Computer Science 1

Lecture 6

Recursion

Labs

- Pay attention to the Minesweeper Lab

- Check your grades for short feedback
  - Also for errors in submissions, etc.

- **Stay motivated and keep practicing**
  - **Ask for help if you need it!**

Overview

Recursion

- What?
- Why?
- How?

... most students add “in God’s name” to each of these questions ...

More examples/questions in the remaining time …

Recursion

A different kind of loop

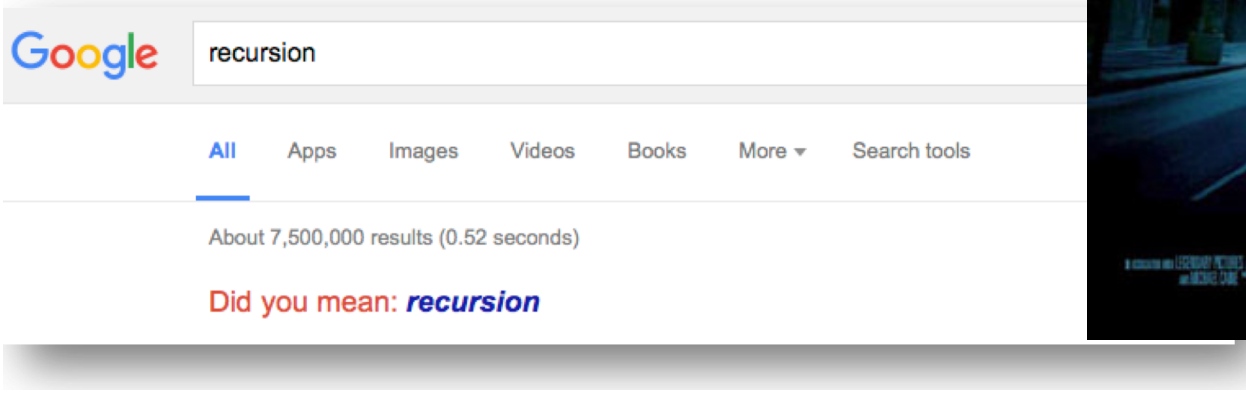
= Building a loop by having a method call itself

Needs:

- stopping criterion (base case)
- self invocation with parameters closer to base-case

We don't often tell you to, but please read Chapter 13 from the book!

Examples in Nature



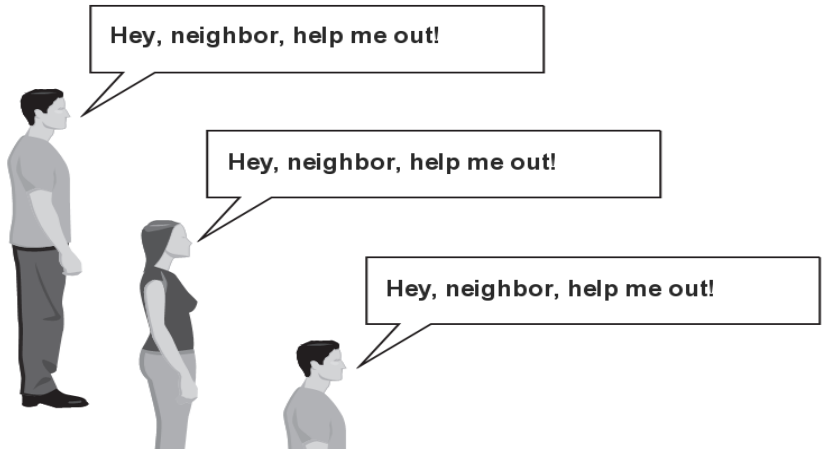
Recursive solutions

Find a leaf of the tree:

- Climb the current branch to the next split
- Choose side
- Find a leaf of (the remainder of) the tree

