Introduction to Computer Science I

Module 6 – Recursion

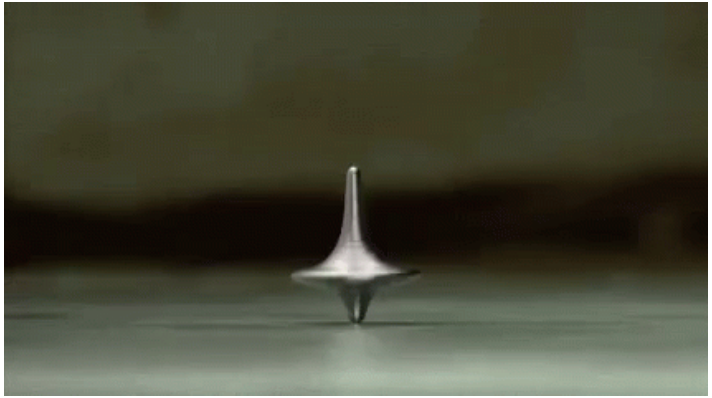
Overview

Recursion

- What?
- Why?
- How?

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

Recursion



Recursion

A different kind of loop

= Building a loop by having a method calling itself

Needs:

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

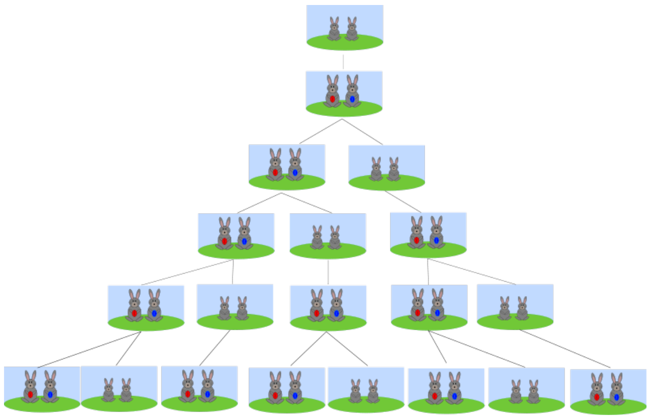
Examples in Nature

Example from math

Fibonacci numbers

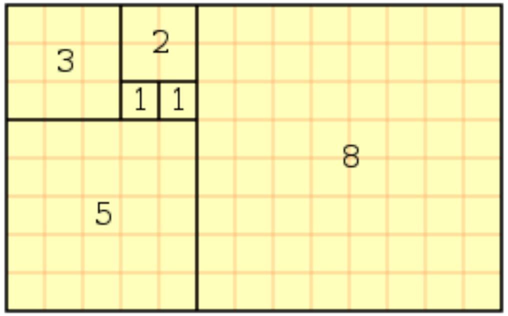
≈ breeding rabbits and other natural phenomena

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



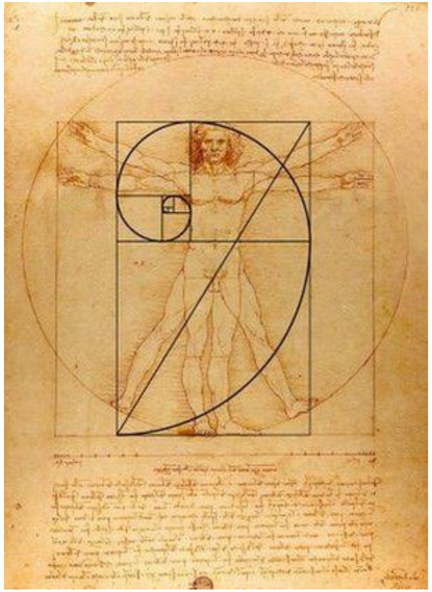
0
1
1
2
3
5
8
...

