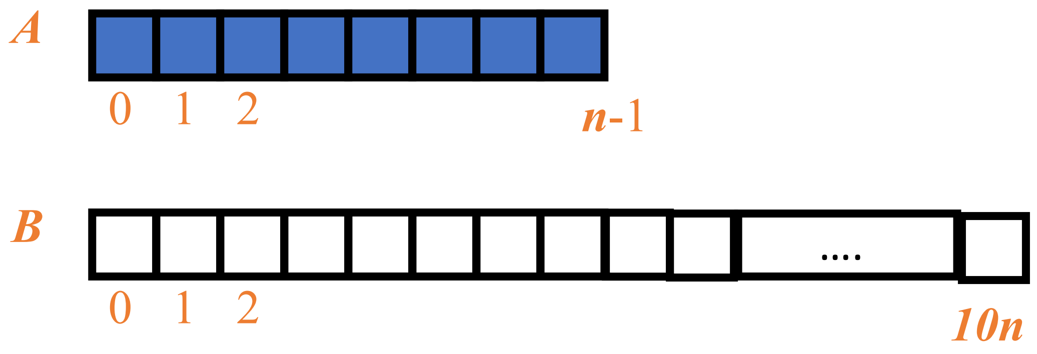
Just one more slot



Pro: only use much space needed

Con: can lead to lots of copying over

10x the amount of slots

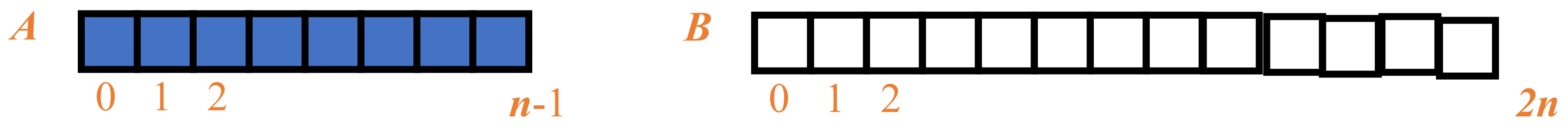


Pro: don't need to copy lots of times

Con: lots of unused space

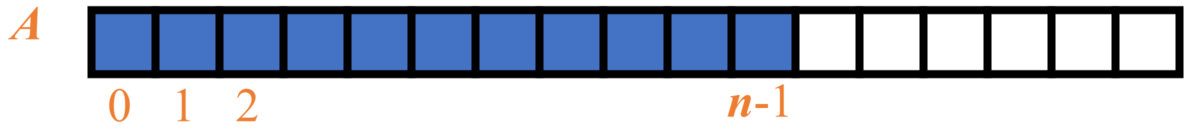
How big should the new array be?

- 2 times the length of the full array



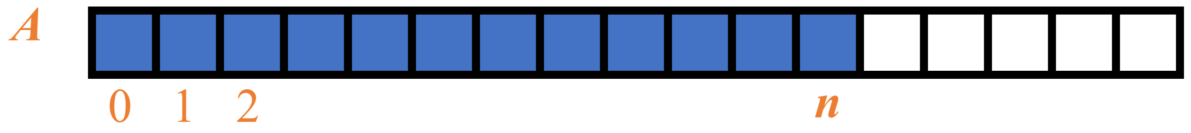
- Compromise between creating too much unnecessary space and having to expand the array too many times

