12/5/22

6.4: Array Algorithms part 2

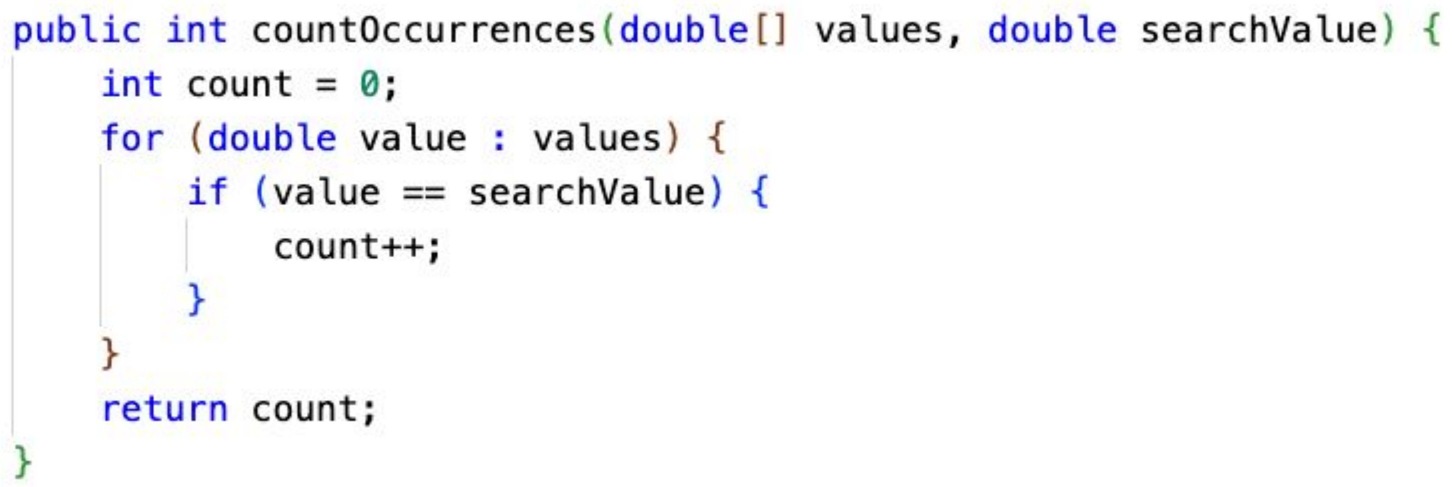
Common Array Algorithms (**bold** ones today)

Here are some common algorithms that you should be familiar with for the AP CS A exam:

- Determine the minimum or maximum value in an array
- Compute a sum, average, or **mode** of array elements
- **Search for a particular element in the array**
- **Determine if at least one element has a particular property**
- **Determine if all elements have a particular property**
- **Access all consecutive pairs of elements**
- **Determine the presence or absence of duplicate elements**
- **Determine the number of elements meeting specific criteria**
- **Shift or rotate elements left or right**
- **Reverse the order of the elements**

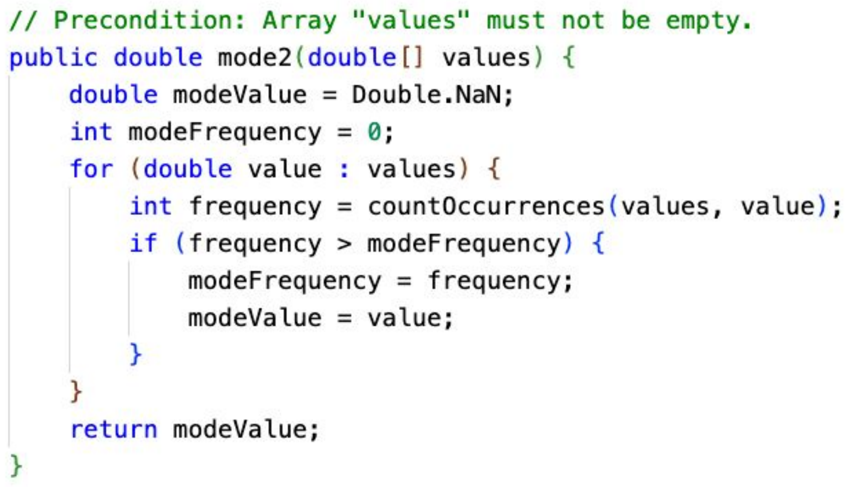
Hmm, we didn't get very far on Friday... going to have to have an efficient algorithm today to cover all these...

Determine number of elements meeting specific criteria

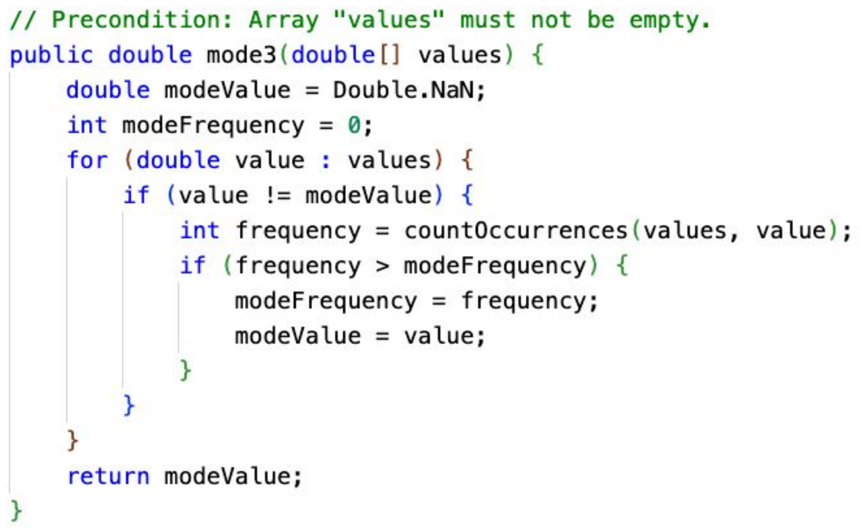


```java
public int countOccurrences(double[] values, double searchValue) {
    int count = 0;
    for (double value : values) {
        if (value == searchValue) {
            count++;
        }
    }
    return count;
}
```