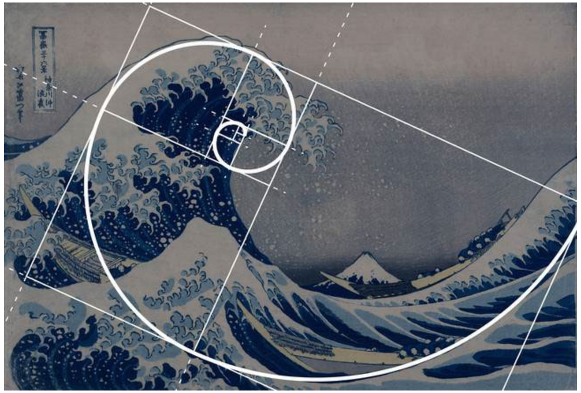
0 – 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21 – 34 – 55 – …

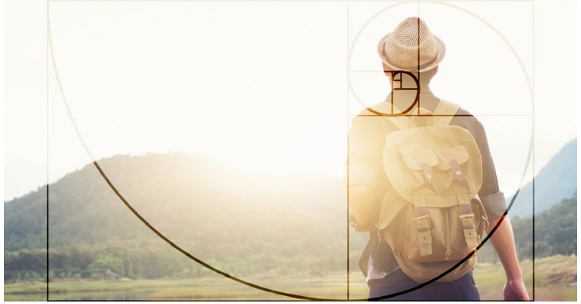


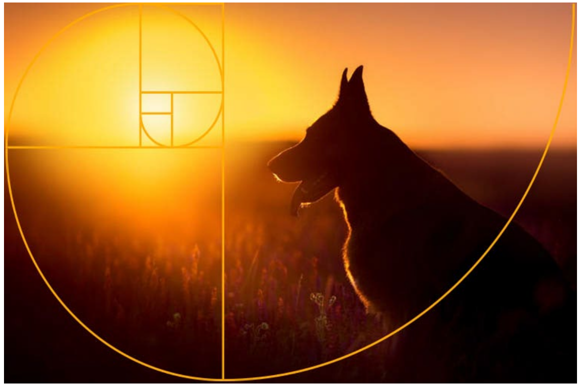
$$F_0 = 0, F1 = 1$$
$$F_n = F_{n-1} + \mathrm{F}_{\mathrm{n}-2}$$

And more









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

Recursion

Printing numbers from 0 to 10

With FOR and WHILE loops:

```
for (int i=0; i<=10; i++)
  System.out.print(i + " ");
```

```
int i = 0;
while (i <= 10) {
  System.out.print(i + " ");
  i++;
}
```

A recursive way:

```
public static void forToRecursion (int i) {
  System.out.print(i + " ");
  i++;
  if (i <=10)
    forToRecursion(i);
}
```

Fibonacci using while

```java
public static int fibWhile (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;
  }
}
```

