PROGRAMMING AND DATA STRUCTURES

RECURSION

HOURIA OUDGHIRI

SPRING 2023

OUTLINE

- Recursion
- Recursive methods and helper methods
- Recursion vs. Iteration
- Examples of recursion

STUDENT LEARNING OUTCOMES

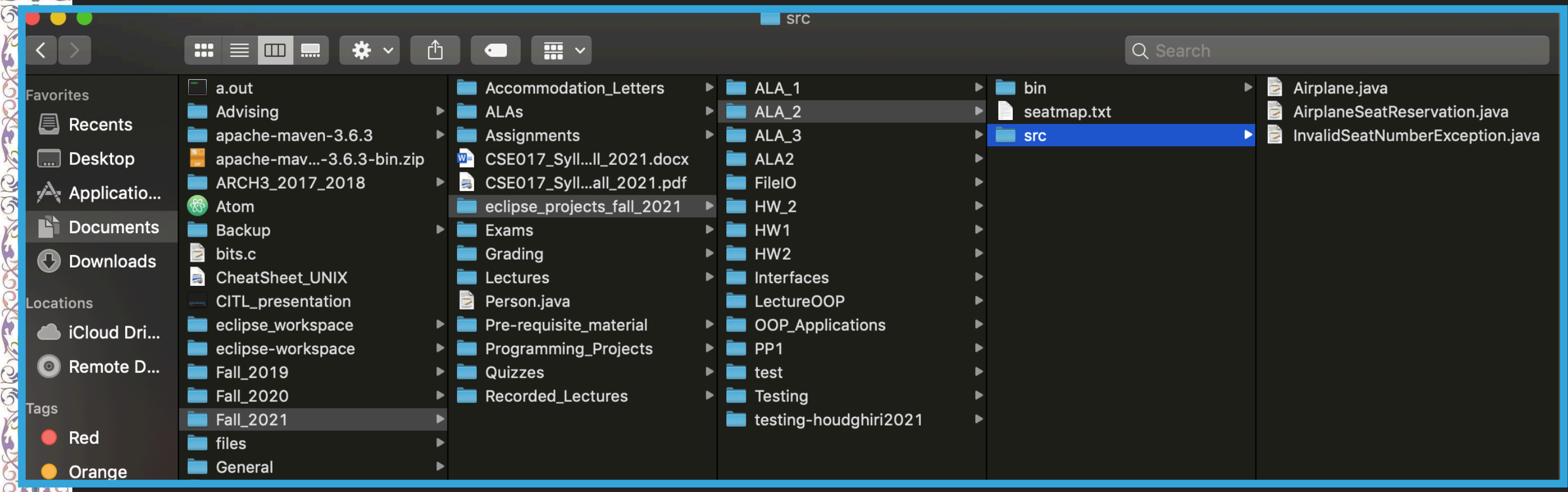
At the end of this chapter, you should be able to:

- Explain the concept of recursion and recursive methods
- Trace recursive method execution
- Write and test recursive methods
- Compare iterative and recursive methods