Mode of unsorted array, using helper method



```java
// Precondition: Array "values" must not be empty.
public double mode2(double[] values) {
    double modeValue = Double.NaN;
    int modeFrequency = 0;
    for (double value : values) {
        int frequency = countOccurrences(values, value);
        if (frequency > modeFrequency) {
            modeFrequency = frequency;
            modeValue = value;
        }
    }
    return modeValue;
}
```



```java
// Precondition: Array "values" must not be empty.
public double mode3(double[] values) {
    double modeValue = Double.NaN;
    int modeFrequency = 0;
    for (double value : values) {
        if (value != modeValue) {
            int frequency = countOccurrences(values, value);
            if (frequency > modeFrequency) {
                modeFrequency = frequency;
                modeValue = value;
            }
        }
    }
    return modeValue;
}
```

Mode of unsorted array, using helper method

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
17	17	9	9	9	9	1	17	19	3	3	5

countOccurrences(values, values[i])

3	skip	4	skip	skip	skip	1	3	1	2	skip	1

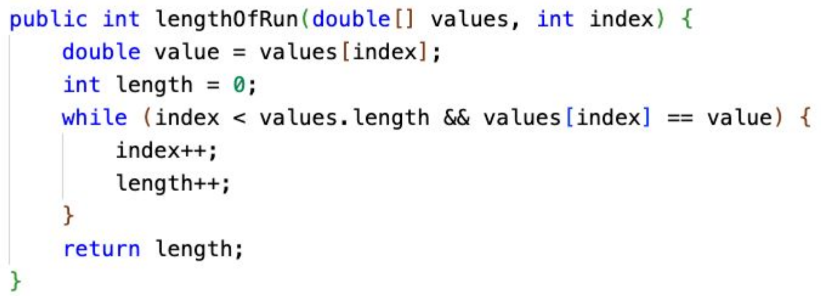
modeValue

```
17          9
```

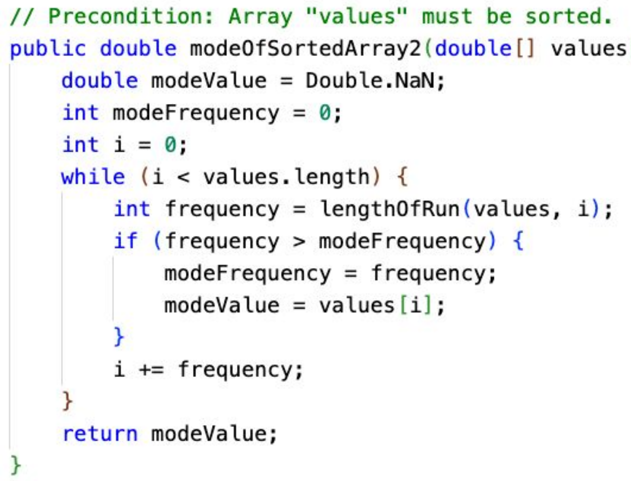
modeFrequency

```
3           4
```

Mode of sorted array, with helper method



```java
public int lengthOfRun(double[] values, int index) {
    double value = values[index];
    int length = 0;
    while (index < values.length && values[index] == value) {
        index++;
        length++;
    }
    return length;
}
```

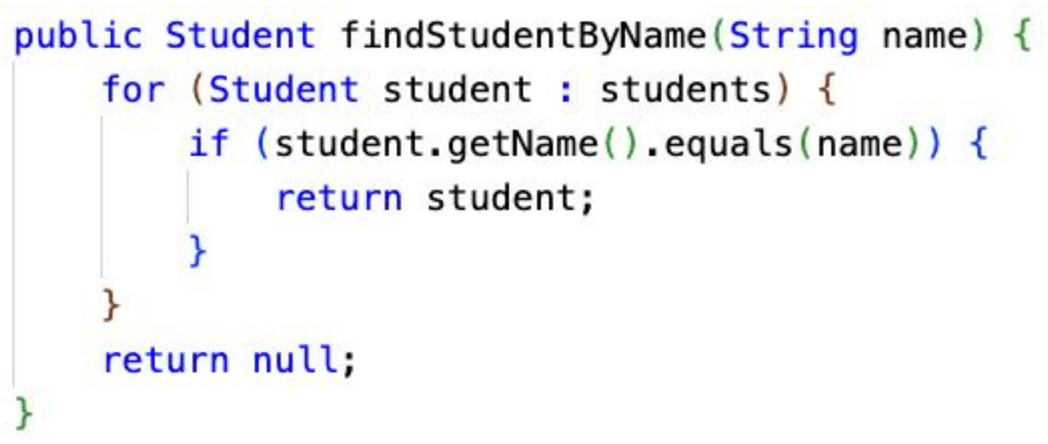


```java
// Precondition: Array "values" must be sorted.
public double modeOfSortedArray2(double[] values) {
    double modeValue = Double.NaN;
    int modeFrequency = 0;
    int i = 0;
    while (i < values.length) {
        int frequency = lengthOfRun(values, i);
        if (frequency > modeFrequency) {
            modeFrequency = frequency;
            modeValue = values[i];
        }
        i += frequency;
    }
    return modeValue;
}
```

1	1	2	2	2	3	3	3	3	4	4	5

Search for a particular element in the array

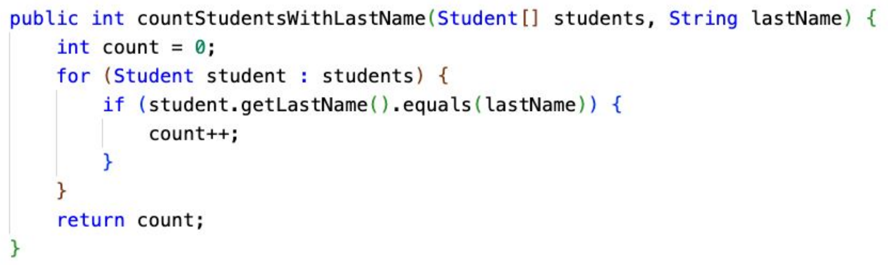
This is known as **linear search**, the simplest (and least efficient) of search algorithms. We'll learn others later.