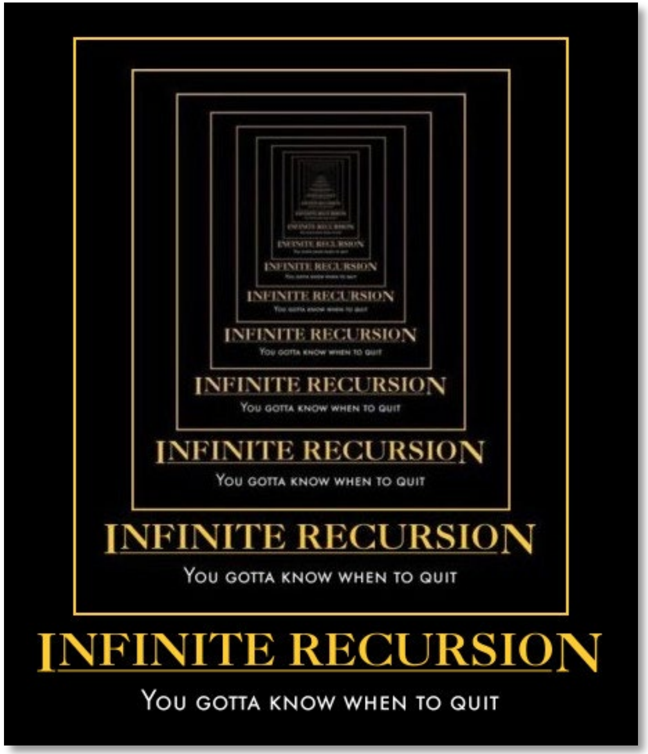
Fibonacci using recursion

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

Recursion in general

Two 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

Base Case

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

Recursive Call

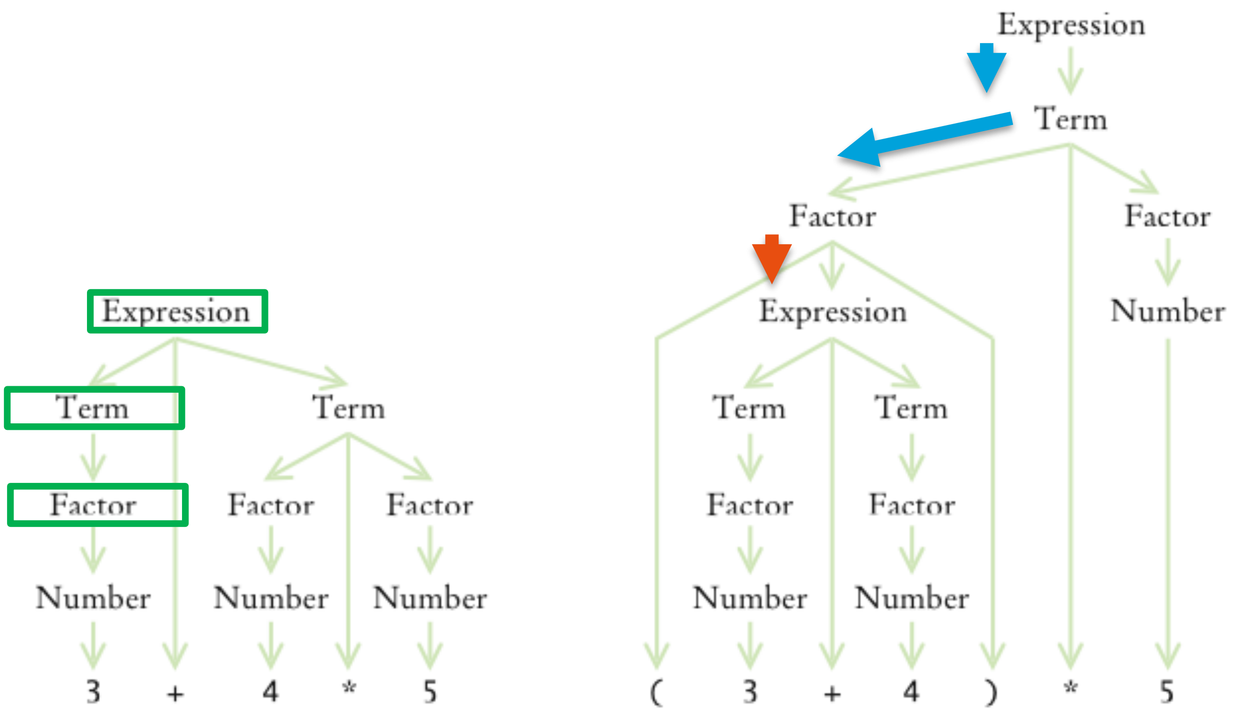
Mutual recursion

More than one method that call each other

```java
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));
}
```

Mutual recursion (2)



