Announcement

- The fourth test will be 75 minutes, will consist of two parts and will take place next week.
- The programming part will be about Chapter 2-6, excluding Section 2.6, 4.5, and 6.8.8.
- The "written" part will be about Chapter 2-6, excluding Section 2.6, 4.5, and 6.8.8.
- During the test, you will have access to the textbook. You may bring a blank piece of paper to the test

Recursion

notes Chapter 8

Fibonacci Numbers

- the sequence of additional pairs
  - `0, 1, 1, 2, 3, 5, 8, 13, ...`

  are called Fibonacci numbers

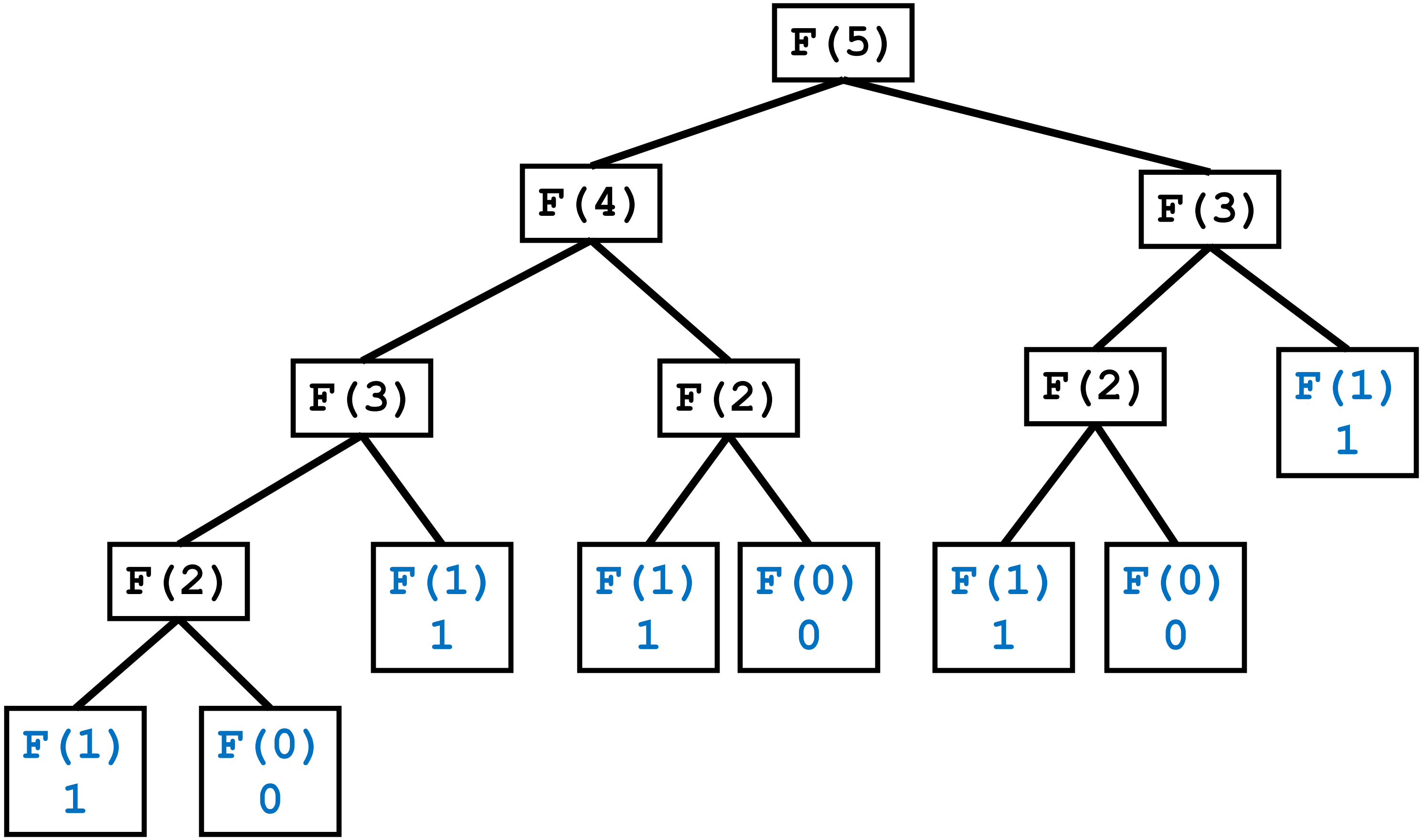
- base cases
  - `F(0) = 0`
  - `F(1) = 1`
- recursive definition
  - `F(n) = F(n – 1) +  F(n – 2)`