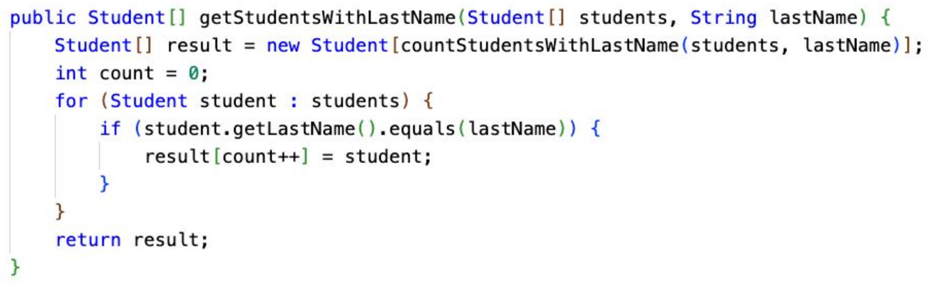
```java
public Student findStudentByName(String name) {
    for (Student student : students) {
        if (student.getName().equals(name)) {
            return student;
        }
    }
    return null;
}
```

Filter an array for all matching elements

What if you want to find all matching elements? One way is to **filter** the array into a new array.



```
public int countStudentsWithLastName(Student[] students, String lastName) {
    int count = 0;
    for (Student student : students) {
        if (student.getLastName().equals(lastName)) {
            count++;
        }
    }
    return count;
}
```



```
public Student[] getStudentsWithLastName(Student[] students, String lastName) {
    Student[] result = new Student[countStudentsWithLastName(students, lastName)];
    int count = 0;
    for (Student student : students) {
        if (student.getLastName().equals(lastName)) {
            result[count++] = student;
        }
    }
    return result;
}
```

When returning an array that is a subset of the original array, you have to decide how big to make the result array.

Here, we do it by doing another pass through the array just to count how big the result array should be, using a helper method.

Another option would be the "growable array" pattern... In Unit 7, we'll cover ArrayList which could be used for this purpose.

Search for all matching elements

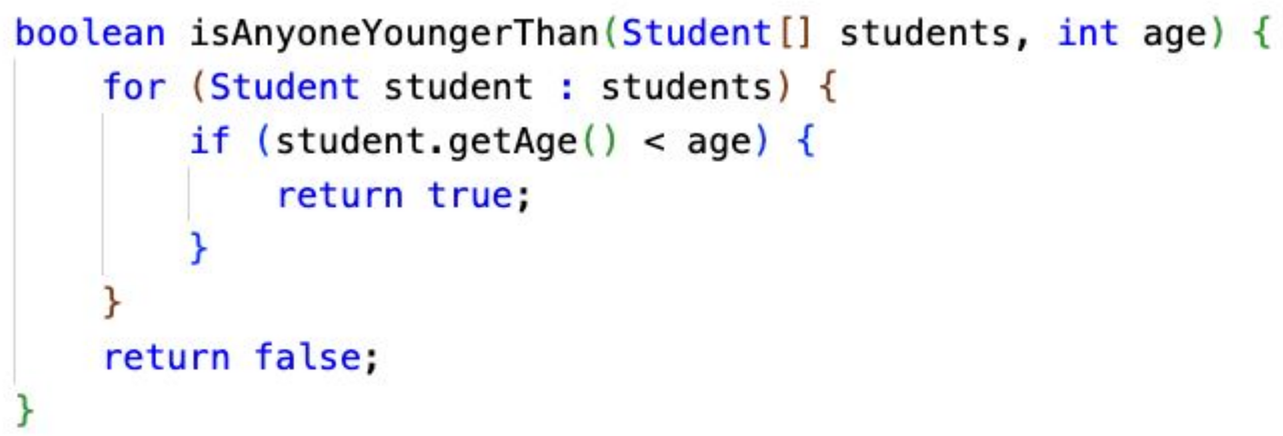
Another way to find all matching elements is to build your own indexOf with startIndex parameter.

```
public int indexOfStudentWithLastName(Student[] students, String lastName, int startIndex) {
    for (int i=startIndex, n=students.length; i<n; i++) {
        if (students[i].getLastName().equals(lastName)) {
            return i;
        }
    }
    return -1;
}
```

```
int index = 0;
while ((index = indexOfStudentWithLastName(students, "Smith", index)) != -1) {
    System.out.println(students[index]);
    index++;
}
```

Determine if at least one element has a particular property

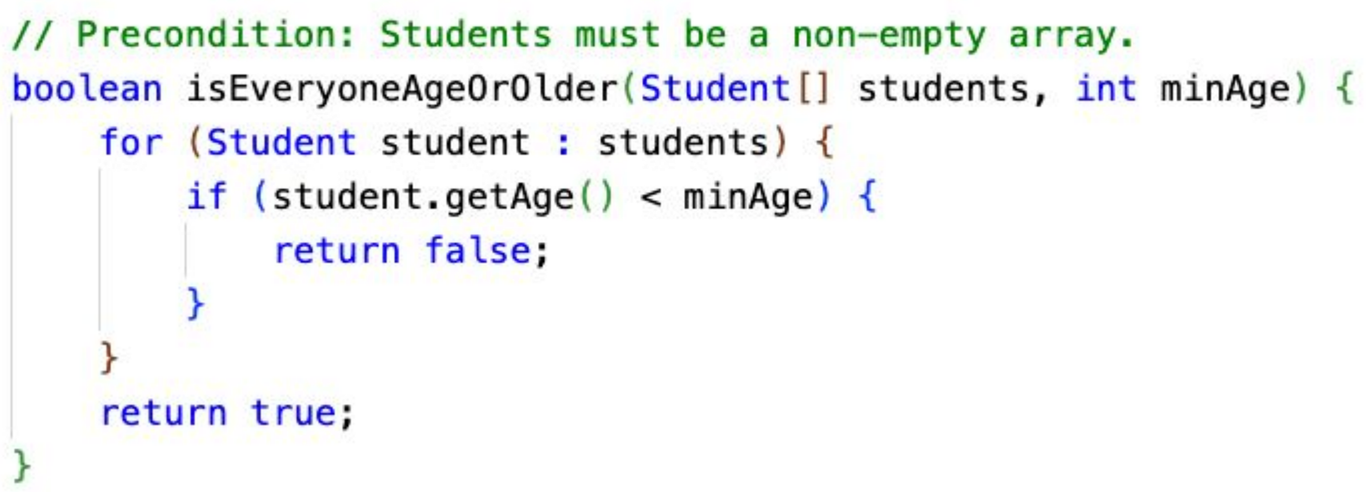
The output of this type of algorithm is a boolean. As soon as you find the first element with the desired property, you can return true.



```
boolean isAnyoneYoungerThan(Student[] students, int age) {
    for (Student student : students) {
        if (student.getAge() < age) {
            return true;
        }
    }
    return false;
}
```

Determine if all elements have a particular property

This is essentially the same, except we're trying to ensure that ALL elements have some property. As soon as we find an element that doesn't, we return false.