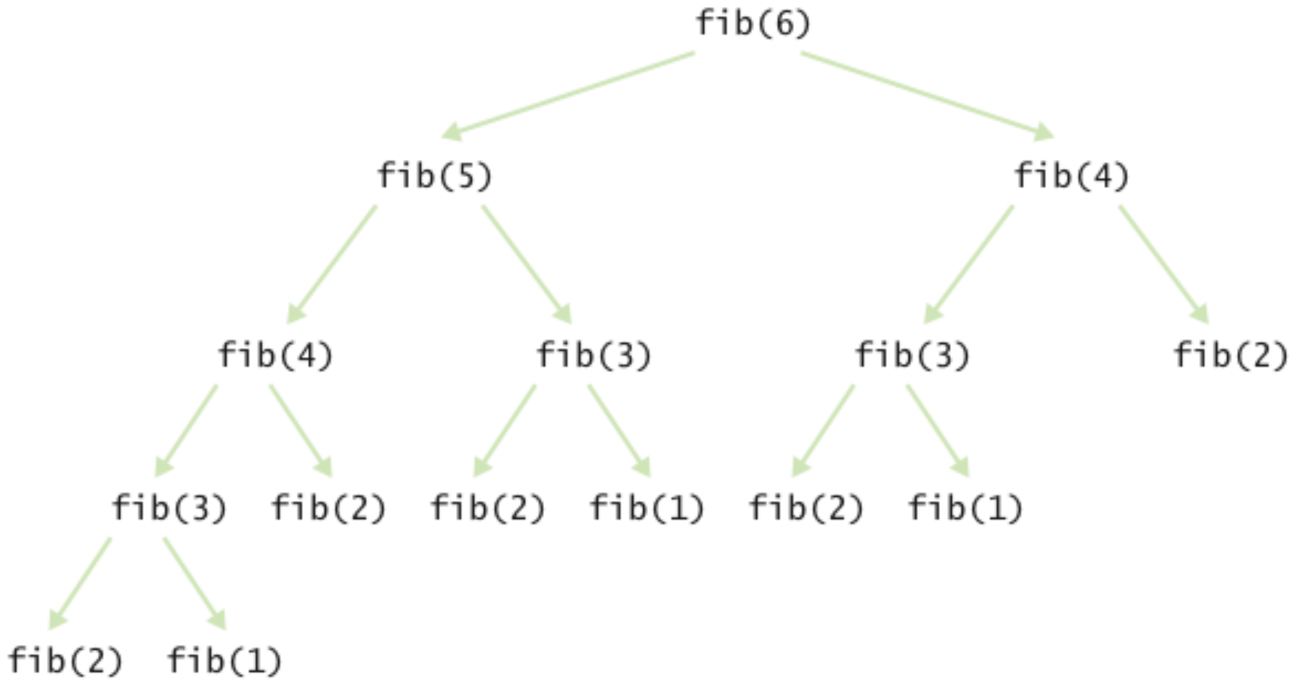
The power of recursion

Isolation of complexity

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

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"

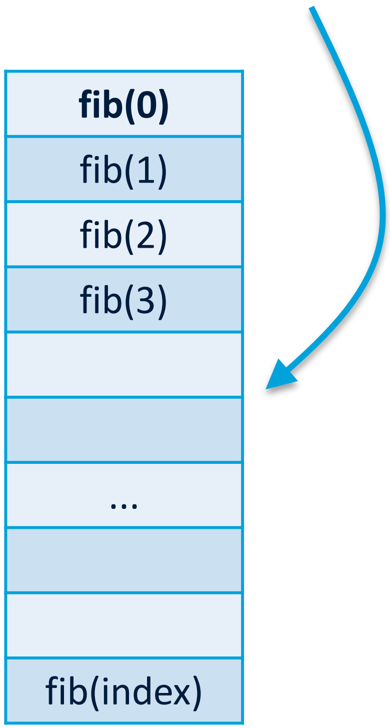
Tabling

```java
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) {
    if (lookUpTable == null) {
        // first call to fib:
        // construct a lookuptable with at least 2 spaces
        lookUpTable = new long[Math.max(index+1,2)];
        // fill in the base cases
        lookUpTable[0] = 0;
        lookUpTable[1] = 1;
    }
    // the base case now changed to: "It has been computed already"
    if (index > 1 && lookUpTable[index] == 0)
        lookUpTable[index] = fib(index-1,lookUpTable)
                             + fib(index-2,lookUpTable);

    // now the correct value is in the array
    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

Other examples - tattarrattat

- civic
- mom
- radar
- rotavator

- eva, can I see bees in a cave?
- never odd or even

- Tip, el pastor ara farà rots a ple pit

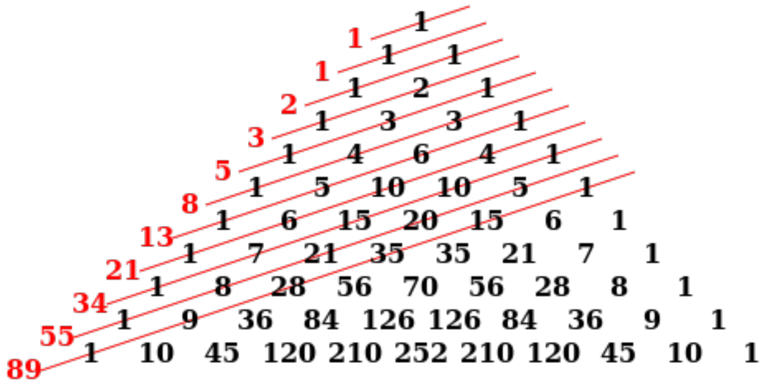
Other examples - tattarrattat

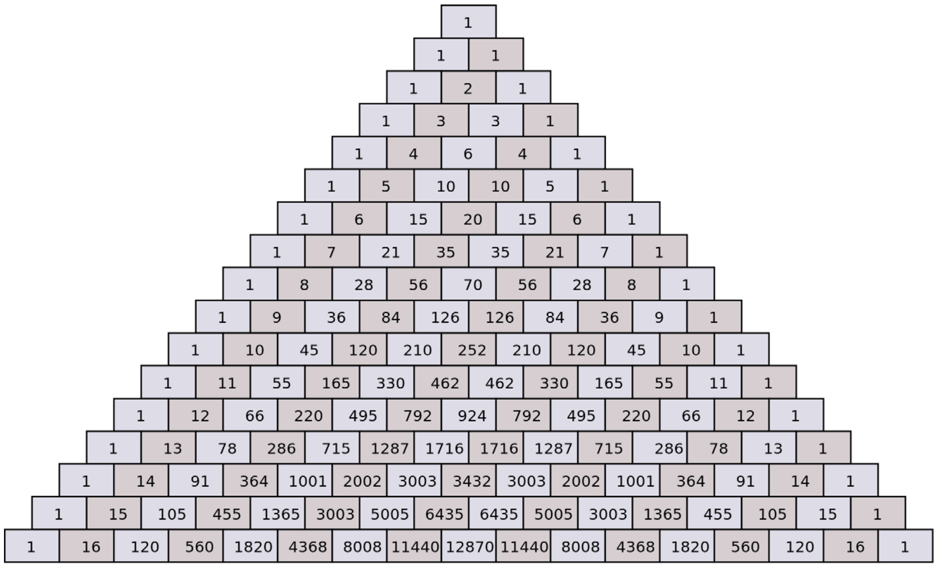
```java
public class Main {
  public static boolean isPalindrome (String str) {
    if(str.length() <= 1)
      return true;
    if(str.charAt(0) != str.charAt(str.length()-1))
      return false;
    return isPalindrome(str.substring(1, str.length()-1));
  }