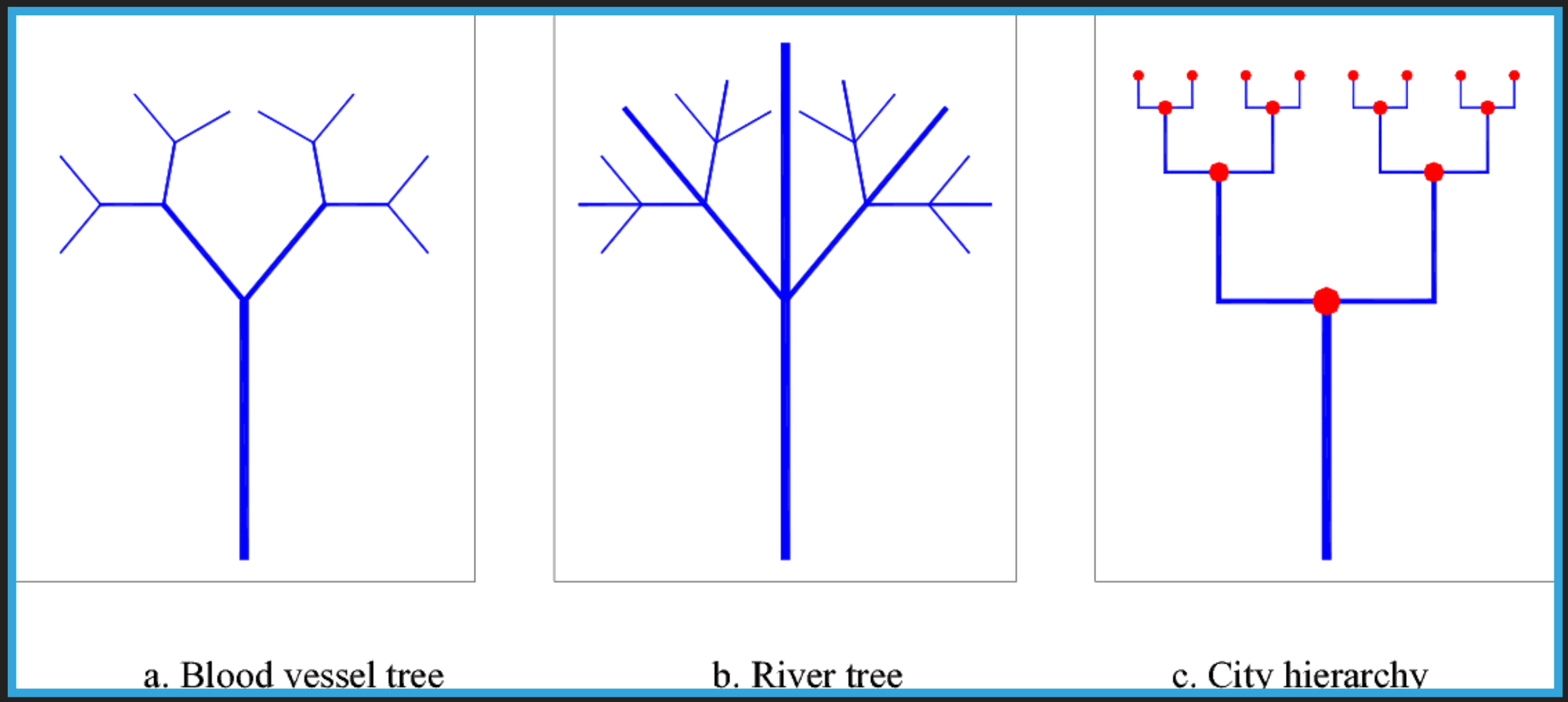
RECURSION

- Recursion is an algorithmic paradigm to solve iterative problems that are difficult to solve using loops
- Recursive mathematical series
- Exploring hierarchical or tree structures

File Hierarchy

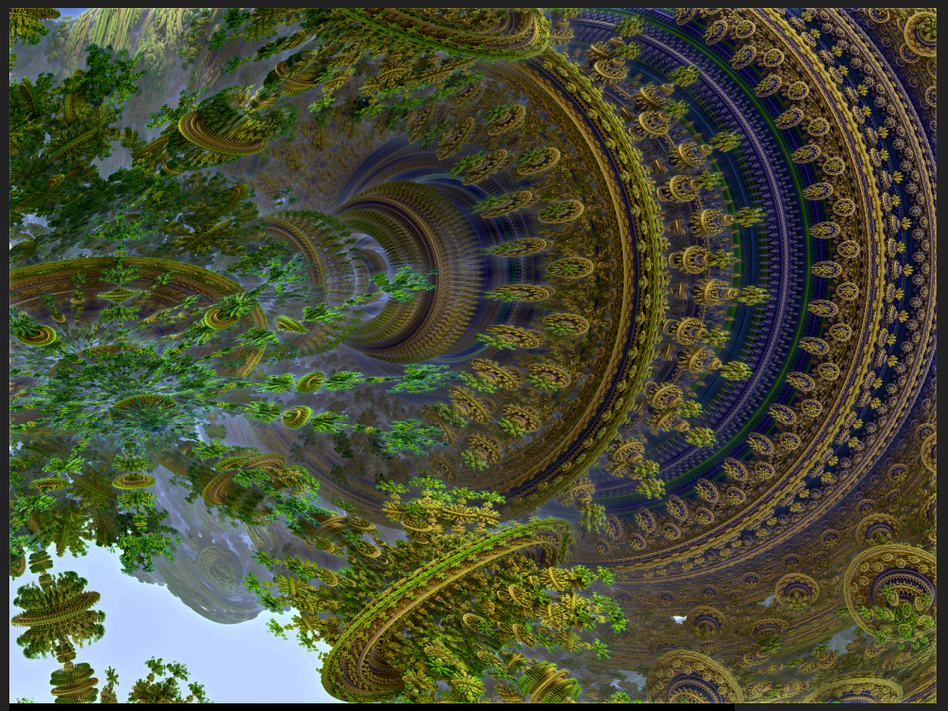


Drawing or exploring tree structures



https://www.semanticscholar.org/paper/Fractals-and-fractal-dimension-of-systems-of-blood-Chen/