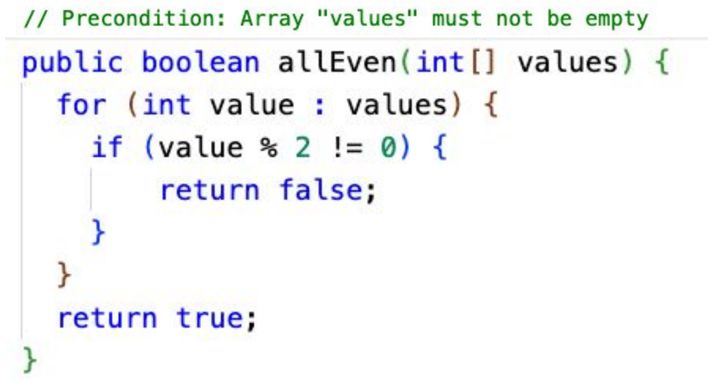


```java
// Precondition: Students must be a non-empty array.
boolean isEveryoneAgeOrOlder(Student[] students, int minAge) {
    for (Student student : students) {
        if (student.getAge() < minAge) {
            return false;
        }
    }
    return true;
}
```

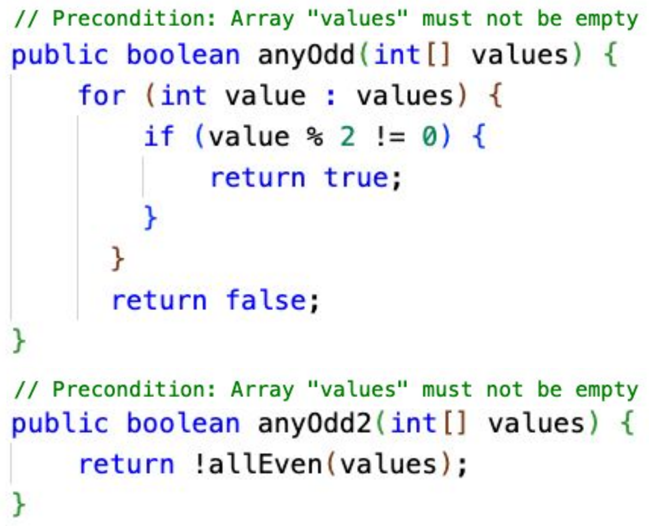
Universal / Existential Quantifiers

If all of the numbers all even, then there are no odd numbers.

If there are any odd numbers, then not all of the numbers are even.

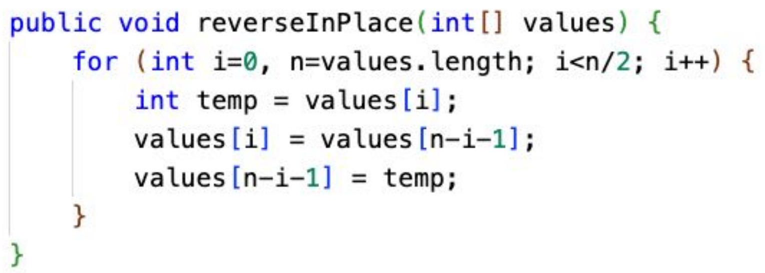


```java
// Precondition: Array "values" must not be empty
public boolean allEven(int[] values) {
  for (int value : values) {
    if (value % 2 != 0) {
        return false;
    }
  }
  return true;
}
```

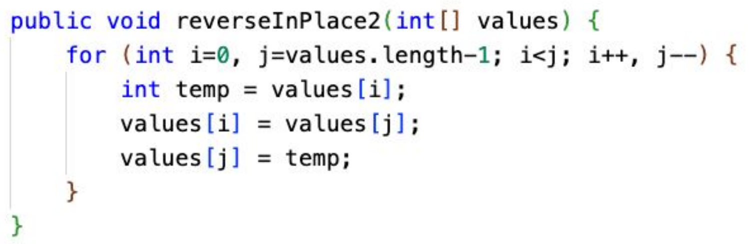


```java
// Precondition: Array "values" must not be empty
public boolean anyOdd(int[] values) {
    for (int value : values) {
        if (value % 2 != 0) {
            return true;
        }
    }
    return false;
}
// Precondition: Array "values" must not be empty
public boolean anyOdd2(int[] values) {
    return !allEven(values);
}
```

Reverse an array (in place)



```java
public void reverseInPlace(int[] values) {
    for (int i=0, n=values.length; i<n/2; i++) {
        int temp = values[i];
        values[i] = values[n-i-1];
        values[n-i-1] = temp;
    }
}
```



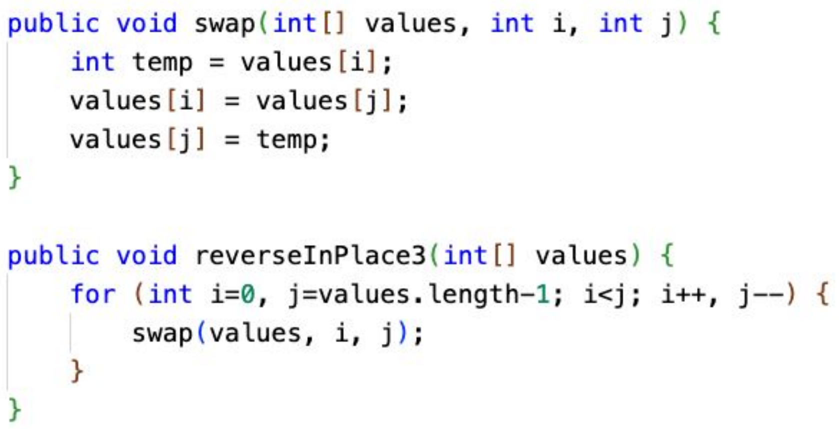
```java
public void reverseInPlace2(int[] values) {
    for (int i=0, j=values.length-1; i<j; i++, j--) {
        int temp = values[i];
        values[i] = values[j];
        values[j] = temp;
    }
}
```

The top implementation is fine, but the bottom one uses i and j instead of just i.

