MA2252 Introduction to computing

lectures 9-10

Recursive and sorting algorithms

Matias Ruiz

October 2023

Recursion: first, a question

What does the following function do?

```
function out = myFunction(n)

    if n==1
        out = 1;
    else
        out = n + myFunction(n-1);
    end

end
```

Recursion: the answer

It gives the sum of the first *n* integers.

```
function out = recursiveSum(n)

    if n==1
        out = 1;
    else
        out = n + recursiveSum(n-1);
    end

end
```

This is an example of a *recursive* algorithm

Recursion: definition of a recursive algorithm

A **recursive** function is a function that makes calls to itself.

They are composed of two parts:

- **Base case:** function's value is given or can be calculated without using recursion.
- **Recursive call:** function's value is calculated by calling to itself.

```
function out = recursiveSum(n)

    if n==1
        out = 1; % base case
    else
        out = n + recursiveSum(n-1); %
            recursive call
    end

end
```

Recursion: simple description of complex structures

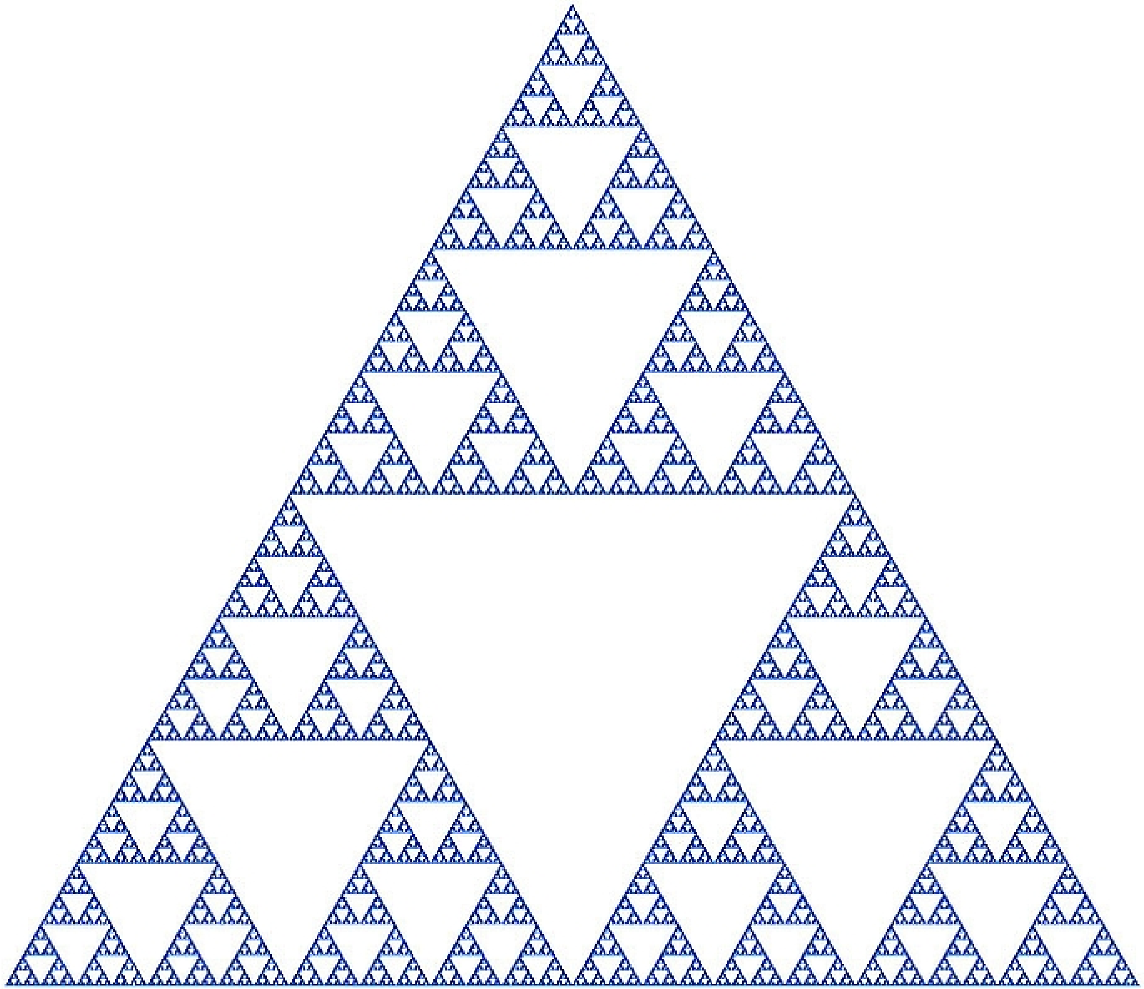


Figure: Sierpinski triangle

Recursion: simple description of complex structures

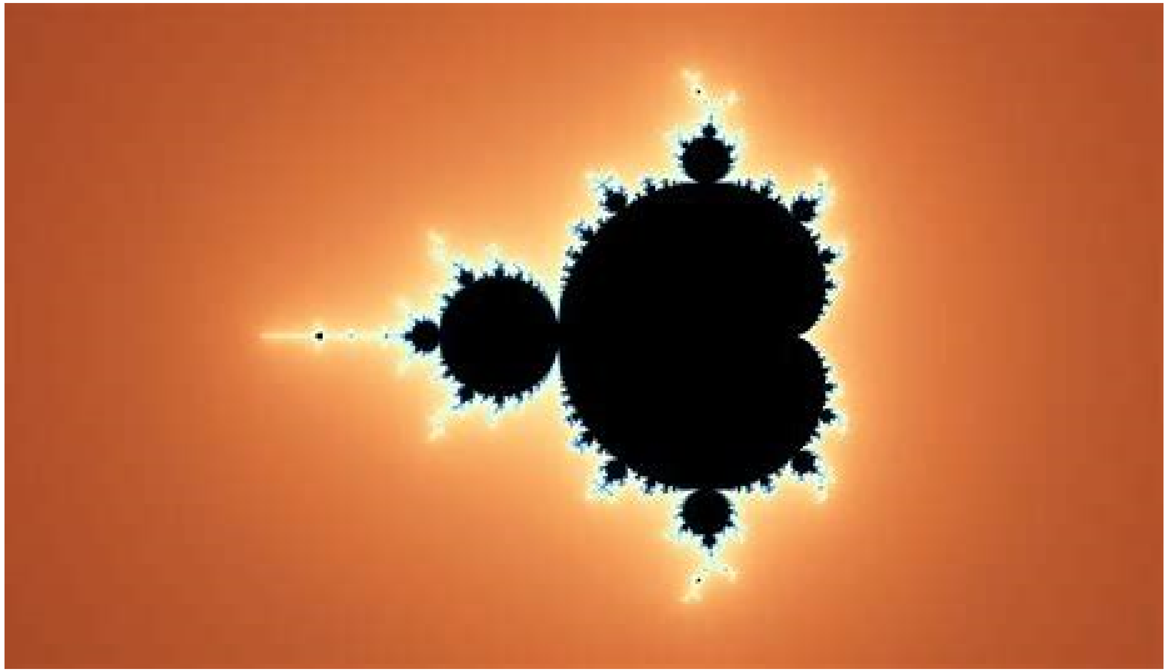
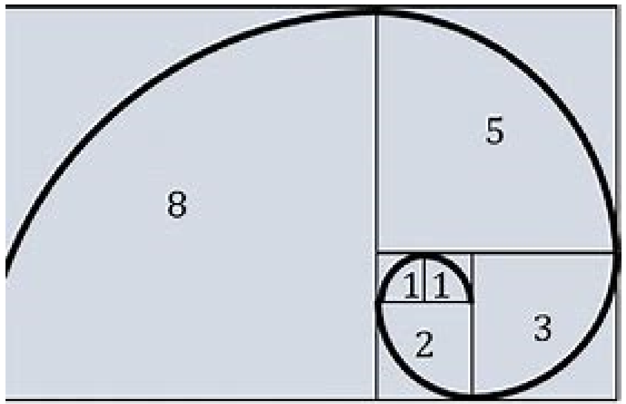


Figure: Mandelbrot set

$$z_{n+1} = z_n^2 + c$$

Recursion: another example – Fibonacci numbers

Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,. . .



$$F(n) = \begin{cases} 1 \text{ if } n = 1, \\ 1 \text{ if } n = 2, \\ F(n-1) + F(n-2) \text{ otherwise.} \end{cases}$$

