

Recursive Algorithms

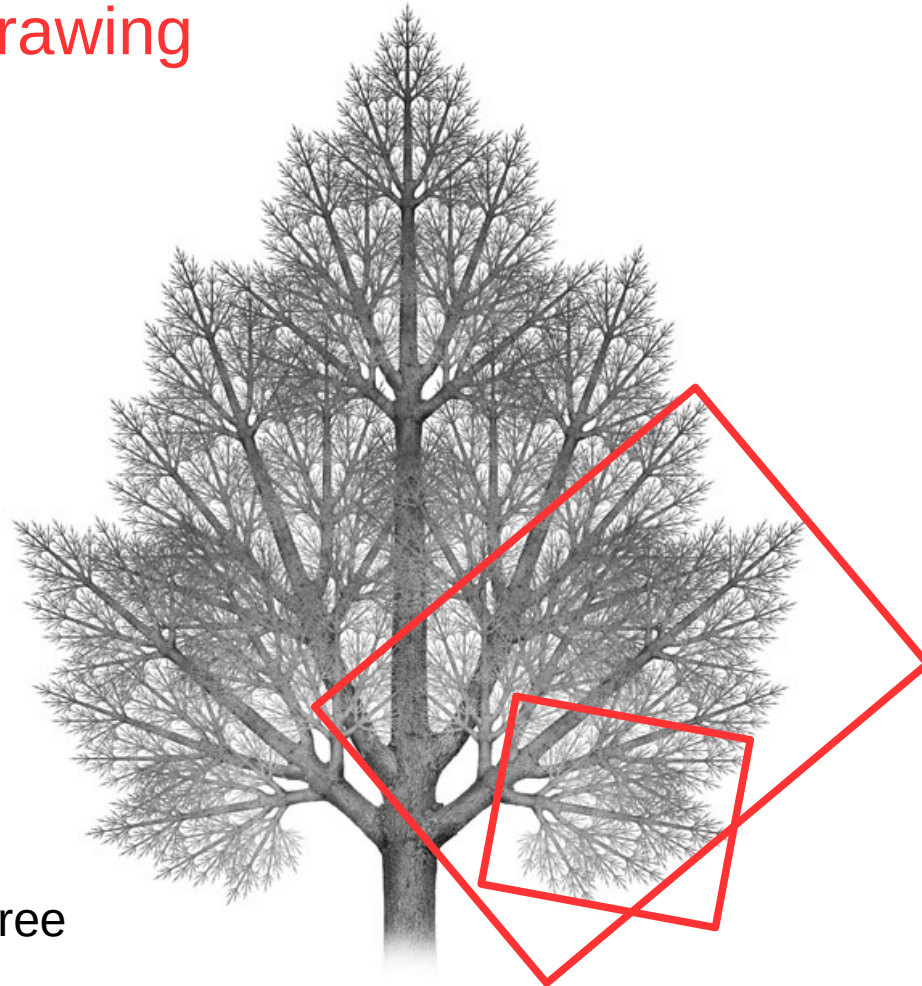
Hendrik Speleers

Recursive Algorithms

- **Overview**
 - Recursive calls
 - Proof of correctness + complexity
 - Examples
 - Greatest common divisor, power, binary search
 - Tower of Hanoi
 - Sorting: quick sort

Recursive Algorithms

- A recursive drawing



A fractal tree



Recursive Algorithms

- **A recursive algorithm**
 - Base case: result is determined explicitly
 - Recursive case: result is expressed in terms of same (but reduced) problem
 - The base case should be reached after a finite number of iterations
- **Recursive calls**
 - Direct recursion
 - A method $f()$ of object A sends a message $f()$ to object A
 - Indirect recursion
 - A method $f()$ of object A sends a message $g()$ to object A
 - And the method $g()$ of object A sends a message $f()$ to object A



Recursive Algorithms

- Examples
 - Greatest common divisor (gcd) and power, version 3

```
public int gcd(int x, int y) { // pre: x > 0 && y > 0
    if (x == y) return x;
    else if (x > y) return gcd(x - y, y);
    else return gcd(x, y - x);
}
```

```
public int pow(int x, int n) { // pre: n >= 0
    if (n == 0) return 1;
    else if (n % 2 == 0) return pow(x * x, n / 2);
    else return x * pow(x, n - 1);
}
```