Fractals



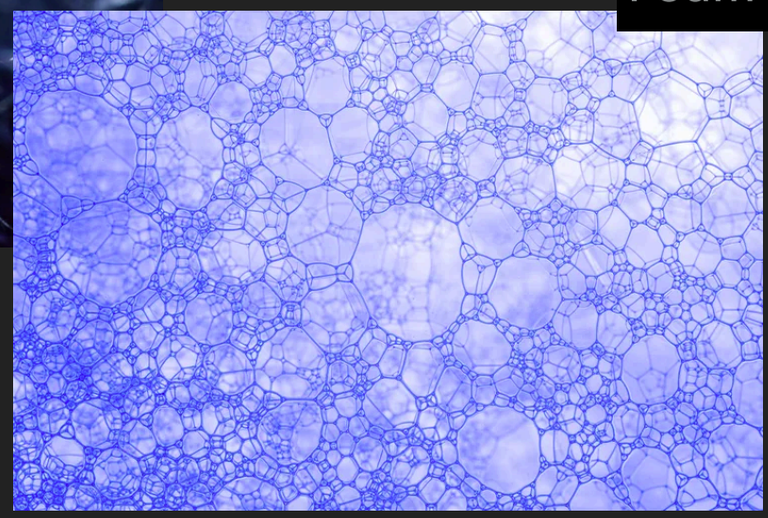
https://commons.wikimedia.org

Trees

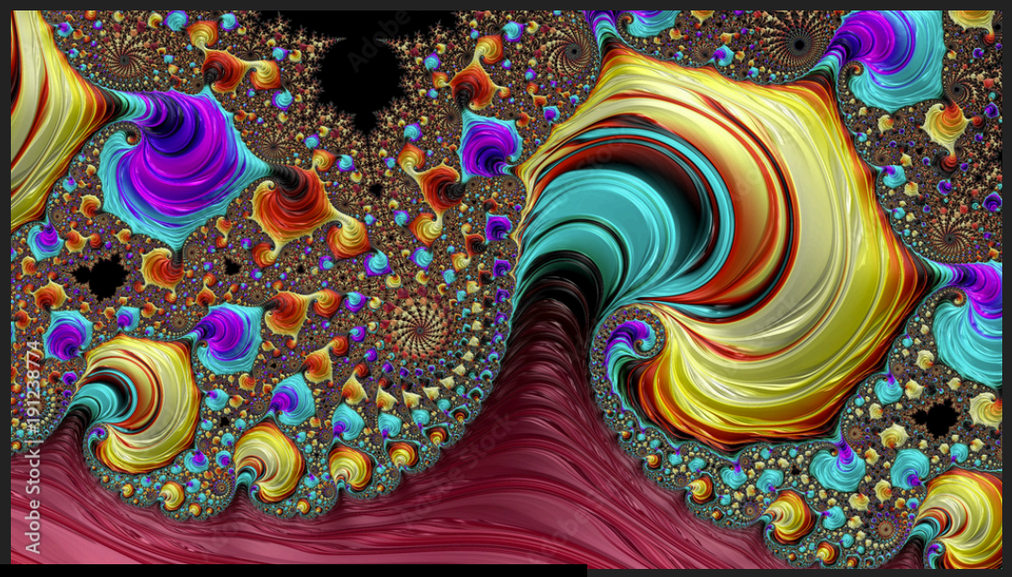
Broccoli

Ice and Snow

Foam



https://www.treehugger.com



https://stock.adobe.com

Recursive methods

- A method that calls itself
- Example: Recursive series in mathematics (Calculating Factorial)

```
n! = n x (n-1)!   1! = 0! = 1
5! = 5 x 4!
4! = 4 x 3!
3! = 3 x 2!
2! = 2 x 1!
1! = 1
```

Recursive methods

```
public static int factorial(int n){
    int fact = 1;
    for(int i = 2; i <= n; i++){
        fact = i * fact;
    }
    return fact;
}
```

Iterative Factorial

Recursive methods

Iterative Factorial

```
public static int factorial(int n){
    int fact = 1;
    for(int i = 2; i <= n; i++){
        fact = i * fact;
    }
    return fact;
  }
```

Recursive Factorial

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

Recursive methods

```
public static void main(String[] args)
{
 int N = 5;
 int f = factorial(N);
}
```

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

5

Recursive methods

```
int rFactorial(int n){          n = 5
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

```
int rFactorial(int n){          n = 4
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

```
int rFactorial(int n){          n = 3
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

```
int rFactorial(int n){          n = 2
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

```
int rFactorial(int n){          n = 1
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

Recursive methods

```
int rFactorial(int n){
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

n = 5

```
int rFactorial(int n){
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

n = 4

24(4*6)

```
int rFactorial(int n){
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

n = 3

6 (3* 2)

```
int rFactorial(int n){
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

n = 2

2 (2 * 1)

```
int rFactorial(int n){
 if (n == 1 || n == 0)
   return 1;
 else
   return n *rFactorial(n-1);}
```

n = 1

1

Recursive methods

```
static void main(String[] args){
 int N = 5;
 int f = rFactorial(N);
}
```

120

```
int rFactorial(int n){ n = 5
 if (n == 1 || n == 0)
   return 1;
 else
   return n * rFactorial(n-1);}
```

5

5*24

Recursive methods

```
int rFactorial(int n){
  if (n == 1 || n == 0)
    return 1;
  else
    return n * rFactorial(n-1);
}
```

Base Case

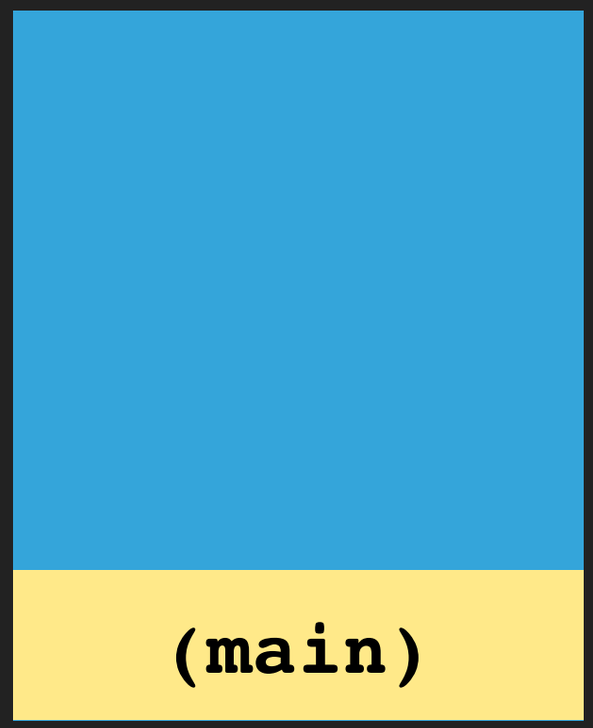
Stopping Case

Recursion

- Base Case or Stopping Case must be present
- Infinite recursion if there is no base case
- Stack overflow

Recursive methods and the Stack