  public static void main(String[] args) {
    System.out.println(isPalindrome("tattarrattat"));
  }
}
```

Example to practise – Pascal triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1





```java
public class Main {
    static int factorial(int n) {
        int f;
        for(f = 1; n > 1; n--)
            f *= n;
        return f; }

    static int ncr(int n, int r) {
        return factorial(n) / ( factorial(n-r) * factorial(r) ); }

    public static void main(String args[]){
        int n, i, j;
        n = 5;
        for(i = 0; i <= n; i++) {
            for(j = 0; j <= n-i; j++)
                System.out.print(" ");
            for(j = 0; j <= i; j++)
                System.out.print(" "+ncr(i, j));
            System.out.println();
        }
    }
}
```

$$\frac{n!}{(n-r)! * r!}$$

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5  10  10   5   1
```

Example to practice – Find minimum

- Base case:
  - There is just one element (it is considered as the minimum)

- Recursive case:
  - Calculate the minimum between the last element and the rest

Math.min(A[n], A[n-1], …, A[1], A[0])

Bonus track: Collatz conjecture

$$\left.\begin{array}{ll} \text{x (even):} & \frac{x}{2} \\ \text{x (odd):} & 3x+1 \end{array}\right\} 1$$

23
70
35
106
53
160
80
40
20
10
5
16
8
4
2
1

Recursive solution

```java
public static int recursiveSeries (int x) {
  System.out.print(x + " ");

  if (x == 1)
    return 0;
  else
    if (x%2 == 0)
      return recursiveSeries(x/2) + 1;
    else
      return recursiveSeries(3*x+1) + 1;
}
int number = 23;
int result = recursiveSeries (number);
System.out.println();
System.out.println("This took " + result + " steps.");
```

Overview

- Base case + recursive calls
- From for/while to recursion
- Mutual recursion
- Backtracking search
- Efficiency
  - Tabling
- Examples