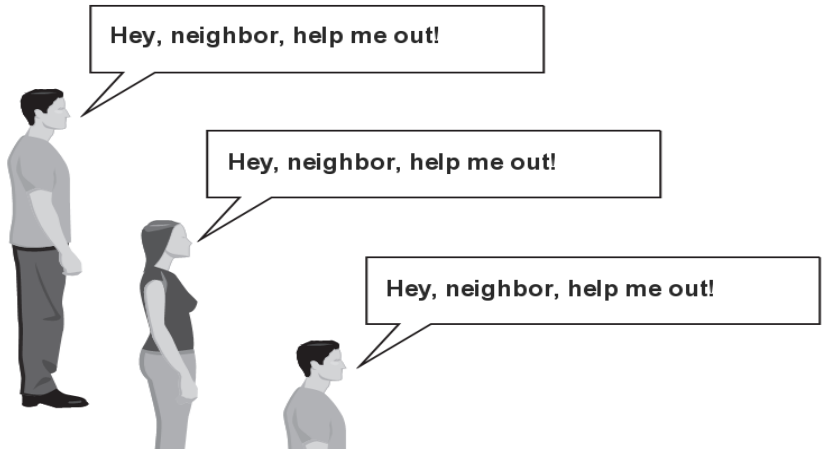
The idea

- Recursion is all about breaking a big problem into smaller occurrences of that same problem.
  - Each person can solve a small part of the problem.
    - What is a small version of the problem that would be easy to answer?
    - What information from a neighbor might help me?



Let’s play it

- Number of people behind me:
  - If there is someone behind me,
    ask him/her how many people are behind him/her.
    - When they respond with a value **N**, then I will answer **N + 1**.
  - If there is nobody behind me, I will answer **0**.

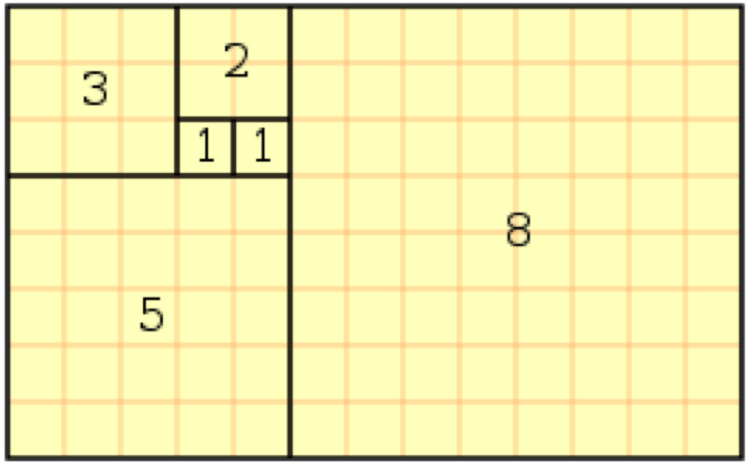


Example from math

Fibonacci numbers

≈ breeding rabbits and other natural phenom.

1. Start with 0 and 1
2. Next number is previous two numbers added



$$F_0 = 0, F_1 = 1$$

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

0 – 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21 – 34 – 55 – …

Fibonacci using while

```
public static int fib(int index) {
    if (index == 0)
        return 0;
    else if (index == 1)
        return 1;
    else {
        int fnMinus2 = 0;
        int fnMinus1 = 1;
        int n = 2;
        int f = 1;
        while (n < index) {
            n++;
            fnMinus2 = fnMinus1;
            fnMinus1 = f;
            f = fnMinus1 + fnMinus2;
        }
        return f;
    }
}
```

