Section 2

CSCI E-22

Will Begin Shortly

Recall: Recursive Problem Solving

When solving problems using recursion, we break the problem down into smaller subproblems.

Once we have broken down the problem into the smallest subproblem (one that we can solve), we have reached a base case.

Then, we can progressively build up the solutions to the subproblems until we have a solution for the overall problem.

```
sum(n) =
```

Recall: Recursive Problem Solving

When solving problems using recursion, we break the problem down into smaller subproblems.

Once we have broken down the problem into the smallest subproblem (one that we can solve), we have reached a base case.

Then, we can progressively build up the solutions to the subproblems until we have a solution for the overall problem.

```
sum(n) = n + sum(n-1)
```

Recall: Recursive Problem Solving

When solving problems using recursion, we break the problem down into smaller subproblems.

Once we have broken down the problem into the smallest subproblem (one that we can solve), we have reached a base case.

Then, we can progressively build up the solutions to the subproblems until we have a solution for the overall problem.

```
sum(n) = n + sum(n-1)
```

```
public static int sum(int n) {
    if (n <= 0)
        return 0;

    int rest = sum(n - 1);
    return n + rest;
}
```

Recall: Recursive Problem Solving

When solving problems using recursion, we break the problem down into smaller subproblems.

Once we have broken down the problem into the smallest subproblem (one that we can solve), we have reached a base case.

Then, we can progressively build up the solutions to the subproblems until we have a solution for the overall problem.

```
sum(n) = n + sum(n-1)
```

```
public static int sum(int n) {
    if (n <= 0)
        return 0;

    int rest = sum(n - 1);
    return n + rest;
}
```

```
sum(4) = 4 + sum(3)
  sum(3) = 3 + sum(2)
    sum(2) = 2 + sum(1)
      sum(1) = 1 + sum(0)
        sum(0) = 0
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = ________________;

    ...



}
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = ____________________;

    ...



}
```

```
removeVowels("apple")
```

```
return "ppl"
```

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
- What's the recursive subproblem?
- What work do we need to do before returning?

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = ____________________;

    ...
```

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
  - When `s` is the empty string
- What's the recursive subproblem?
- What work do we need to do before returning?

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = ____________________;

    ...
```

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
  - When `s` is the empty string
- What's the recursive subproblem?
  - Removing the vowels from the rest of the string
- What work do we need to do before returning?

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = ____________________;

    ...
```

`removeVowels("apple")`

If we knew the solution to `removeVowels("pple")`, we could solve `removeVowels("apple")`

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
  - When `s` is the empty string
- What's the recursive subproblem?
  - Removing the vowels from the rest of the string
- What work do we need to do before returning?

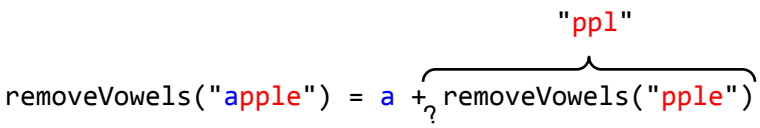
`removeVowels("apple") = a +? removeVowels("pple")`

If we knew the solution to `removeVowels("pple")`, we could solve `removeVowels("apple")`

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
  - When `s` is the empty string
- What's the recursive subproblem?
  - Removing the vowels from the rest of the string
- What work do we need to do before returning?

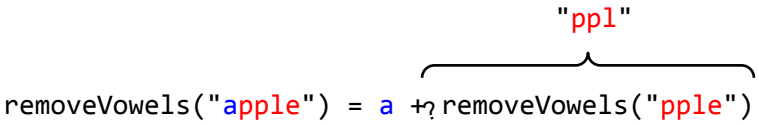


If we knew the solution to `removeVowels("pple")`, we could solve `removeVowels("apple")`

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
  - When `s` is the empty string
- What's the recursive subproblem?
  - Removing the vowels from the rest of the string
- What work do we need to do before returning?
  - Test whether the first character in the string is a vowel; if it is, it should **not** be included in the return value

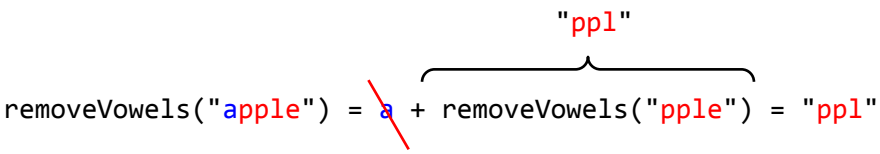


If we knew the solution to `removeVowels("pple")`, we could solve `removeVowels("apple")`

Recursive Problem Solving Approach

Before writing a recursive method, we can try to plan:

- What's the base case?
  - When `s` is the empty string
- What's the recursive subproblem?
  - Removing the vowels from the rest of the string
- What work do we need to do before returning?
  - Test whether the first character in the string is a vowel; if it is, it should **not** be included in the return value



If we knew the solution to `removeVowels("pple")`, we could solve `removeVowels("apple")`

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = ____________________;

    ...



}
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    ...



}
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (____________________) {
        ...


}
```


Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:


```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        ...


}
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        return removedFromRest;
    } else {
        ...
    }
}
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        return removedFromRest;
    } else {
        return first + removedFromRest;
    }
}
```

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        return removedFromRest;
    } else {
        return first + removedFromRest;
    }
}
```

How many calls of this method are needed when…

n is 0?

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        return removedFromRest;
    } else {
        return first + removedFromRest;
    }
}
```

How many calls of this method are needed when…

n is 0? 1
n is 1?

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        return removedFromRest;
    } else {
        return first + removedFromRest;
    }
}
```

How many calls of this method are needed when…

n is 0? 1
n is 1? 2
n is 2? 3

What is the general formula for the number of calls needed to process a string of length n?

Practice with Recursion

The following Java method should use recursion to remove all vowels from a string. Some of the code has been omitted and we will need to fill it in:

```
public static String removeVowels(String s) {
    if (s == null)
        throw new IllegalArgumentException();

    if (s.equals(""))
        return "";

    String removedFromRest = removeVowels(s.substring(1));

    char first = s.charAt(0);
    if (first == 'a' || first == 'e' || first == 'i' ||
        first == 'o' || first == 'u') {
        return removedFromRest;
    } else {
        return first + removedFromRest;
    }
}
```

How many calls of this method are needed when…

n is 0? 1
n is 1? 2
n is 2? 3

What is the general formula for the number of calls needed to process a string of length n?

n + 1

The Fibonacci Sequence

The Fibonacci sequence is a well-known number series in which each number in the series is the sum of the two previous numbers.

We define the first two numbers as $F_0 = 0$ and $F_1 = 1$, and all successive numbers as:

$$F_n = F_{n-1} + F_{n-2}$$

Since the sequence is defined recursively, using recursion to calculate Fibonacci numbers is natural. Finish the code below to write an algorithm that calculates the *n*th Fibonacci number.

```
public static long fib(int n) {

    // base case
    if (_________________)




}
```


The Fibonacci Sequence

The Fibonacci sequence is a well-known number series in which each number in the series is the sum of the two previous numbers.

We define the first two numbers as $F_0 = 0$ and $F_1 = 1$, and all successive numbers as:

$$F_n = F_{n-1} + F_{n-2}$$

Since the sequence is defined recursively, using recursion to calculate Fibonacci numbers is natural. Finish the code below to write an algorithm that calculates the *n*th Fibonacci number.


```
public static long fib(int n) {

    // base case
    if (n == 0 || n == 1)
        return _________;




}
```

The Fibonacci Sequence

The Fibonacci sequence is a well-known number series in which each number in the series is the sum of the two previous numbers.

We define the first two numbers as $F_0 = 0$ and $F_1 = 1$, and all successive numbers as:

$$F_n = F_{n-1} + F_{n-2}$$

Since the sequence is defined recursively, using recursion to calculate Fibonacci numbers is natural. Finish the code below to write an algorithm that calculates the *n*th Fibonacci number.