I find it easier to reason about what this algorithm is doing by having two index counters, "racing" toward each other from each end of the array.

Computers have many **registers** for storage of frequently used variables, so there is really no additional cost to having two variables instead of one. It can be even faster, since less arithmetic is performed.

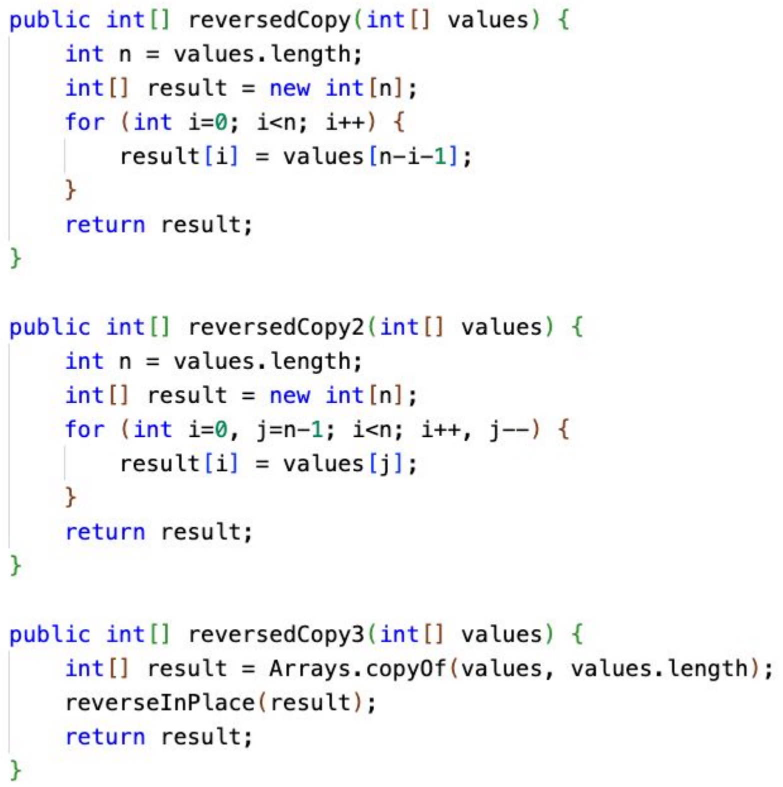
Reverse an array (in place)



```java
public void swap(int[] values, int i, int j) {
    int temp = values[i];
    values[i] = values[j];
    values[j] = temp;
}

public void reverseInPlace3(int[] values) {
    for (int i=0, j=values.length-1; i<j; i++, j--) {
        swap(values, i, j);
    }
}
```

A helper method to swap array elements makes the code even easier to understand.

Return a reversed copy of an array



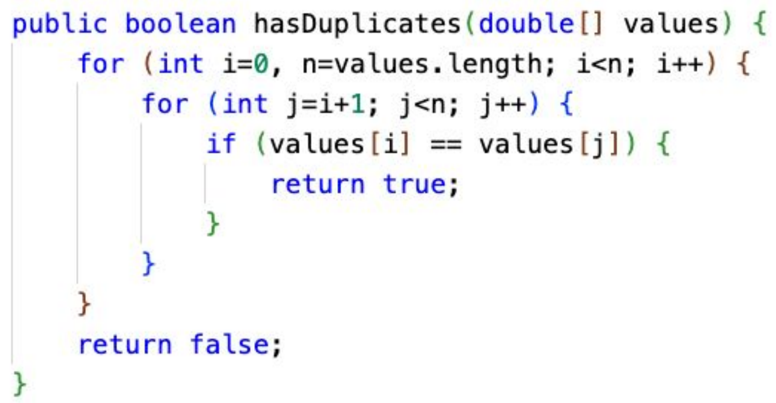
```java
public int[] reversedCopy(int[] values) {
    int n = values.length;
    int[] result = new int[n];
    for (int i=0; i<n; i++) {
        result[i] = values[n-i-1];
    }
    return result;
}

public int[] reversedCopy2(int[] values) {
    int n = values.length;
    int[] result = new int[n];
    for (int i=0, j=n-1; i<n; i++, j--) {
        result[i] = values[j];
    }
    return result;
}

public int[] reversedCopy3(int[] values) {
    int[] result = Arrays.copyOf(values, values.length);
    reverseInPlace(result);
    return result;
}
```

If you're returning a copy of an array in reverse order, you don't have to do any swapping.

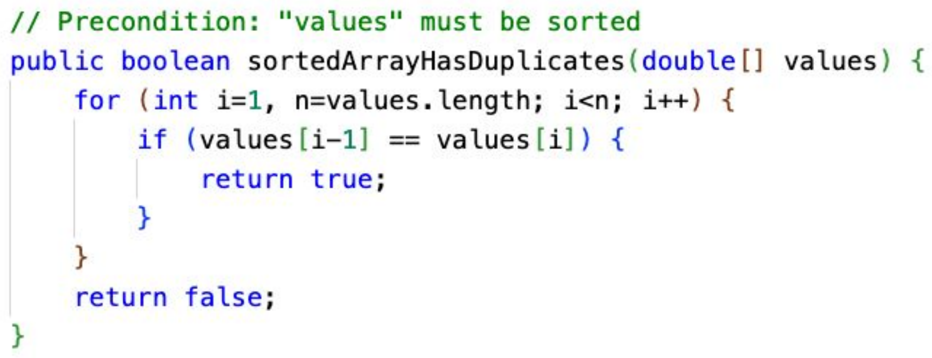
It could still be helpful to use two counters instead of one.