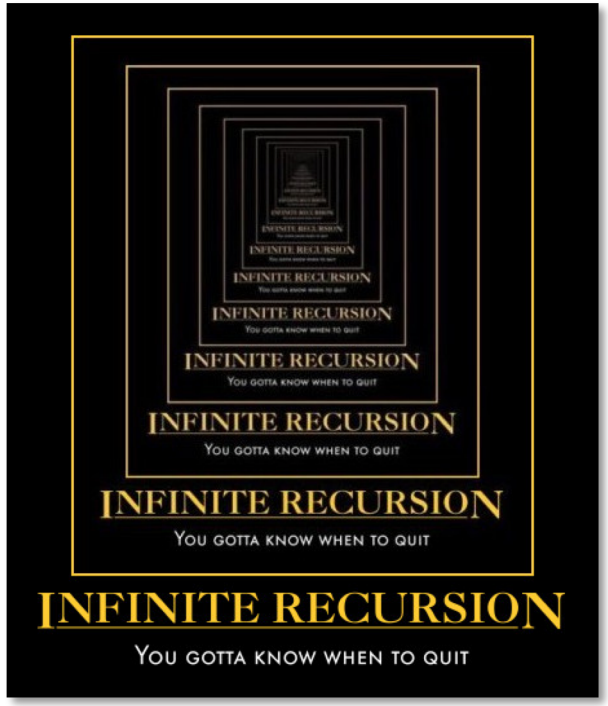
Fibonacci using recursion

```
public static int fib(int index) {
    if (index == 0)
        return 0;
    else if (index == 1)
        return 1;
    else
        return fib(index-1) + fib(index-2);
}
```

Recursion in general

2 important parts

1. Base case
   - handles the simplest cases directly with (almost) no computation
2. Recursive call
   - must simplify the computation in some way



Recursion scheme

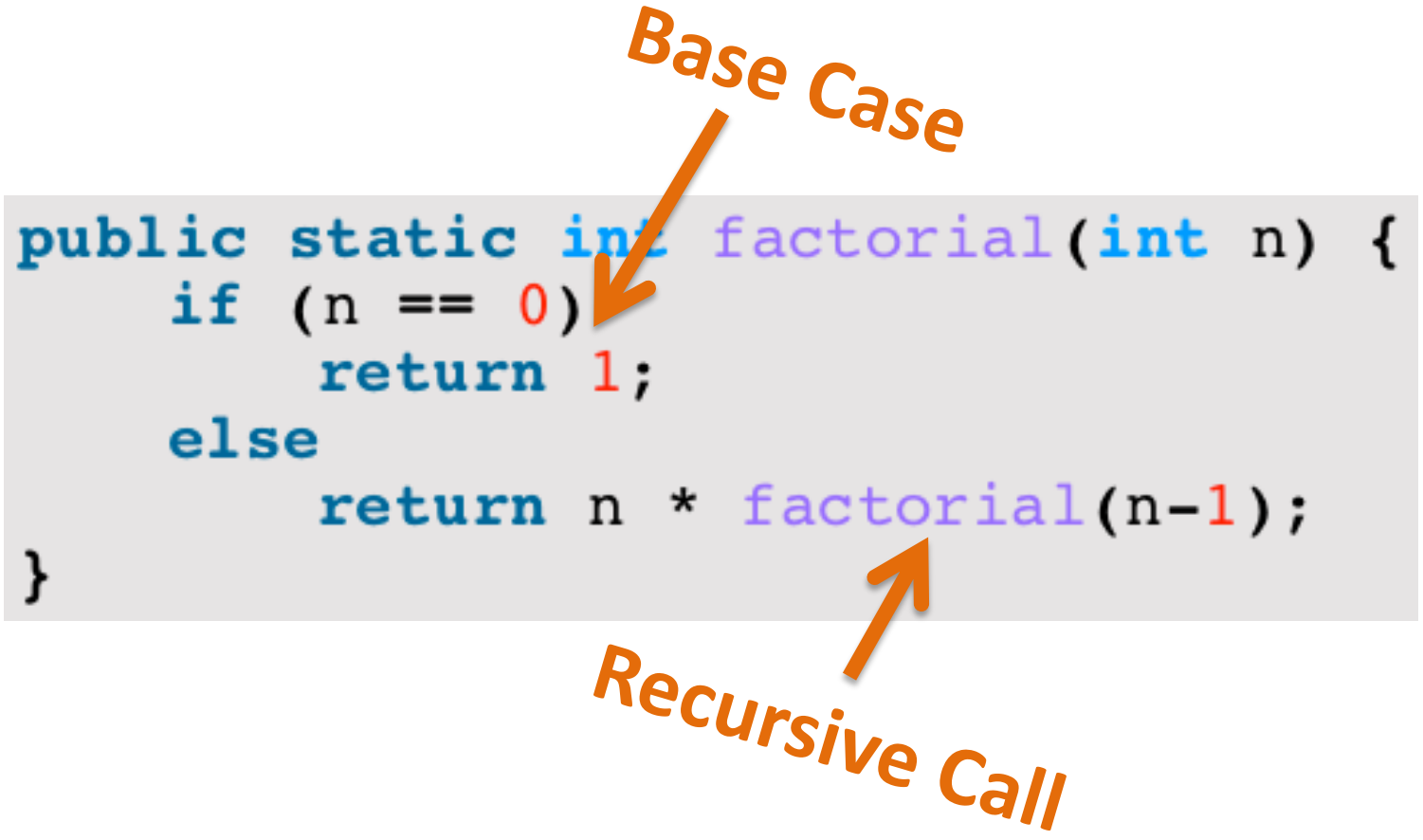
```
public static <type> f(x) {
    if (base_case(x))
        return result;
    else {
        y = reduce(x);
        return … f(y) …;
    }
}
```