Recursive Methods & Return Values

- a recursive method can return a value
- example: compute the nth Fibonacci number

```
public static int fibonacci(int n) {
    if (n == 0) {
        return 0;
    }
    else if (n == 1) {
        return 1;
    }
    else {
      int f = fibonacci(n - 1) + fibonacci(n - 2);
      return f;
    }
}
```

Fibonacci Call Tree

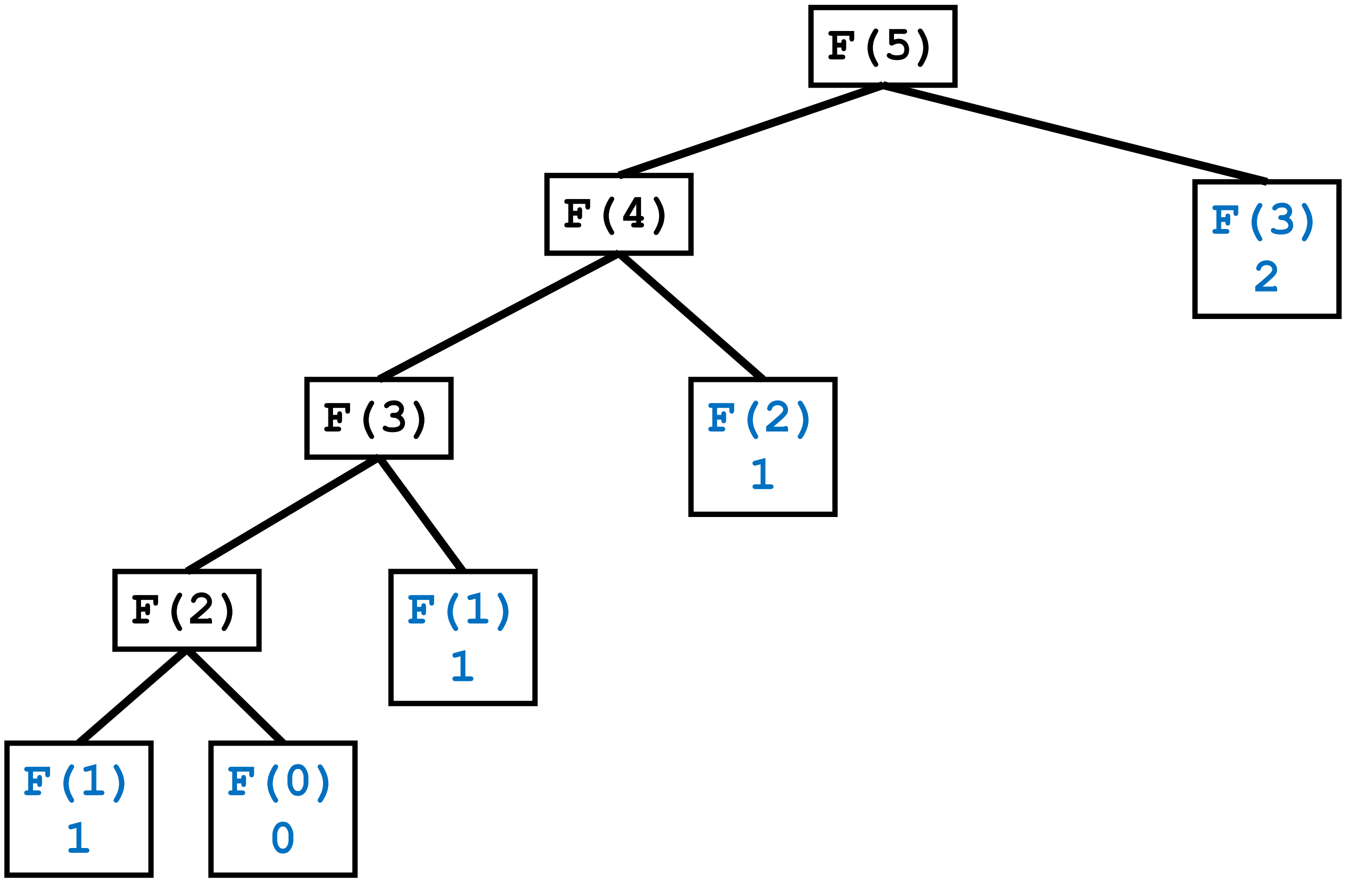


A Better Recursive Fibonacci

```
public class Fibonacci {
  private static Map<Integer, Long> values = new HashMap<Integer, Long>();
  static {
  Fibonacci.values.put(0, (long) 0);
  Fibonacci.values.put(1, (long) 1);
  }

  public static long getValue(int n) {
    Long value = Fibonacci.values.get(n);
    if (value != null) {
      return value;
    }
    value = Fibonacci.getValue(n - 1) + Fibonacci.getValue(n - 2);
    Fibonacci.values.put(n, value);
    return value;
  }
}
```