You also could just not write the code at all... and leverage the reverse-in-place algorithm we just wrote. It will take a little more CPU time, though, since the array will first be copied, then reversed.

Check for presence of duplicate elements



```java
public boolean hasDuplicates(double[] values) {
    for (int i=0, n=values.length; i<n; i++) {
        for (int j=i+1; j<n; j++) {
            if (values[i] == values[j]) {
                return true;
            }
        }
    }
    return false;
}
```

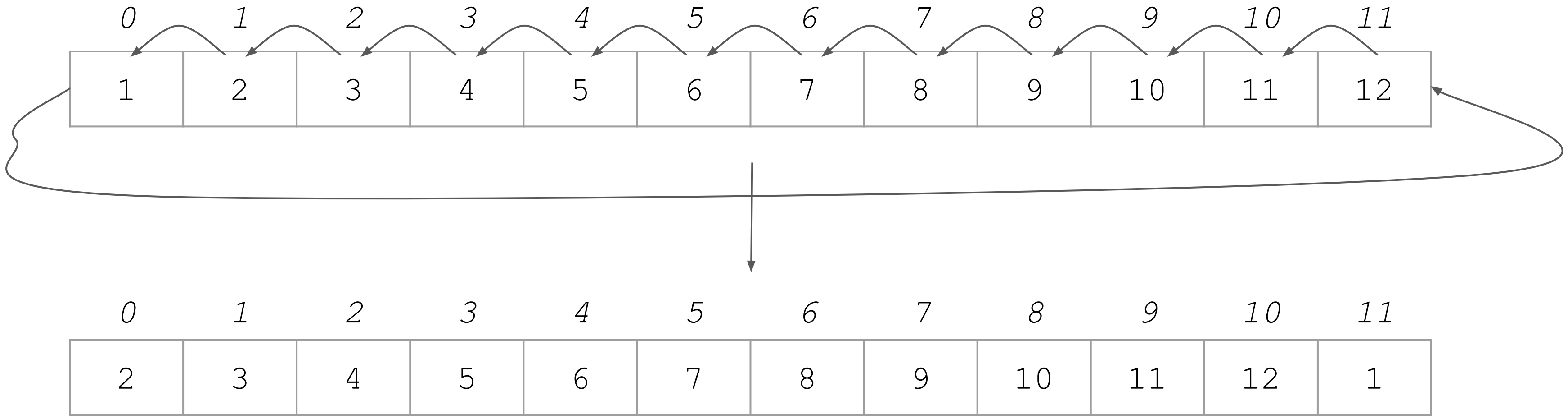


```java
// Precondition: "values" must be sorted
public boolean sortedArrayHasDuplicates(double[] values) {
    for (int i=1, n=values.length; i<n; i++) {
        if (values[i-1] == values[i]) {
            return true;
        }
    }
    return false;
}
```

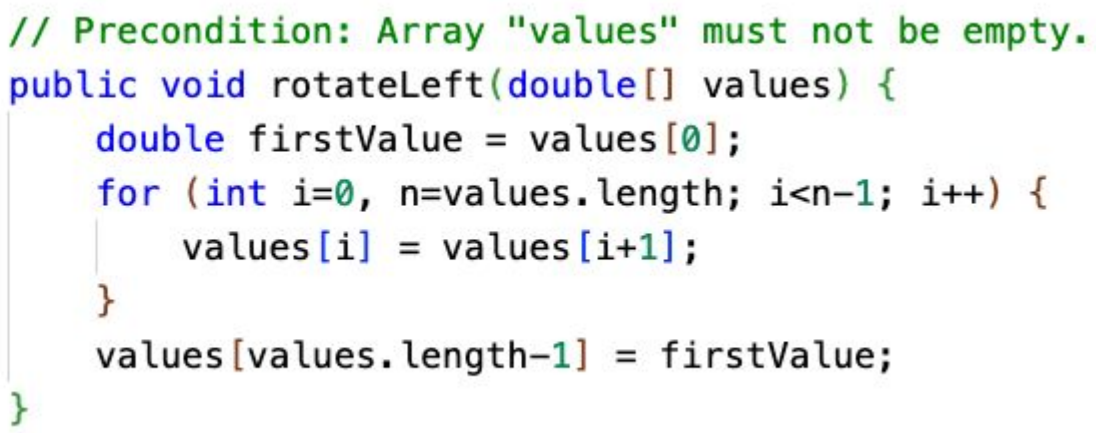
Shift or rotate an array

Here, we rotate an array of numbers to the left by 1 position.

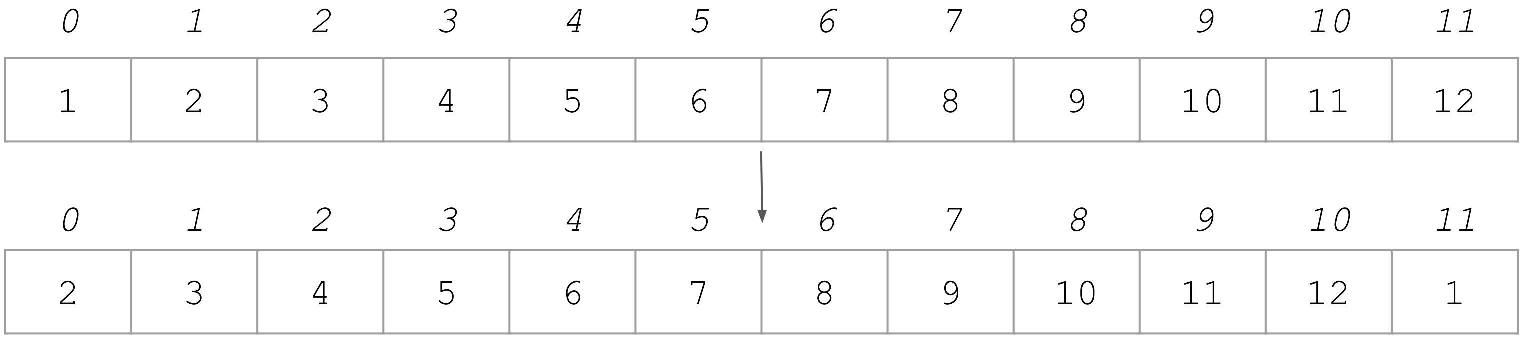
a[i] = a[i+1] for all i. The first element gets moved to the last position.