Insertion



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

The first open spot

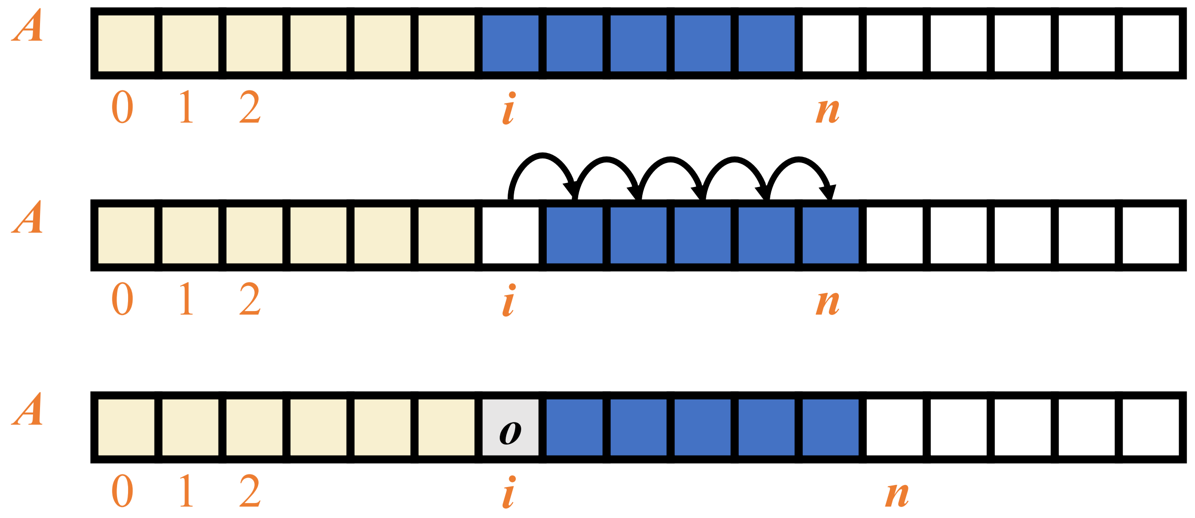


Why might we want to insert an item in the beginning of the array?

If we are going to search for that item a bunch

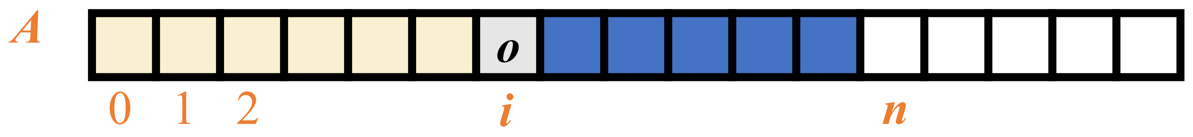
Insertion

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



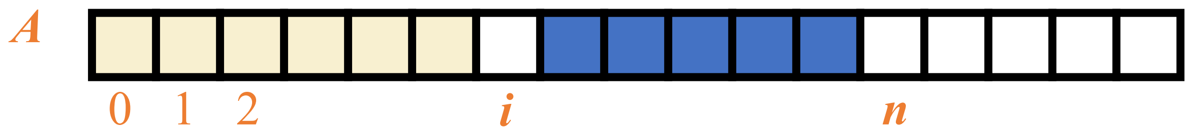
Removing

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



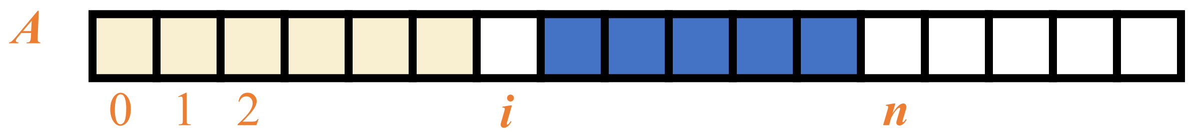
What's the simplest approach?

Just remove it!



That was easy!

What is wrong with this setup?