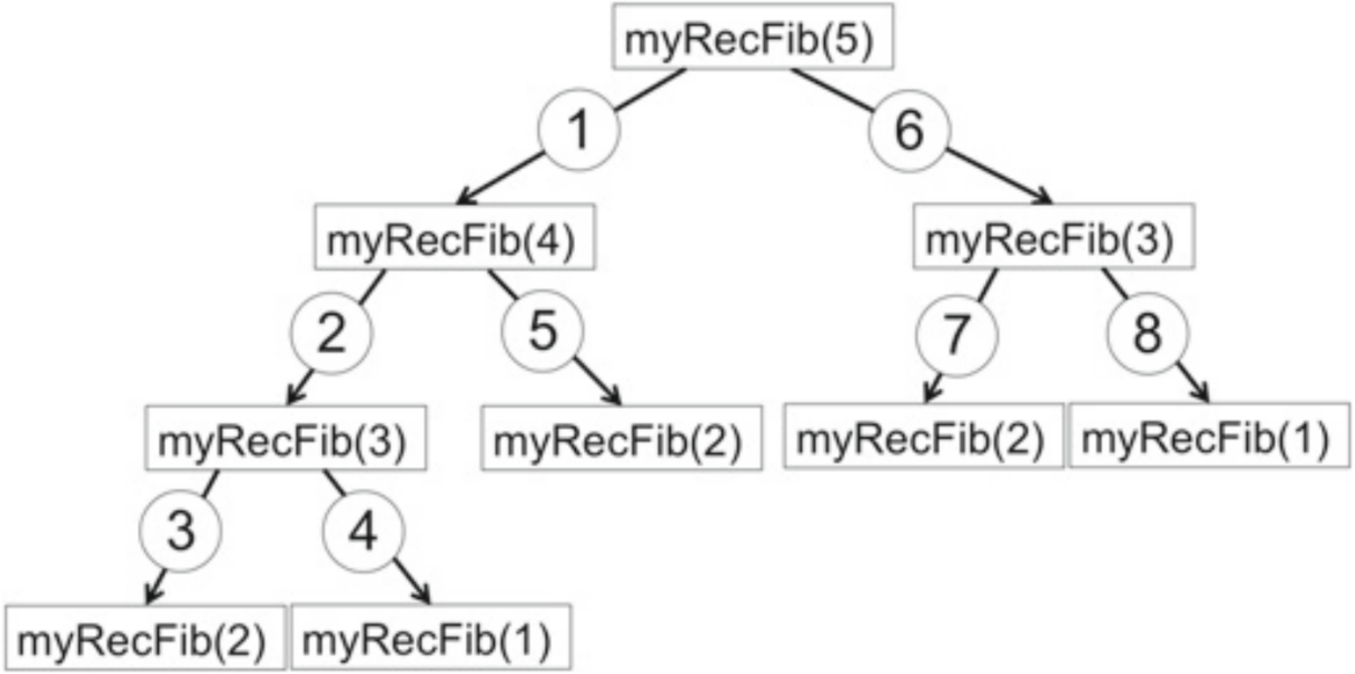
Recursion: recursive Fibonacci function

```
function F = recursiveFibonacci(n)

    if n==1
        F = 1;
    elseif n==2
        F = 1;
    else
        F = recursiveFibonacci(n-1) +
           recursiveFibonacci(n-2);
    end

end
```

Recursion: Fibonacci recursion tree



Recursion: exercise

Write a **recursive** function that given an integer n it outputs $n!$

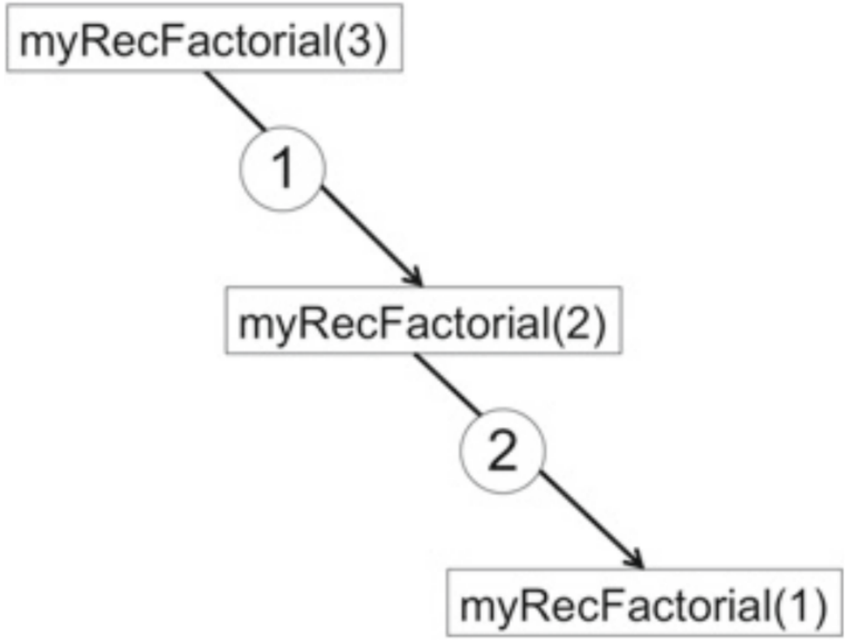
Recursion: the recursive factorial function

```
function out = recursiveFactorial(n)

    if n==0
        out = 1;
    else
        out = n*recursiveFactorial(n-1);
    end

end
```