```
public class Stack{
    public static void method2(){
        // some code here
        return;
    }
    public static void method1(){
        // some code here
        method2();
        // some code here
        return;
    }
    public static void main(String[] args){
        // some code here
        method1();
        // some code here
        return;
    }
}
```

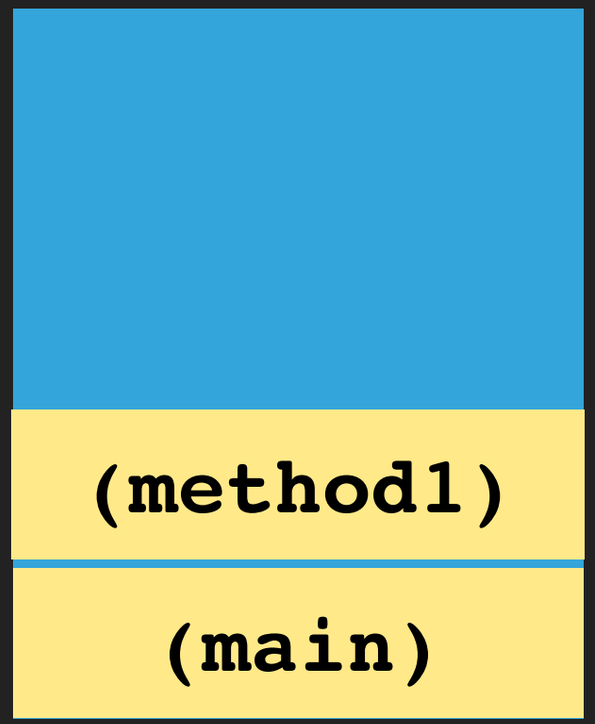


Stack

main is the top of the stack

Recursive methods and the Stack

```
public class Stack{
    public static void method2(){
        // some code here
        return;
    }
    public static void method1(){
        // some code here
        method2();
        // some code here
        return;
    }
    public static void main(String[] args){
        // some code here
        method1();
        // some code here
        return;
    }
}
```

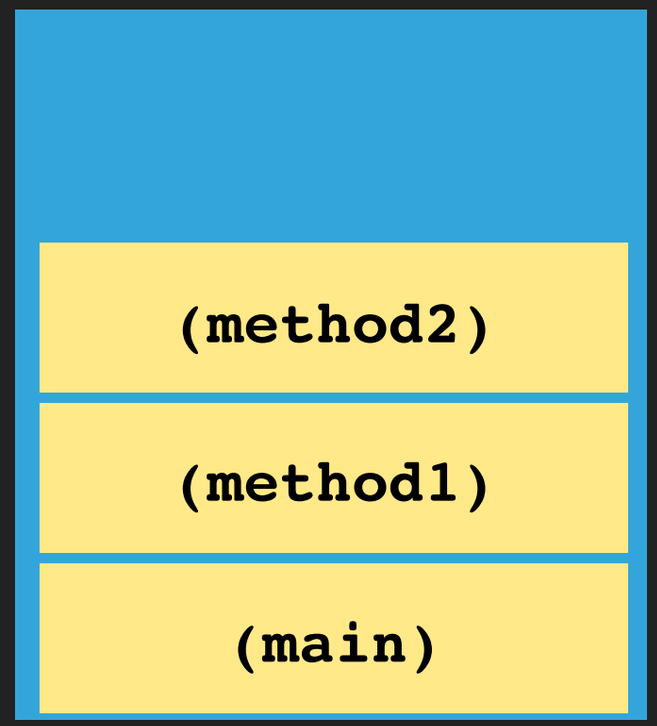


Stack

method1 is the top of the stack

Recursive methods and the Stack

```
public class Stack{
    public static void method2(){
        // some code here
        return;
    }
    public static void method1(){
        // some code here
        method2();
        // some code here
        return;
    }
    public static void main(String[] args){
        // some code here
        method1();
        // some code here
        return;
    }
}
```



Stack

Method2 is the top of the stack

Recursive methods and the Stack