Better Fibonacci Call Tree



Compute Powers of 10

- write a recursive method that computes 10^n for any integer value **n**
- recall:
  - $10^n = 1 / 10^{-n}$ if $n < 0$
  - $10^0 = 1$
  - $10^n = 10 * 10^{n-1}$

```java
public static double powerOf10(int n) {
  if (n < 0) {
    return 1.0 / powerOf10(-n);
  }
  else if (n == 0) {
    return 1.0;
  }
  return n * powerOf10(n - 1);
}
```

A Better Powers of 10

- recall:
  - $10^{n} = 1 / 10^{-n}$ if $n < 0$
  - $10^{0} = 1$
  - $10^{n} = 10 * 10^{n-1}$ if n is odd
  - $10^{n} = 10^{n/2} * 10^{n/2}$ if n is odd

```java
public static double powerOf10(int n) {
  if (n < 0) {
    return 1.0 / powerOf10(-n);
  }
  else if (n == 0) {
    return 1.0;
  }
  else if (n % 2 == 1) {
    return 10 * powerOf10(n - 1);
  }
  double value = powerOf10(n / 2);
  return value * value;
}
```

Proving Correctness and Termination

- to show that a recursive method accomplishes its goal you must prove:
  1. that the base case(s) and the recursive calls are correct
  2. that the method terminates

Proving Correctness

- to prove correctness:
  1. prove that each base case is correct
  2. assume that the recursive invocation is correct and then prove that each recursive case is correct

printItToo

```
public static void printItToo(String s, int n) {
  if (n == 0) {
    return;
  }
  else {
    System.out.print(s);
    printItToo(s, n - 1);
  }
}
```

Correctness of printItToo

1. (prove the base case) If `n == 0` nothing is printed; thus the base case is correct.
2. Assume that `printItToo(s, n-1)` prints the string `s` exactly `(n - 1)` times. Then the recursive case prints the string `s` exactly `(n - 1)+1 = n` times; thus the recursive case is correct.

Proving Termination

- to prove that a recursive method terminates:
  1. define the size of a method invocation; the size must be a non-negative integer number
  2. prove that each recursive invocation has a smaller size than the original invocation

Termination of printItToo

1. **`printItToo(s, n)`** prints **`n`** copies of the string **`s`**; define the size of **`printItToo(s, n)`** to be **`n`**
2. The size of the recursive invocation **`printItToo(s, n-1)`** is **`n-1`** (by definition) which is smaller than the original size **`n`**.

countZeros

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

  if(n == 0L) {  // base case 1
    return 1;
  }
  else if(n < 10L) {  // base case 2
    return 0;
  }

  boolean lastDigitIsZero = (n % 10L == 0);
  final long m = n / 10L;
  if(lastDigitIsZero) {
    return 1 + countZeros(m);
  }
  else {
    return countZeros(m);
  }
}
```

Correctness of countZeros

1. (base cases) If the number has only one digit then the method returns **`1`** if the digit is zero and **`0`** if the digit is not zero; therefore, the base case is correct.
2. (recursive cases) Assume that **`countZeros(n/10L)`** is correct (it returns the number of zeros in the first **`(d - 1)`** digits of **`n`**). If the last digit in the number is zero, then the recursive case returns **`1 +`** the number of zeros in the first **`(d - 1)`** digits of **`n`**, otherwise it returns the number of zeros in the first **`(d - 1)`** digits of **`n`**; therefore, the recursive cases are correct.

Termination of countZeros

1. Let the size of **`countZeros(n)`** be **`d`** the number of digits in the number **`n`**.
2. The size of the recursive invocation **`countZeros(n/10L)`** is **`d-1`**, which is smaller than the size of the original invocation.

Proving Correctness and Termination

- prove that `fibonacci` is correct and terminates
- prove that `powersOf10` is correct and terminates

Proving Termination

- prove that the algorithm on the next slide terminates

```java
public class Print {

  public static void done(int n) {
    if (n == 1) {
      System.out.println("done");
    }
    else if (n % 2 == 0) {
      System.out.println("not done");
      Print.done(n / 2);
    }
    else {
      System.out.println("not done");
      Print.done(3 * n + 1);
    }
  }

}
```