```
public static long fib(int n) {

    // base case
    if (n == 0 || n == 1)
        return n;



}
```


The Fibonacci Sequence

The Fibonacci sequence is a well-known number series in which each number in the series is the sum of the two previous numbers.

We define the first two numbers as $F_0 = 0$ and $F_1 = 1$, and all successive numbers as:

$$F_n = F_{n-1} + F_{n-2}$$

Since the sequence is defined recursively, using recursion to calculate Fibonacci numbers is natural. Finish the code below to write an algorithm that calculates the *n*th Fibonacci number.


```
public static long fib(int n) {

    // base case
    if (n == 0 || n == 1)
        return n;

    // recursive case
    return ____________________;

}
```

The Fibonacci Sequence

The Fibonacci sequence is a well-known number series in which each number in the series is the sum of the two previous numbers.

We define the first two numbers as $F_0 = 0$ and $F_1 = 1$, and all successive numbers as:

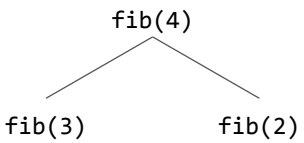
$$F_n = F_{n-1} + F_{n-2}$$

Since the sequence is defined recursively, using recursion to calculate Fibonacci numbers is natural. Finish the code below to write an algorithm that calculates the *n*th Fibonacci number.

```
public static long fib(int n) {

    // base case
    if (n == 0 || n == 1)
        return n;

    // recursive case
    return fib(n - 1) + fib(n - 2);