Recursive Algorithms

- Examples
 - Binary search: value v in sorted array *list* ?

```
public boolean binarySearch() { // pre: sorted list
    return binarySearchRec(0, list.length-1);
}

private boolean binarySearchRec(int a, int b) {
    // pre: sorted list && a <= b
    if (a == b) return (list[a] == v);
    else {
        int m = (a + b) / 2;
        if (v <= list[m]) return binarySearchRec(a, m);
        else return binarySearchRec(m+1, b);
    }
}
```

Recursive Algorithms

- **Proof of correctness**
 - What we have seen before: iterative algorithm
 - Specify precondition and postcondition (result)
 - Specify invariant
 - Prove invariant is satisfied initially
 - Prove invariant is satisfied after each iteration
 - Prove “invariant && !condition” leads to postcondition
 - Termination in finite number of iterations
 - Recursive algorithm

Recursive Algorithms

- **Proof of correctness**
 - What we have seen before: iterative algorithm
 - Recursive algorithm
 - Specify precondition and postcondition (result)
 - Prove postcondition is satisfied for base case
 - Prove precondition is satisfied for recursive case
 - Prove “precondition && postcondition of base case” leads to postcondition
 - Termination in finite number of iterations
 - Example: power, version 3

Recursive Algorithms

- Proof of correctness

```
public int pow(int x, int n) {  
    // pre: n >= 0  
    // res: result == power(x,n)  
    if (n == 0) {  
        return 1;  
  
    } else if (n % 2 == 0) {  
        return pow(x * x, n / 2);  
  
    } else {  
        return x * pow(x, n - 1);  
    }  
}
```

Recursive Algorithms

- Proof of correctness

```
public int pow(int x, int n) {  
    // pre: n >= 0  
    // res: result == power(x,n)  
    if (n == 0) {  
        return 1;  
        // n == 0 && result == 1 == power(x,n)  
    } else if (n % 2 == 0) { // n >= 0  
        return pow(x * x, n / 2);  
        // result == power(x*x,n/2) == power(x,n)  
    } else { // n >= 1  
        return x * pow(x, n - 1);  
        // result == x*power(x,n-1) == power(x,n)  
    }  
}
```

Recursive Algorithms

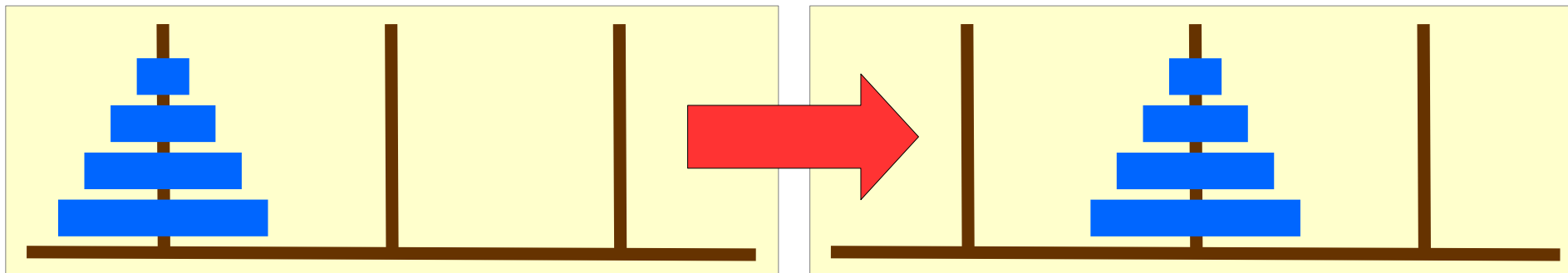
- Complexity
 - Example: power, version 3
 - Complexity: counting most time-consuming operations
 - Sending a message “pow” (SM_n for given n)
 - Number of messages
 - $SM_0 = 1$
 - $SM_{2m} = 1 + SM_m$
 - $SM_{2m+1} = 1 + SM_{2m} = 2 + SM_m$
 - $SM_n = O(\log(n))$

Recursive Algorithms

- To recurse or not to recurse ...
 - So far, examples that were easy to implement as
 - Iterative algorithm
 - Recursive algorithm
 - Actually, these algorithms are called “tail recursive”
 - The last statement in the algorithm is to “restart” the algorithm
 - Tail recursive algorithms can be directly translated into loops
 - Example: Tower of Hanoi