Is this input for which the answer is trivial?

Brings the input closer to the trivial case.

Compute the answer for the original input using the answer for the simpler case.

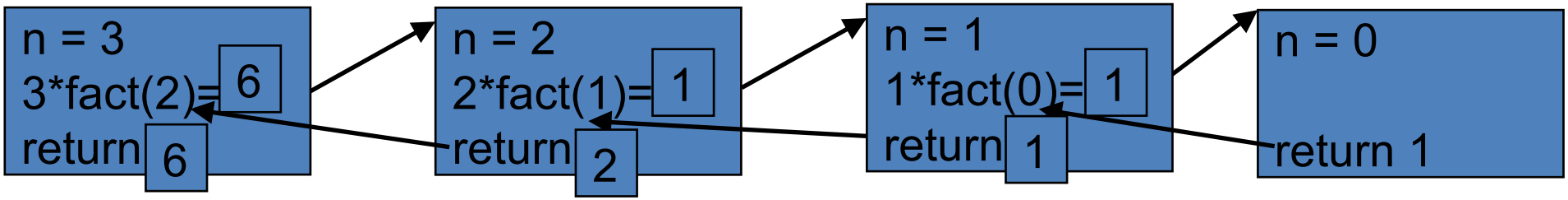
Another example: factorial



Trace call == Recursive stack

I need factorial(3)

```
public static int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n-1);
}
```



So, factorial(3)=6

- When the stack gets too deep/big or when it's infinite then…

Mutual recursion

More than one method that call each other

```
public static boolean isOdd(int x) {
        return isEven(Math.abs(x-1));
}

public static boolean isEven(int x) {
    if (x == 0)
        return true;
    else if (x == 1)
        return false;
    else
        return isOdd(Math.abs(x-1));
}
```

The power of recursion

Isolation of complexity

1. Separate out base case(s)

2. Separate out single step(s) of a process

Backtracking

Example problem:

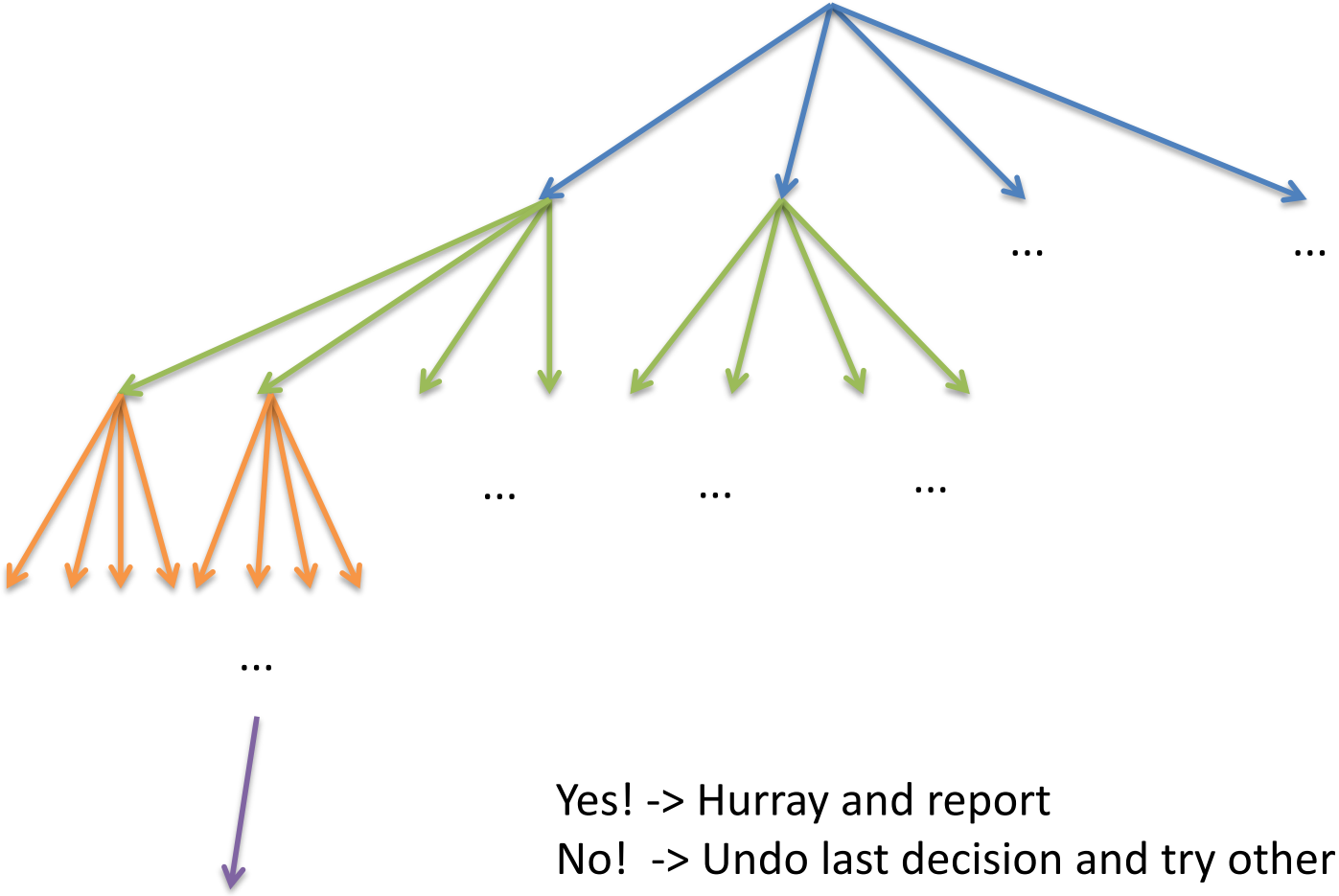
Given an array of ints, is it possible to choose a group of some of the ints, such that the group sums to the given target?