Rotate array to the left, in place



```java
// Precondition: Array "values" must not be empty.
public void rotateLeft(double[] values) {
    double firstValue = values[0];
    for (int i=0, n=values.length; i<n-1; i++) {
        values[i] = values[i+1];
    }
    values[values.length-1] = firstValue;
}
```

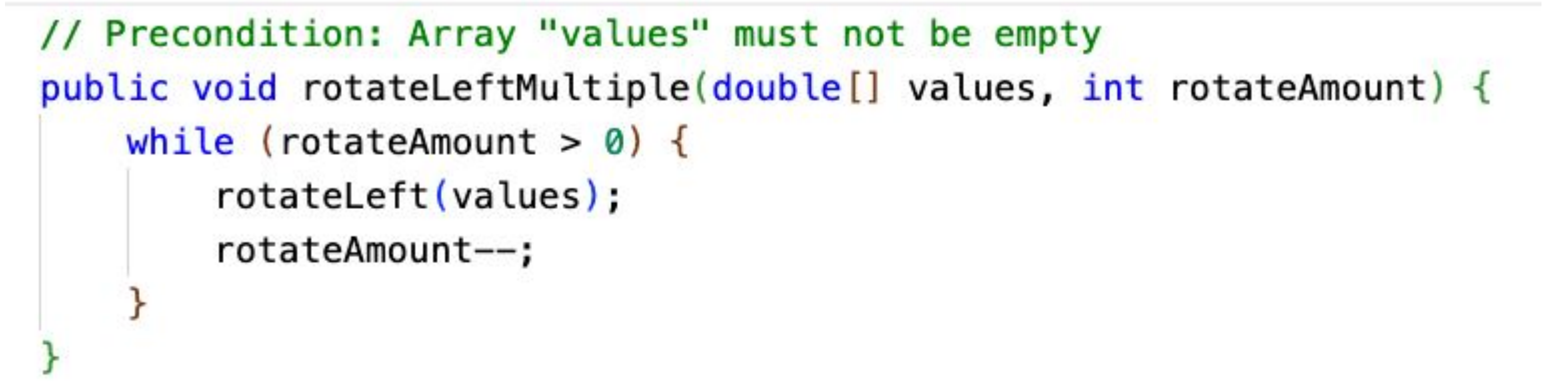


0	1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8	9	10	11	12

0	1	2	3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	9	10	11	12	1

Rotate array to the left multiple positions, in place

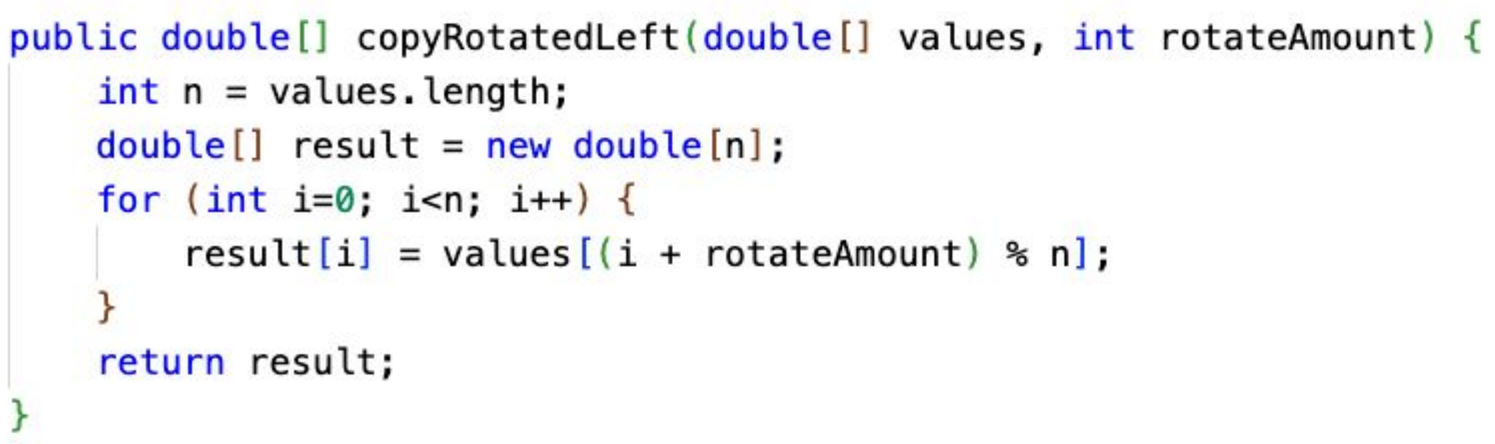
There are other ways... but this is a quick and dirty way to do it.



```java
// Precondition: Array "values" must not be empty
public void rotateLeftMultiple(double[] values, int rotateAmount) {
    while (rotateAmount > 0) {
        rotateLeft(values);
        rotateAmount--;
    }
}
```