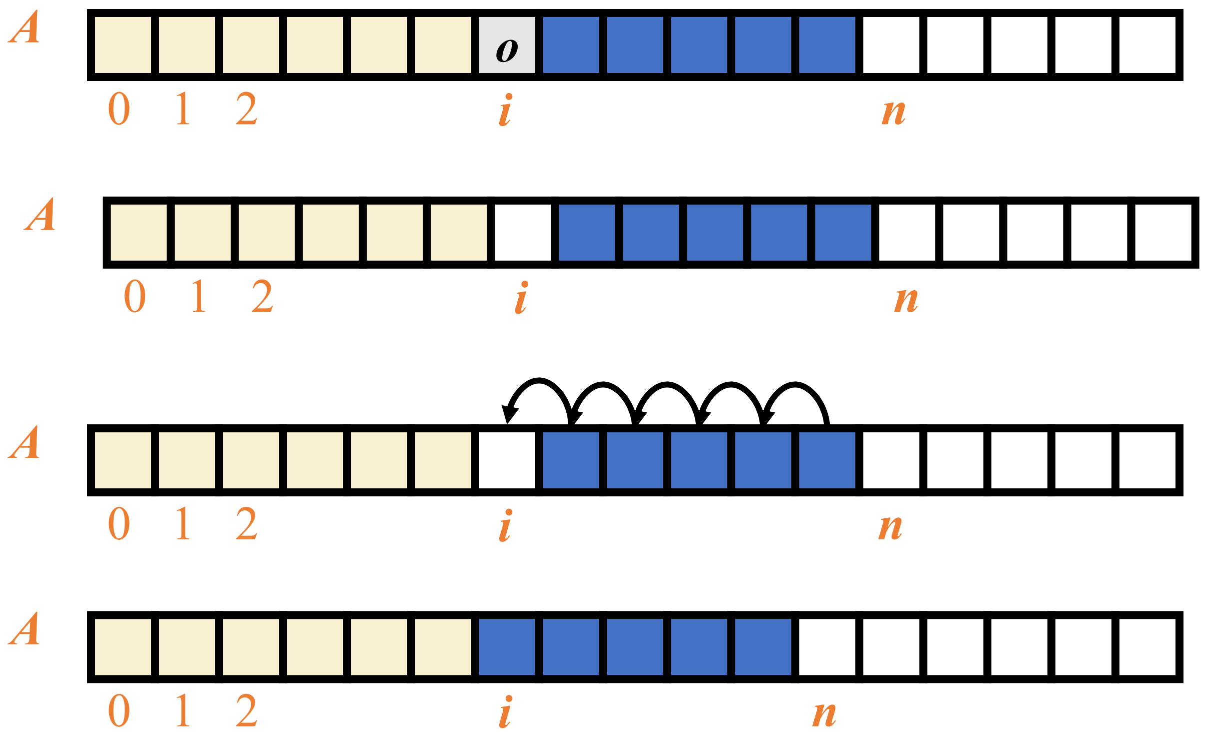
Why is having an empty slot in the middle of the array not ideal? What issues might arise?

- 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]`



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

ExpandableArray

In our ExpandableArray class, how should we store our data?

- Use an array!

Write a class that supports an self-expanding array

```
public class ExpandingArray {
  private int[] array;
  public ExpandingArray(int size) {
    this.array = new int[size];
  }
  public void insert(int item) { … }
  public int getItem(int index) { … }
  public int indexOf(int item) { … }
}
ExpandingArray obj = new ExpandingArray(10);
```

ExpandableArray

In our ExpandableArray class, how should we store our data?

- Use an array!

But we want our ExpandableArray to work with any data type?

- It should be able to store ints, doubles, Strings, Students

Generics

Generics

- A way to write classes or methods that can operate on a variety of data types without being locked into specific types at the time of definition
- Write definitions with type parameters
- The types are instantiated (locked down) when objects are created

Self-expanding Array as Generic Class

```
public class ExpandingArray<T> {
  private T[] array;
  public ExpandingArray(int size) {
    …
  }
  public void insert(T item) { … }
  public T getItem(int index) { … }
  public int indexOf(T item) { … }
}
ExpandingArray<String> obj1 = new ExpandingArray<String>(10);
ExpandingArray<Integer> obj1 = new ExpandingArray<Integer>(10);
```

Generic Class

```
public class Pair<A, B> {
  private A first; private B second;
  public Pair(A first, B second) {
    this.first = first; this.second = second;
  }
  public A getFirst() {return first;}
  public B getSecond() {return second;}
  public String toString() {//??}
}
Pair<String, Double> deposit = new Pair<>("USD", 500.00);
```

Generics Restrictions

No instantiation with primitive types

Can not declare static instance variables of a parameterized type

Can not create arrays of parameterized types

- `private T[] array;` is not valid

- Casting to the rescue!
  - `T[] array = (T[]) new Object[10];`