{1,2,3,4,5,6,7,8,9} can form 10? Yes, for example 1+2+3+4

{2,2,3,4,5} can form 6? Yes, 2+4

{2,3,5,12,20) can form 11? No

Backtracking search



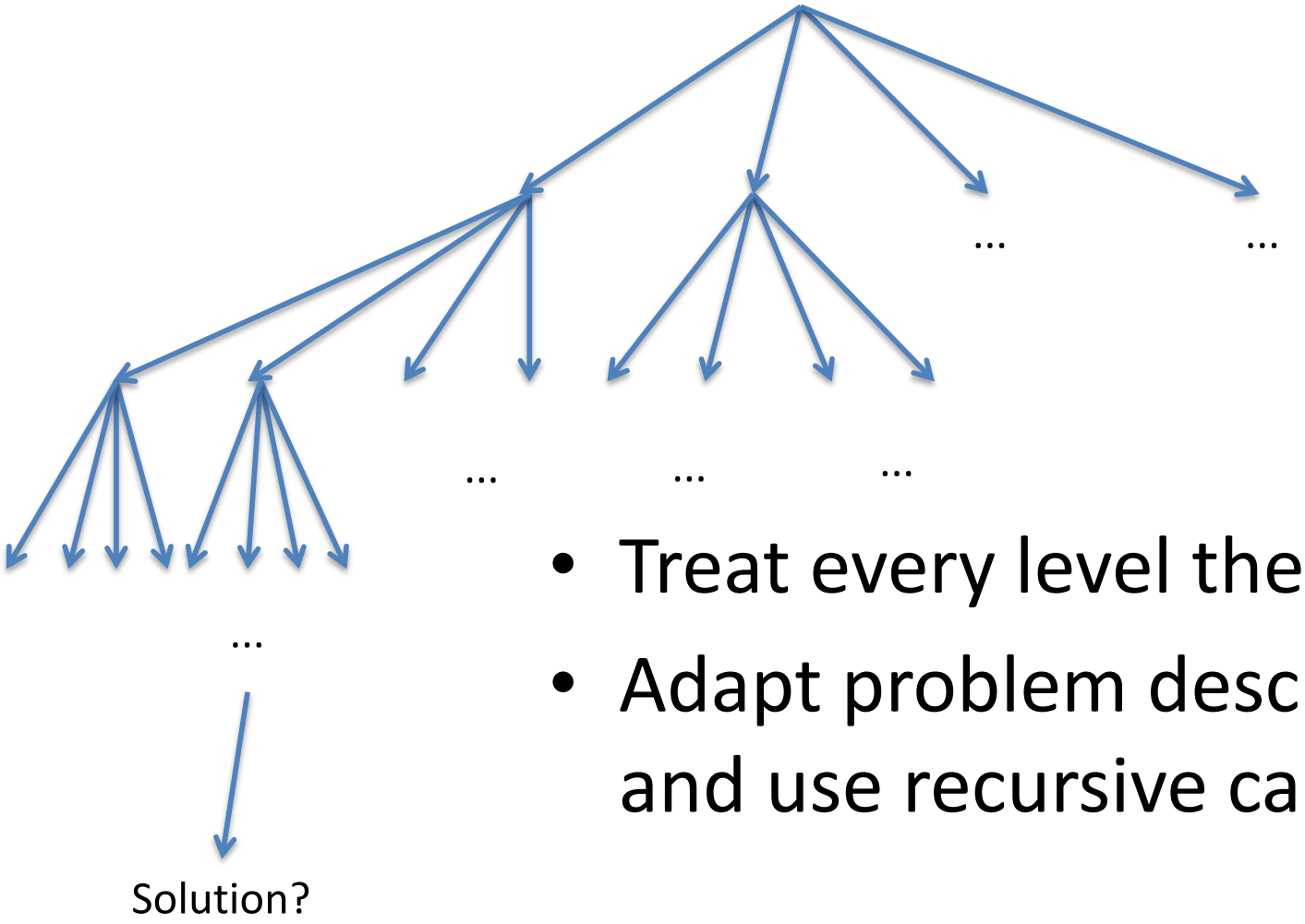
Yes! -> Hurray and report
No! -> Undo last decision and try other alternative

Bookkeeping

- Remember the choices at each level
- Remember which level of search
- Remember what has been tried
- Remember what still needs to be tried

How will you handle this with while or for loops?