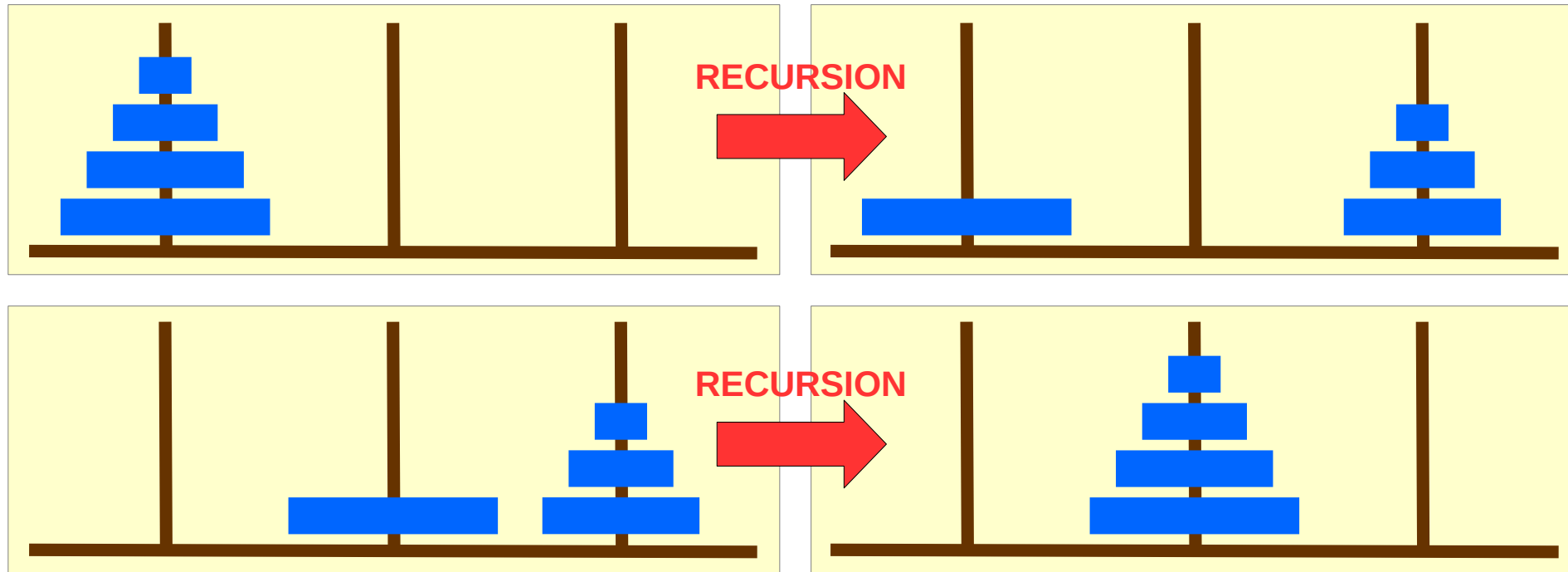
Recursive Algorithms

- To recurse or not to recurse ...
 - Example: Tower of Hanoi
 - Objective is to move a stack of disks from one rod to another rod
 - Rules
 - One disk can be moved at a time
 - Only the upper disk from a stack can be moved on top of another stack
 - No disk may be placed on top of a smaller disk



Recursive Algorithms

- To recurse or not to recurse ...
 - Example: Tower of Hanoi



Recursive Algorithms

- To recurse or not to recurse ...
 - Example: Tower of Hanoi
 - Recursion ?
 - Elegant solution
 - Iterative version is more cumbersome
 - Write a Java program with output

```
Move disk from 1 to 3  
Move disk from 1 to 2  
Move disk from 3 to 2  
Move disk from 1 to 3  
...
```

Recursive Algorithms

- To recurse or not to recurse ...
 - Example: Tower of Hanoi

```
public void hanoi(int n, int from, int to) {  
    if (n == 1) move(from, to);  
    else {  
        int third = 6 - from - to;  
        hanoi(n - 1, from, third);  
        move(from, to);  
        hanoi(n - 1, third, to);  
    }  
}  
  
private void move(int from, int to) {  
    System.out.printf("Move disk from %d to %d\n",  
        from, to);  
}
```

Recursive Algorithms

- To recurse or not to recurse ...
 - Example: Tower of Hanoi

Output of hanoi(4, 1, 2)

Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 3
Move disk from 1 to 3
Move disk from 1 to 2

Move disk from 3 to 2
Move disk from 3 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2

Recursive Algorithms

- To recurse or not to recurse ...
 - Example: Tower of Hanoi
 - Complexity: counting most time-consuming operations
 - Sending a message “hanoi” (SM_n for given n)
 - Number of messages
 - $SM_1 = 1$
 - $SM_n = 1 + 2 * SM_{n-1}$
 - $SM_n = O(2^n)$