```
public class Stack{
    public static void method2(){
        // some code here
        return;
    }
    public static void method1(){
        // some code here
        method2();
        // some code here
        return;
    }
    public static void main(String[] args){
        // some code here
        method1();
        // some code here
        return;
    }
}
```

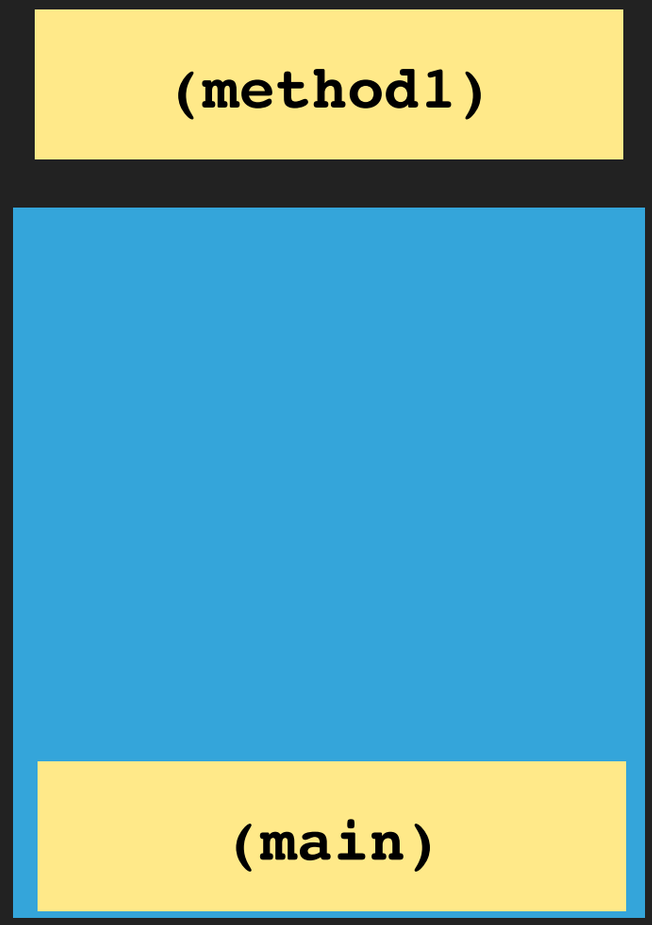
Stack

Method2 removed from the stack

method1 at the top of the stack

Recursive methods and the Stack

```
public class Stack{
    public static void method2(){
        // some code here
        return;
    }
    public static void method1(){
        // some code here
        method2();
        // some code here
        return;
    }
    public static void main(String[] args){
        // some code here
        method1();
        // some code here
        return;
    }
}
```



Stack

Method1 removed from the stack

main at the top of the stack

Recursive methods and the Stack

- Stack overflow - The stack is full
- The number of calls exceeds the size of the stack
- Recursion without base case - infinite number of calls

Recursive methods

Practice: What is the output of the following code?

```
public static void xMethod1(int n){
    if(n > 0){
        System.out.print(n + " “); 5 4 3 2 1
        xMethod1(n-1);
    }
}
public static void xMethod2(int n){
    if(n > 0){
        xMethod2(n-1); xM2(4), xM2(3), xM2(2), xM2(1),
        System.out.print(n + " “); 1 2 3 4 5
    }
}
public static void main(String[] args) {
    xMethod1(5);
    System.out.println();
    xMethod2(5);
}
```

Recursive methods

Practice: What is wrong with the following code?

```
public static void xMethod(double n){
    if(n != 0){
        System.out.println(n + " ");
        xMethod(n/10);
    }
}
public static void main(String[] args) {
    xMethod(1000);
}
```

Recursive methods

- Write a recursive method to compute the following series and test it for $m = 10$

```
double series(int m)
```

$$series(m) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \ldots + \frac{m}{2m+1}$$

Recursive methods

```
int series(int m){

 if (m == 1)

   return 1/3.0;

 else

   return m/(2*m+1) + series(m-1);

}
```

1

Base Case will be reached, m is decremented

2

Base case returns $\frac{1}{3}$ = m(1)

3

Recursion returns

$$\frac{m}{2m+1} + series(m-1)$$

Recursive helper methods

- Writing a recursive Binary search
- Binary Search
- Finding a key in an array of ordered numbers
- A problem that is inherently recursive

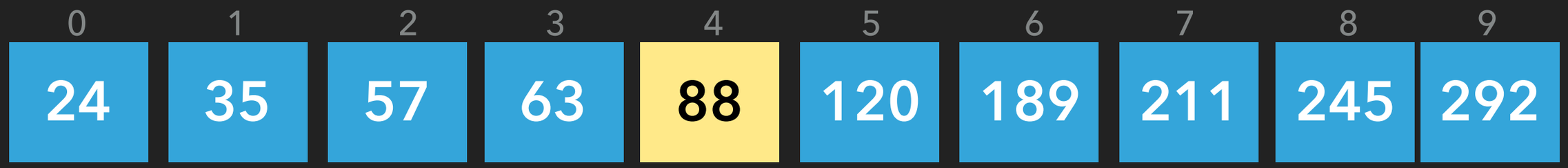
Recursive helper methods

- Finding a key (number) in an ordered list of numbers
- Divide the list in halves

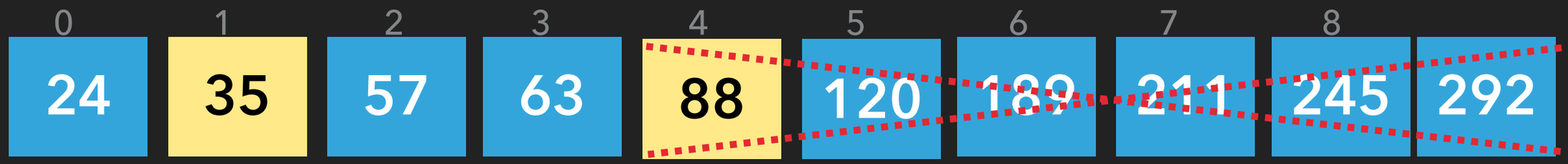
```
List = {24, 35, 57, 63, 88, 120,
        189, 211, 245, 292}
Key = 35 or key = 200
```

Recursive helper methods

- **Key = 35**



Key < 88



Key = 35

Key = 35, Found at index 1

Recursive helper methods

- **Key = 200**

Key > 88

0	1	2	3	4	5	6	7	8	9
24	35	57	63	88	120	189	211	245	292

0	1	2	3	4	5	6	7	8	9
24	35	57	63	88	120	189	211	245	292

Key < 211

0	1	2	3	4	5	6	7	8	9
24	35	57	63	88	120	189	211	245	292

Key > 120

0	1	2	3	4	5	6	7	8	9
24	35	57	63	88	120	189	211	245	292

Key > 189, Not found

Recursive helper methods