Recursion to the rescue!



- Treat every level the same
- Adapt problem description and use recursive call.

Example problem again

How to check for a solution?

- Number reached?
- Out of elements?
- Otherwise: search on!

So two base cases!

Very simple selection problem at each step:

"Is the next element to be used or not?"

- **Yes**, so subtract it from the goal
- **No**, keep searching

So two recursive calls!

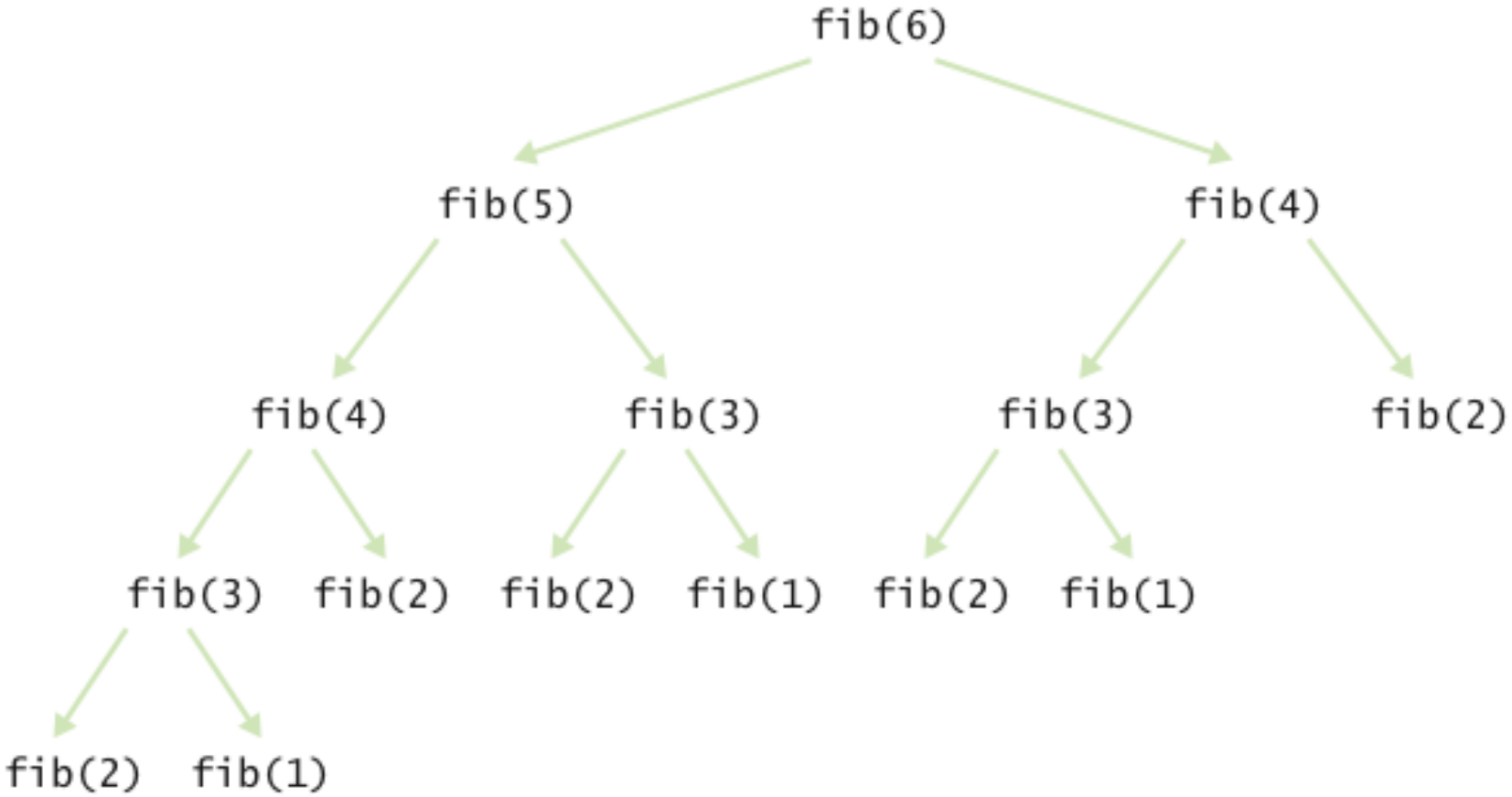
In Java

```
public static boolean findTotal(int goal, int[] array) {

    if (goal==0) {
        // Target found!
        System.out.print("These numbers can be used: ");
        return true;
    } else if (array.length == 0) {
        // no solution to be found
        return false;
    } else {
        int newgoal = goal-array[0];
        int[] newarray = new int[array.length-1];
        System.arraycopy(array,1,newarray,0,array.length-1);
        if (findTotal(newgoal,newarray)) {
            System.out.print(array[0]+" ");
            return true;
        } else
            return findTotal(goal,newarray);
    }
}
```