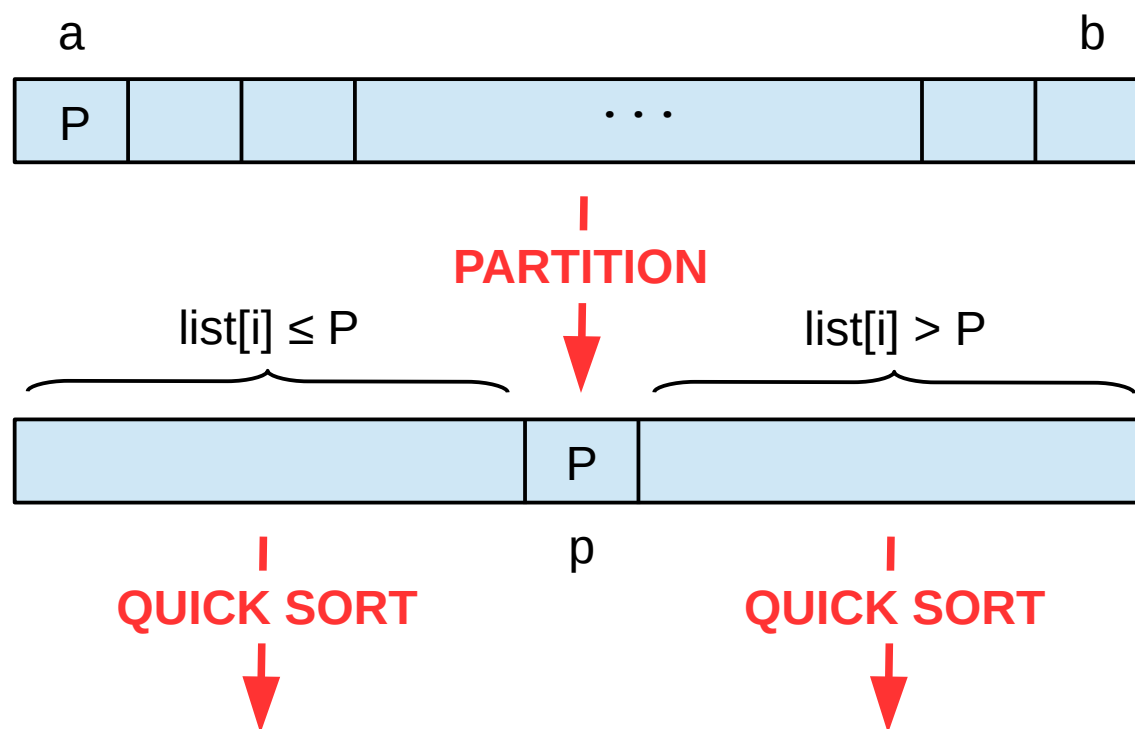
Recursive Algorithms

- **Sorting algorithms**
 - So far, three iterative versions
 - Selection sort
 - Insertion sort
 - Bubble sort (see exercise)
 - Recursive version: quick sort
 - Divide array in two parts
 - Choose random element “pivotelement”
 - Move smaller elements in left part of array
 - Move larger elements in right part of array
 - Apply “quick sort” to both subarrays

} partitioning

Recursive Algorithms

- **Sorting algorithms**
 - Recursive version: quick sort



Recursive Algorithms

- **Sorting algorithms**
 - Recursive version: quick sort

```
public void quickSort() {  
    quickSortRec(0, list.length-1);  
}  
  
private void quickSortRec(int a, int b) {  
    if (a < b) {  
        int p = partition(a, b); // TODO  
        quickSortRec(a, p - 1);  
        quickSortRec(p + 1, b);  
    }  
}
```

Recursive Algorithms

- **Sorting algorithms**
 - Recursive version: quick sort

```
private int partition(int a, int b) {  
    int c = a, d = b, pivot = list[c]; // list[c] free  
    while (c < d) {  
        while (c < d && list[d] > pivot) d--;  
        if (c < d) { // smaller list[d] found -> move  
            list[c] = list[d]; // list[d] free  
        }  
        while (c < d && list[c] <= pivot) c++;  
        if (c < d) // larger list[c] found -> move  
            list[d] = list[c]; // list[c] free  
    }  
    list[c] = pivot; return c;  
}
```