```
public static int binarySearch(int[] list, int key){
    int first, last, middle;
    first = 0;
    last = list.length-1;
    while (first <= last){
        middle = (last + first) / 2;
        if (key == list[middle])
            return middle;
        else if (key < list[middle])
            last = middle - 1;
        else
            first = middle + 1;
    }
    return -1;
}
```

Iterative Binary Search

Recursive helper methods

```
public static int binarySearch(int[] list, int key){
    int first = 0;
    int last = list.length-1;
    return binarySearch(list, key, first, last);
}
```

Helper Method that is recursive and accepts additional parameters

Recursive helper methods

```
public static int binarySearch(int[] list, int key,
                               int first, int last){
    if (first > last)
        return -1;
    else{
        int middle = (last + first) / 2;
        if (key == list[middle])
            return middle;
        else if (key < list[middle])
            last = middle - 1;
        else
            first = middle + 1;
        return binarySearch(list, key, first, last);
    }
}
```

Recursive Binary Search Helper Method

Recursion vs. Iteration

- Recursion usually requires less code
- Recursion reflects the divide-and-conquer strategy for solving a problem
- Recursion requires consecutive calls to the same function (stack push/pop operations)
- Iterations are preferred. They are more efficient (computationally)

Recursion vs. Iteration

Fibonacci sequence

- Used to model many natural phenomena (rabbit population growth, petal structure in a flower, pine cones, financial market analysis)

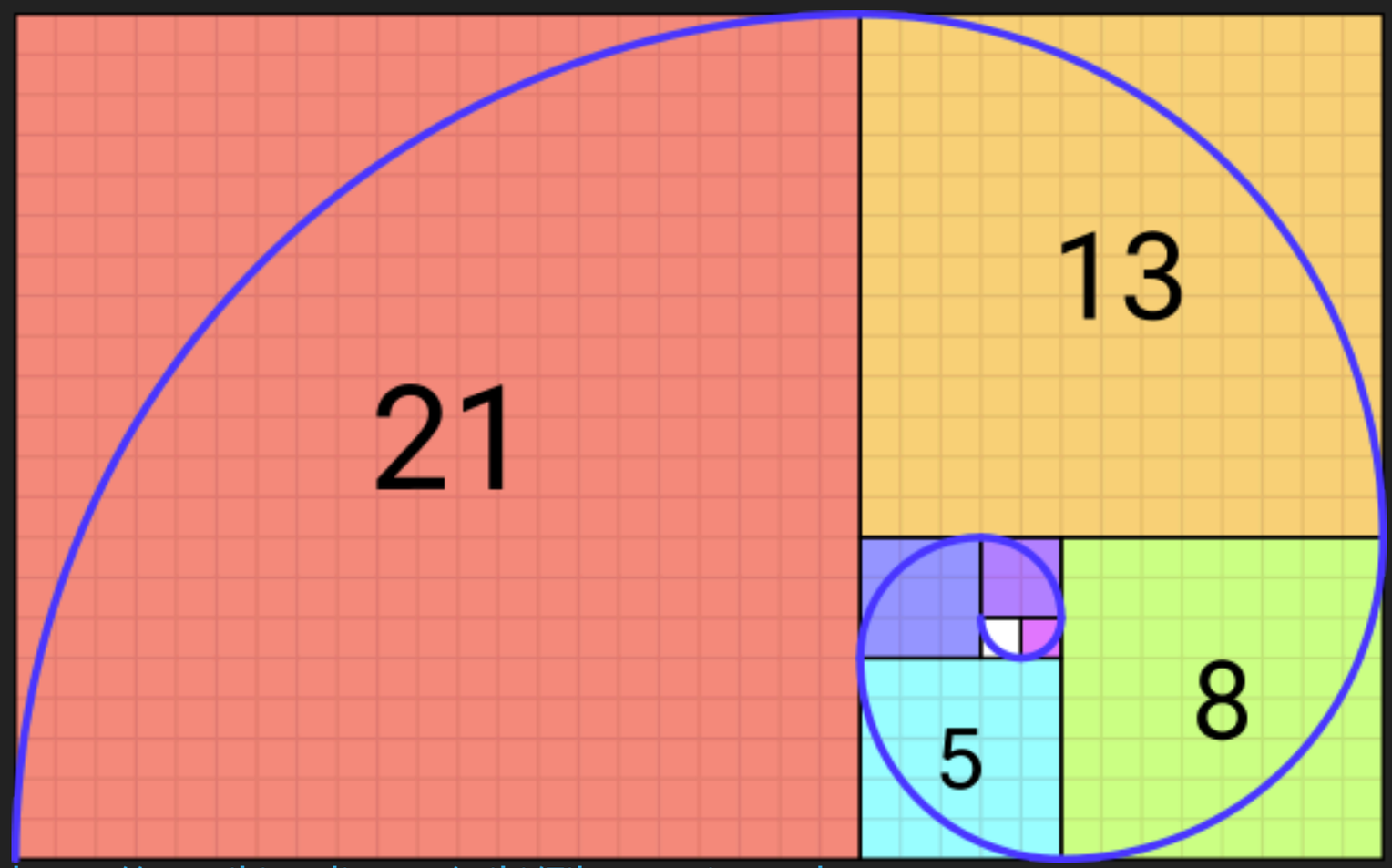
$$F_n = F_{n-1} + F_{n-2}, n > 0$$

$$F_2 = 1, F_1 = 1$$

- 1 1 2 3 5 8 13 21 ($f_8 = 21$)

Recursion vs. Iteration

Fibonacci sequence



https://en.wikipedia.org/wiki/Fibonacci_number

Recursion vs. Iteration

✦ Fibonacci Series