}
```

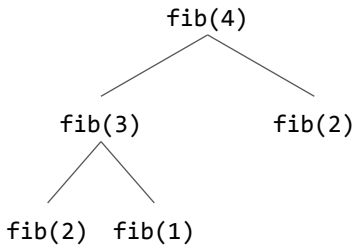
The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:



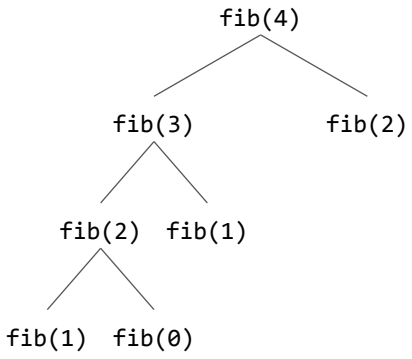
The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:



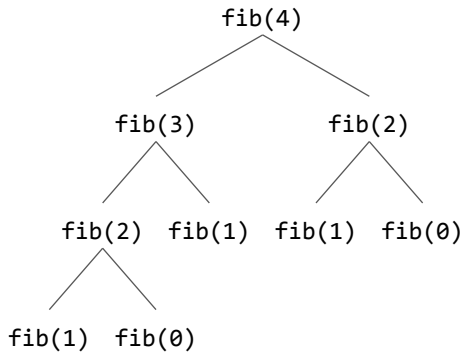
The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:



The Fibonacci Sequence

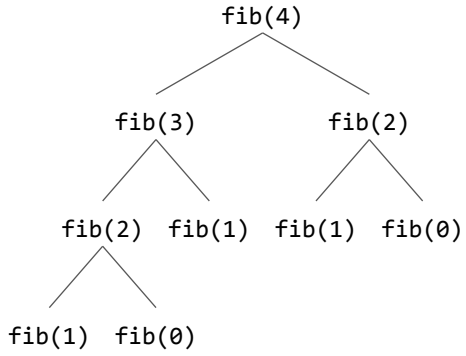
Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:



The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:

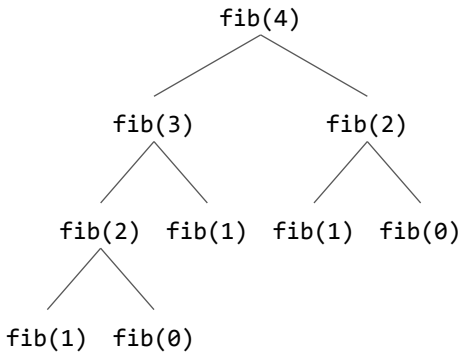
How many times did we call `fib()` to find the fourth Fibonacci number?



The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:

How many times did we call `fib()` to find the fourth Fibonacci number? 9

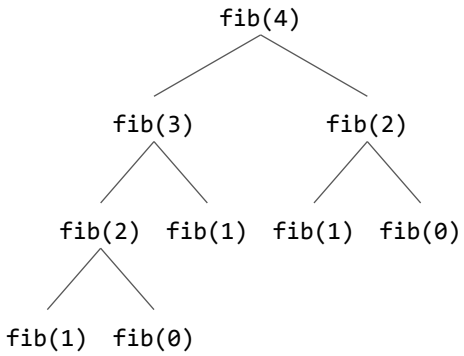


The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:

How many times did we call `fib()` to find the fourth Fibonacci number? 9

Do you see a problem with this? What if we tried larger numbers, like 50?



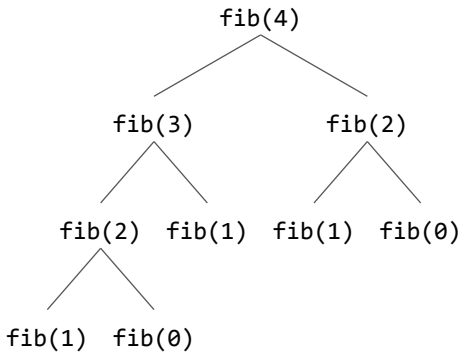
The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:

How many times did we call `fib()` to find the fourth Fibonacci number? 9

Do you see a problem with this? What if we tried larger numbers, like 50?

- The number of calls increases exponentially for the initial value of *n*. This means that we would have to make about 10 billion method calls to calculate `fib(50)`!



The Fibonacci Sequence

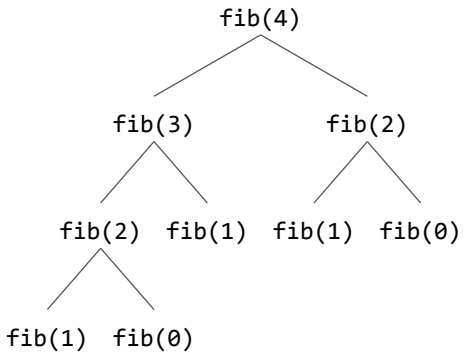
Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:

How many times did we call `fib()` to find the fourth Fibonacci number? 9

Do you see a problem with this? What if we tried larger numbers, like 50?

- The number of calls increases exponentially for the initial value of *n*. This means that we would have to make about 10 billion method calls to calculate `fib(50)`!

How would you rewrite `fib()` to be more efficient, either still as a recursive method or iteratively?



The Fibonacci Sequence

Draw a diagram that shows the number of times `fib()` is called with an initial value of 4. It has been started for you:

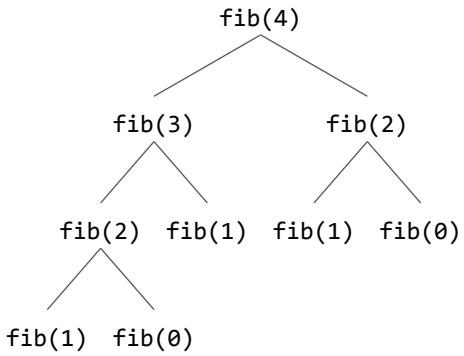
How many times did we call `fib()` to find the fourth Fibonacci number? 9

Do you see a problem with this? What if we tried larger numbers, like 50?

- The number of calls increases exponentially for the initial value of *n*. This means that we would have to make about 10 billion method calls to calculate `fib(50)`!

How would you rewrite `fib()` to be more efficient, either still as a recursive method or iteratively?

- You need to keep track of the previous two Fibonacci values while computing the next



The Fibonacci Sequence

Here is an example iterative solution:

```
public static long fib(int n) {
    if (n <= 0) {
        return 0;
    }

    long previous = 0;      // at the start, previous = F_0
    long current = 1;       // at the start, current = F_1

    for (int i = 2; i <= n; i++) {
        long tmp = previous + current;
        previous = current;
        current = tmp;
    }

    return current;
}
```