What we will cover:
Binary Search
Towers of Hanoi

Binary Search
Binary search involves a comparison of the middle element to a key. We can turn this into a recursive method because of the behavior of the algorithm. Recall that binary search must have a sorted array and localizes its search area based on where the key should be. If the key is less than the middle element, we only search the first half of the array; if the key is larger than the middle element, we search the last half of the array. If the middle element is the key, we return that index.

We can easily create a recursive method that sets the search area within in each call to the search method by adjusting our low, high, and mid pointers.

```java
public static int binarySearch(int[] data, int key, int low, int high) {
    if (low > high) {
        return -low - 1;
    }
    int mid = (low + high) / 2;
    if (key < data[mid]) {
        return binarySearch(data, key, low, mid - 1);
    } else if (key == data[mid]) {
        return mid;
    } else {
        return binarySearch(data, key, mid + 1, high);
    }
}
```

We can also create another method to allow the call to binary search to look like the iterative version:

```java
public static int binarySearch(int[] data, int key) {
    int low = 0;
    int high = data.length - 1;
    return binarySearch(data, key, low, high);
}
```

Towers of Hanoi
Towers of Hanoi is the one of the classic recursive problems in computer science. In fact, this problem has a simple solution when solved recursively, but is very difficult to solve iteratively. This problem has three towers (A, B, C) and n disks. All disks start on one tower and need to be moved to another tower while following a set of rules: no disk can be on top of a smaller disk at any time, all disks are initially on tower A, and only one disk can be moved at a time, this disk must be the top disk on a stack.

Class Exercise 50

Solving the problem involves three sub problems and a base case. If you have one disk, move it from tower A to tower B. If you have more than one, you must execute these three steps:
1. Move the first n-1 disks from A to C by way of B.
2. Move disk n from A to B.
3. Move n-1 disks from C to B by way of A.

If we were to create a method to execute this move, we would have a signature similar to this:
```java
void moveDisks (int n, char fromTower, char toTower, char auxTower)
```
Our algorithm for this would be:

```java
if (n == 1) {
    Move disk 1 from fromTower to toTower
} else {
    moveDisks(n-1, fromTower, auxTower, toTower);
    Move disk n from fromTower to toTower
    moveDisks(n – 1, auxTower, toTower, fromTower);
}
```

The actual code for this:

```java
public static void moveDisks(int n, char fromTower, char toTower, char auxTower) {
    if (n == 1) {
        System.out.println("Move disk "+n+" from "+fromTower+" to "+toTower);
    } else {
        moveDisks(n - 1, fromTower, auxTower, toTower);
        System.out.println("Move disk "+n+" from "+fromTower+" to "+toTower);
        moveDisks(n - 1, auxTower, toTower, fromTower);
    }
}

public static void main(String[] args) {
    int n = 3;
    System.out.println("Follow these moves:");
    moveDisks(n, 'A', 'B', 'C');
}
```