```
public static int fibonacci(int n){
        int f1=1, f2=1, f=0;
        if (n <= 2)
            return 1;
        else{
            while (n > 0){
                f = f1 + f2;
                f1 = f2;
                f2 = f;
                n = n -1;
            }
        }
        return f;
    }
```

Iterative Fibonacci

Recursion vs. Iteration

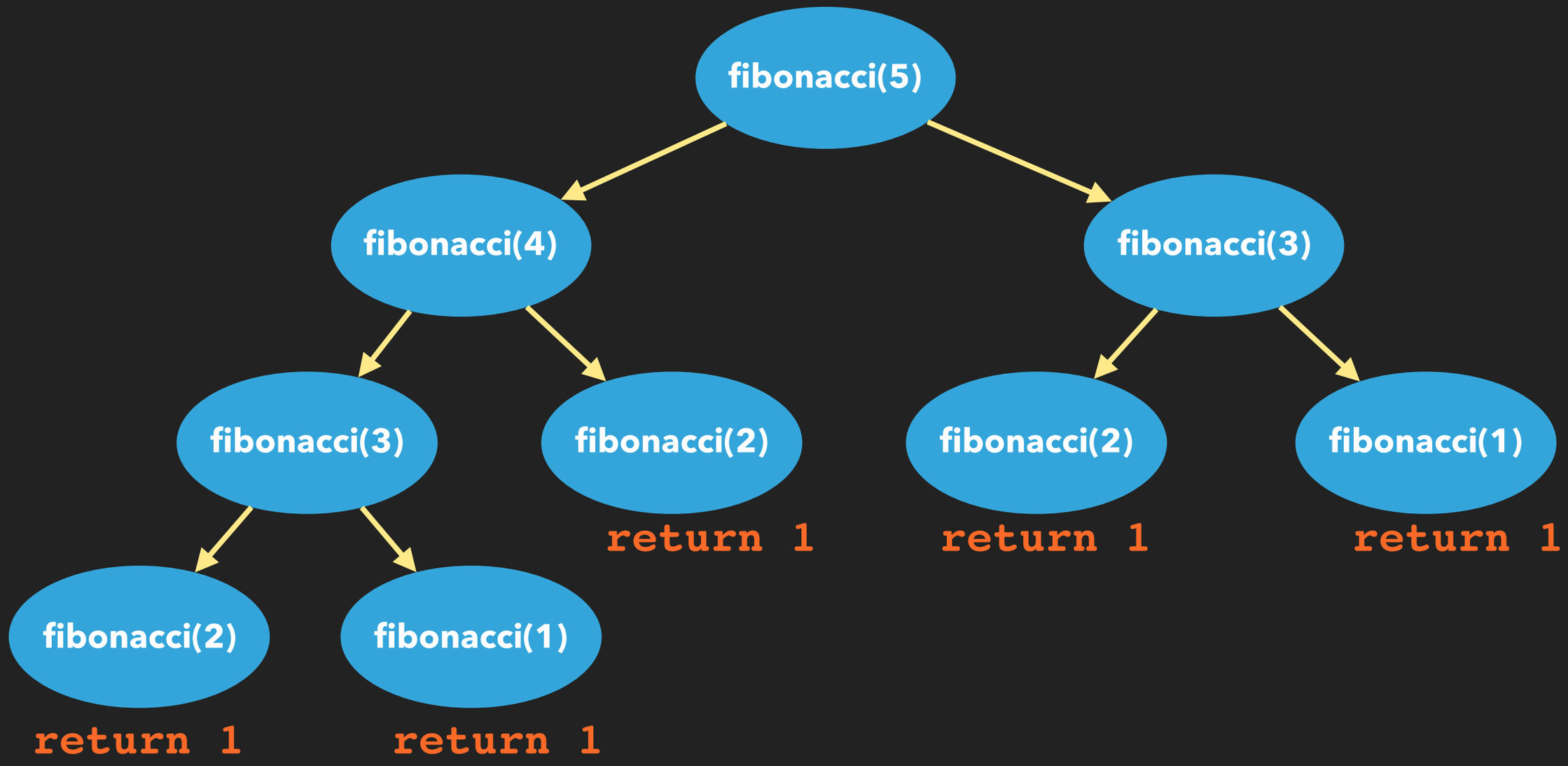
✦ Fibonacci Series

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

Recursive Fibonacci

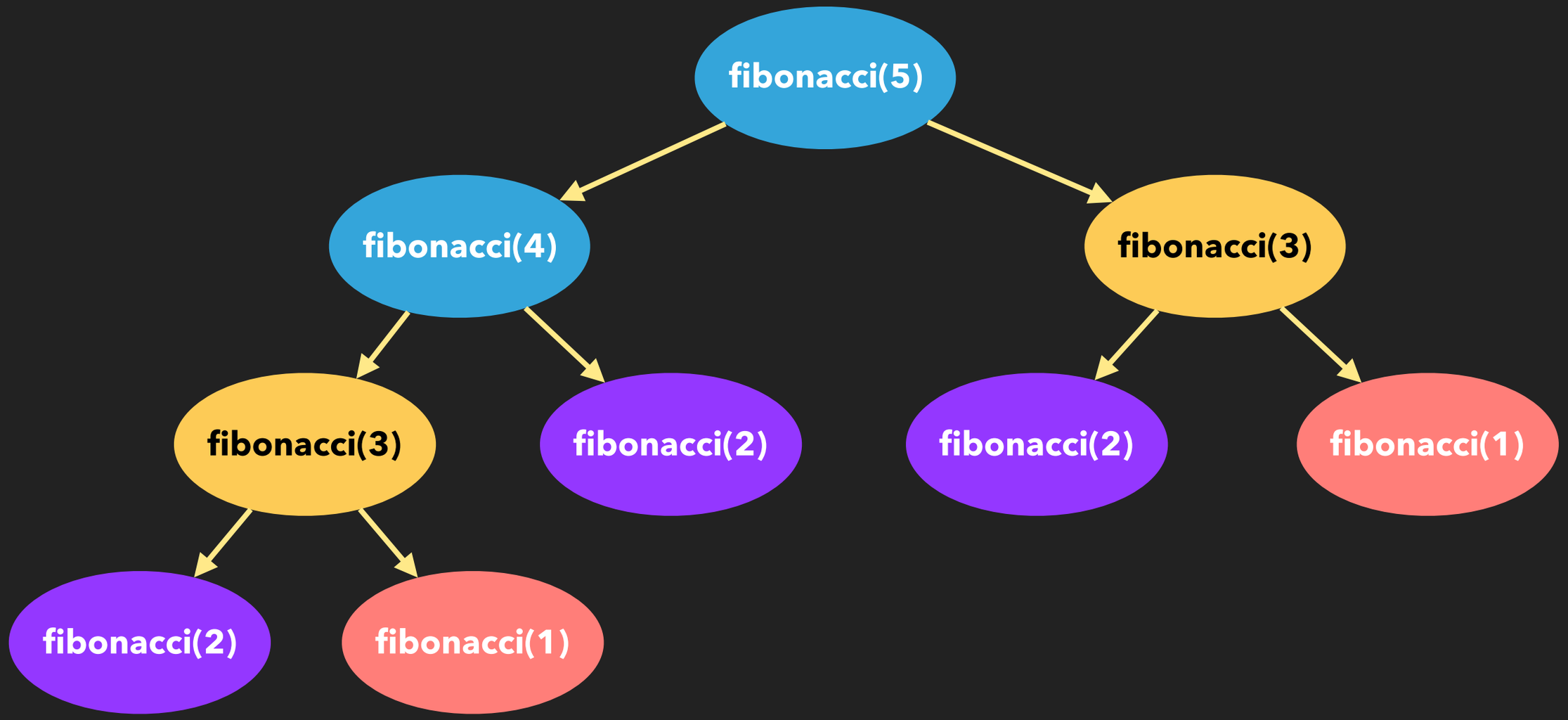
Recursion vs. Iteration

- Fibonacci Series - **`fibonacci(5)`**



Recursion vs. Iteration

- Fibonacci Series - **`fibonacci(5)`**



Recursion vs. Iteration

- Recursion usually results in more compact code than iteration
- Recursion may be computationally less efficient than iteration
- Use recursion only when the problem is hard to solve using loops

Recursion - Practice

Write an iterative and a recursive version of the method

```
public static boolean isPalindrome(String s)
```

that returns **`true`** is **`s`** is a palindrome, **`false`** otherwise

Examples of palindrome strings: **`radar`** , **`mom`**, **`dad`**, **`kayak`**

Summary

- Recursive method - calls itself
- Base case and recursive case - finite number of recursive calls
- Recursion vs Iteration - Iteration is more efficient if you can use it
- Use helper recursive methods when you need to change the signature of the method