Week 5 - Friday

COMP 2100

Last time

- What did we talk about last time?
- Started recursion

Questions?

Project 2

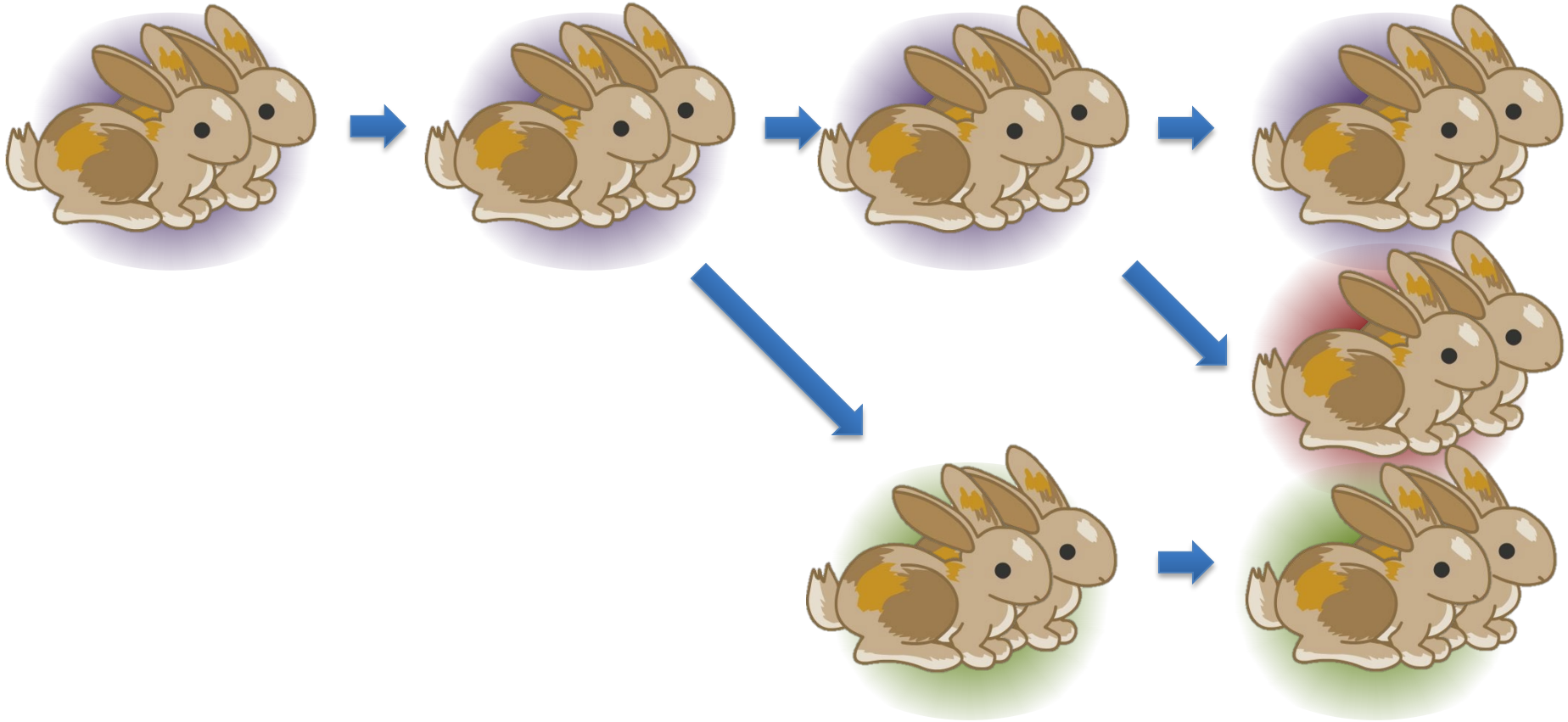
Assignment 3

Exam 2 Post Mortem

Issues of Efficiency

Fibonacci

- The sequence: 1 1 2 3 5 8 13 21 34 55…
- Studied by Leonardo of Pisa to model the growth of rabbit populations



Fibonacci Problem

- Find the n^{th} term of the Fibonacci sequence
- Simple approach of summing two previous terms together
- Example: $n = 7$
- `1 1 2 3 5 8 13`
  `1 2 3 4 5 6 7`

Recursion for Fibonacci

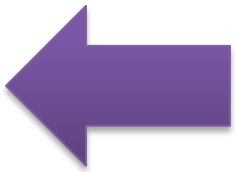
- Base cases (***n*** = 1 and ***n*** = 2):
  - Result = 1
- Recursive case (***n*** > 2):
  - Result = fibonacci(***n*** – 1) + fibonacci(***n*** – 2)