Efficiency of Recursion

CallTree for computing fib(6)



Fibonacci Revisited

```
public static long fib(int index) {
    if (index == 0) return 0;
    else if (index == 1) return 1;
    else return fib(index-1)+fib(index-2);
}
```

What about adding some memory of what was already computed?

often used, is called “Tabling”

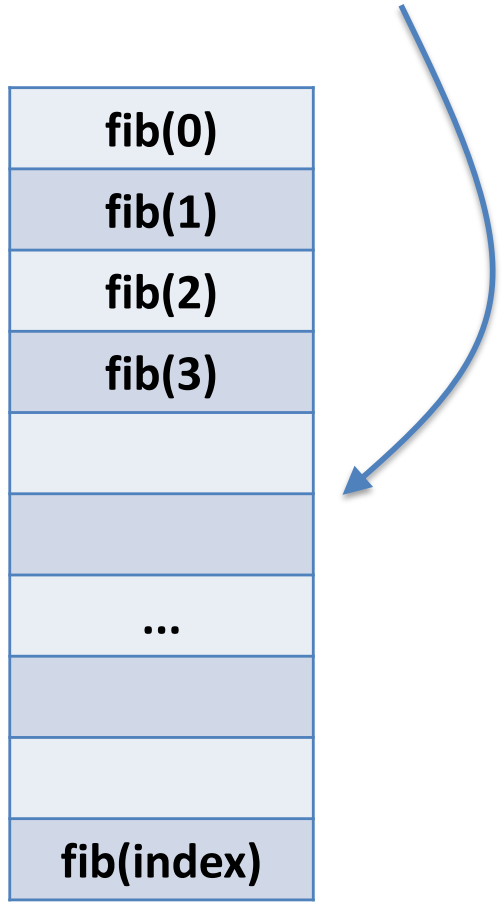
Tabling

```
public static long fib(int index, long[] lookUpTable)
```

Goal: calculate every result once

- Use an array to remember
- Fill in results as computed

- Construct on first call to method
- Pass along array as an extra parameter



In Java

```
public static long fib(int index, long[] lookUpTable) {

    return lookUpTable[index];
}
```

Verdict on recursion?

- Occasionally, a recursive solution runs much slower than its iterative counterpart
  - In most cases, the recursive solution is only slightly slower
  - Smart compilers can avoid recursive method calls if they follow simple patterns

    Most compilers don't do that
- In many cases, a recursive solution is easier to understand and implement correctly than an iterative solution

"To iterate is human, to recurse divine." L. Peter Deutsch

Next week:

- A lab, number 7 on “Recursion”
- A lecture ... Number 7 and the last one!!!
- Another lab, number 8

Then: project week ...

The last week:

- Exam prep on Monday
- Questions + competition on Tuesday