Rotate array to the left multiple positions, returning copy

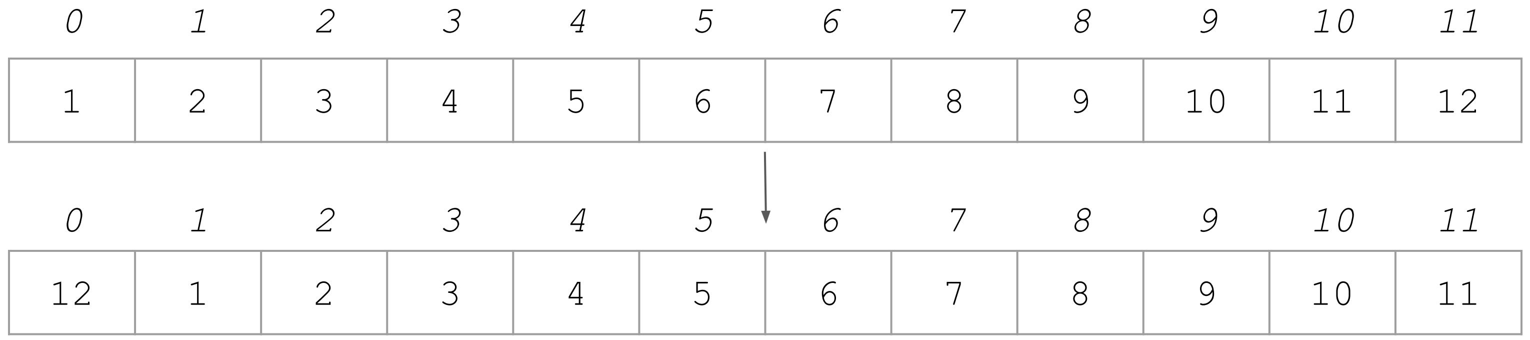
Here's another instance where the % operator is very useful.



```java
public double[] copyRotatedLeft(double[] values, int rotateAmount) {
    int n = values.length;
    double[] result = new double[n];
    for (int i=0; i<n; i++) {
        result[i] = values[(i + rotateAmount) % n];
    }
    return result;
}
```

Rotate array to the right, in place

```java
// Precondition: Array "values" must not be empty.
public void rotateRight(double[] values) {
    int n = values.length;
    double lastValue = values[n-1];
    for (int i=n-1; i>0; i--) {
        values[i] = values[i-1];
    }
    values[0] = lastValue;
}
```

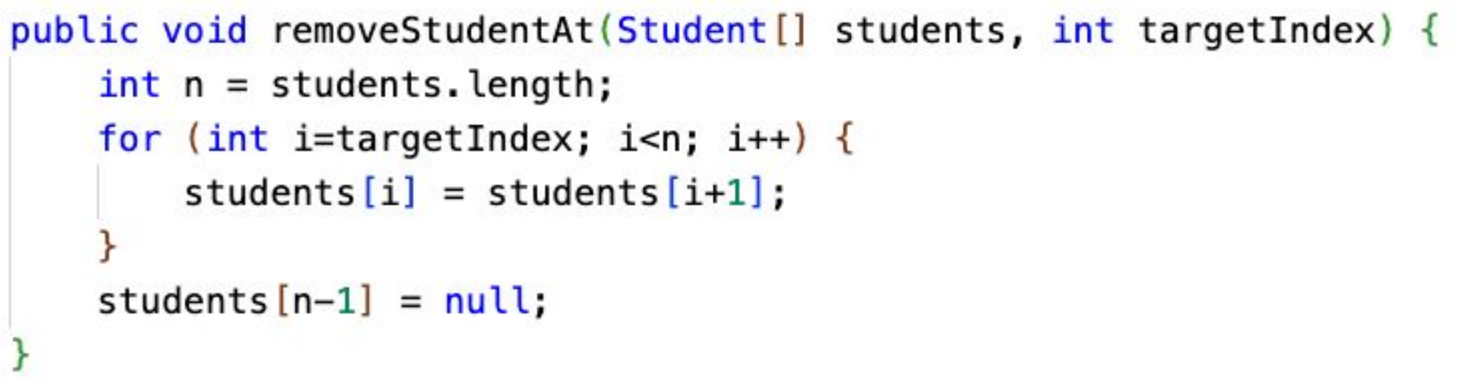


0	1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8	9	10	11	12

0	1	2	3	4	5	6	7	8	9	10	11
12	1	2	3	4	5	6	7	8	9	10	11

Remove element at specific index from an array

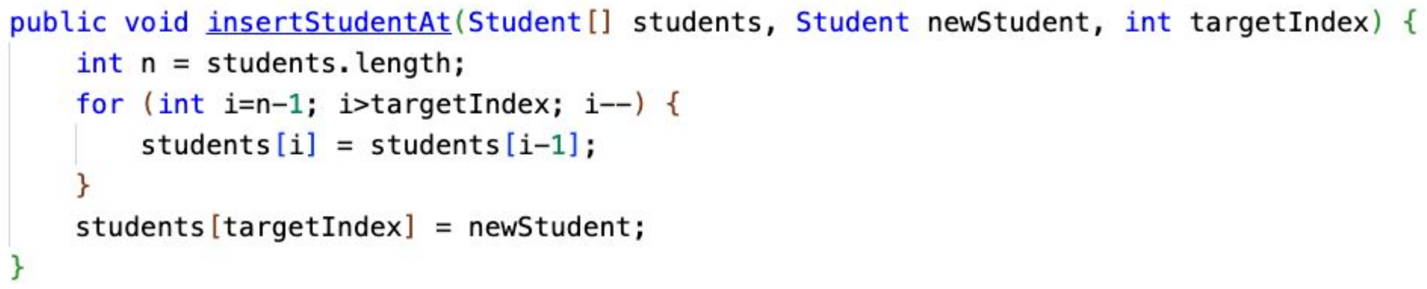
This is essentially like rotating left, but starting at a particular spot. Here, we fill in the end with **null**.



```
public void removeStudentAt(Student[] students, int targetIndex) {
    int n = students.length;
    for (int i=targetIndex; i<n; i++) {
        students[i] = students[i+1];
    }
    students[n-1] = null;
}
```

Insert element at specific index in array

This is very similar to rotating the array right, but starting at a specific index and not "wrapping."



```java
public void insertStudentAt(Student[] students, Student newStudent, int targetIndex) {
    int n = students.length;
    for (int i=n-1; i>targetIndex; i--) {
        students[i] = students[i-1];
    }
    students[targetIndex] = newStudent;
}
```

The last element will be lost, so you'd have to make sure there is empty space at the end!

Practice!

Repl.it **SieveOfEratosthenes**

Repl.it **ArrayAlgorithms** is just the code in this slide deck, for your perusal.

CSAwesome 6.4 has several array algorithm Free Response Questions (FRQ) from real AP exams.

These FRQs build on the basic algorithms we just learned... but will usually have some practical application, may require more than one technique, and may have some curveball to them.

All are worth doing!