Code for Fibonacci

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

  if (n <= 2) {
     return 1;
  } else {
     return fib(n - 1) + fib(n - 2);
  }

}
```

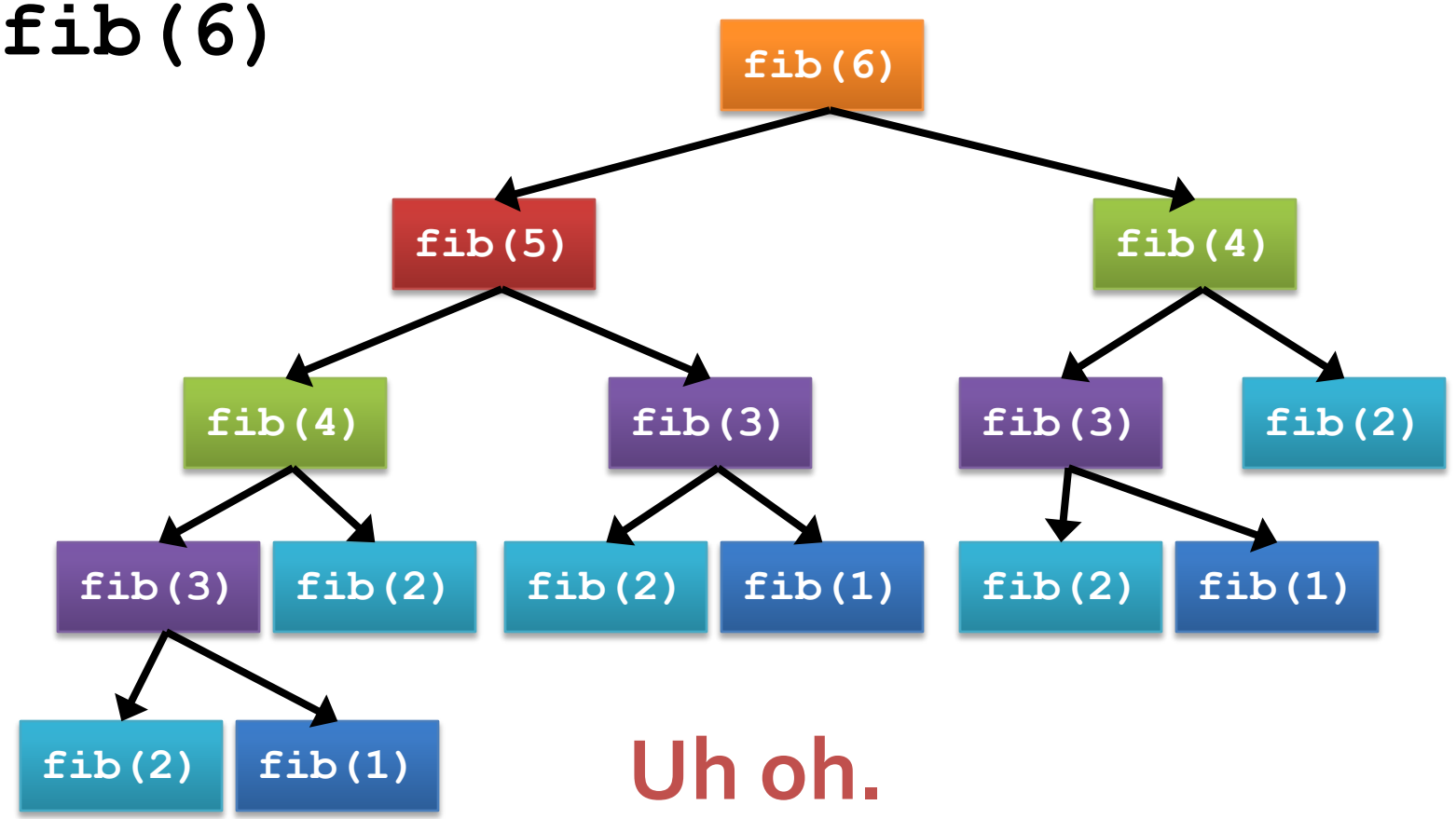


Base Case

Recursive Case

What's the running time for `fib()`?

- Example: **`fib(6)`**



Uh oh.

Exponential Time for `fib`

- For most cases, calling **`fib()`** makes calls two more calls to **`fib()`**, which each make two more calls to **`fib()`**, and so on…
- Many values are redundantly computed
- The final running time is $O(2^{n/2})$

Can we do better?

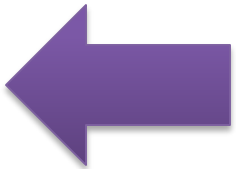
- The recursion is fine from a mathematical perspective
- We just need to avoid recomputing lower terms in the sequence
- We can use the idea of carrying along both the ($n - 1$) term and the ($n - 2$) term in each recursive step

Code for Better Fibonacci

```
public static int fib2( int a, int b, int n ) {

  if (n <= 2) {
     return b;
  } else {
     return fib2(b, a + b, n - 1);
  }
}

// proxy method
int fib( int n ) {
  return fib2(1, 1, n);
}
```



Base Case

Recursive Case

Exponentiation

- We want to raise a number **x** to a power ***n***, like so: x^n
- We allow ***x*** to be real, but ***n*** must be an integer greater than or equal to 0
- Example: $(4.5)^{13} = 310286355.9971923828125$

Recursion for Exponentiation

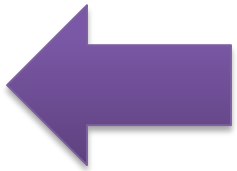
- Base case ($n = 0$):
  - Result = 1
- Recursive case ($n > 0$):
  - Result = $x \cdot x^{(n-1)}$

Code for Exponentiation

```
public static double power(double x, int n) {

  if (n == 0) {
     return 1;
  } else {
     return x * power(x, n - 1);
  }

}
```



Base Case

Recursive Case

Running time for `power`

- Each call reduces ***n*** by 1
- ***n*** + 1 total calls
- What's the running time?
  - $\Theta(n)$

Quiz

Upcoming

Next time…

- Finish recursive running time
- Symbol tables
- Merge sort
- Trees

Reminders

- Keep working on Project 2
- Start working on Assignment 3