Recursion: factorial recursion tree



Recursion: the iterative factorial function

```
function out = iterativeFactorial(n)

if n ==0 || n ==1
    out = 1;
else
    fact = n ;
    while n > 1
        fact = fact *(n-1) ;
        n = n-1;
    end
    out = fact ;
end

end
```

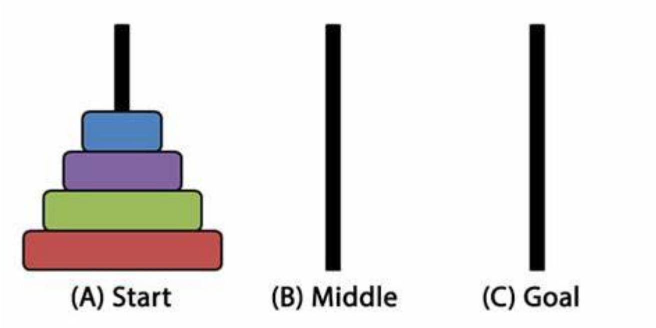
More complex to code, but it is faster to run!

Recursion: disadvantages/advantages

- **Disadvantage:** `MATLAB` opens a new workspace every time a function is called, even for a function calling a function with the same name as itself $\Rightarrow$ Recursive algorithms can run slow and use more memory.
- **Advantage:** Conceptually very useful for solving hard problems: **divide and conquer** approach.

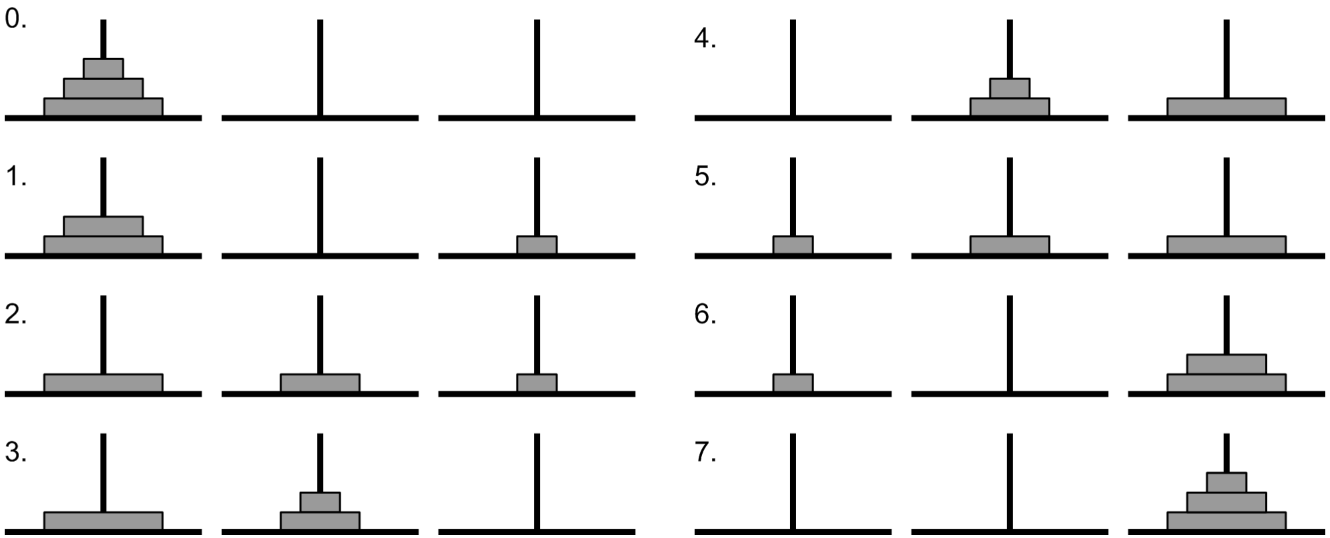
Divide and conquer: difficult problems are broken up to many similar easy, more manageable, problems.

Recursion: Towers of Hanoi



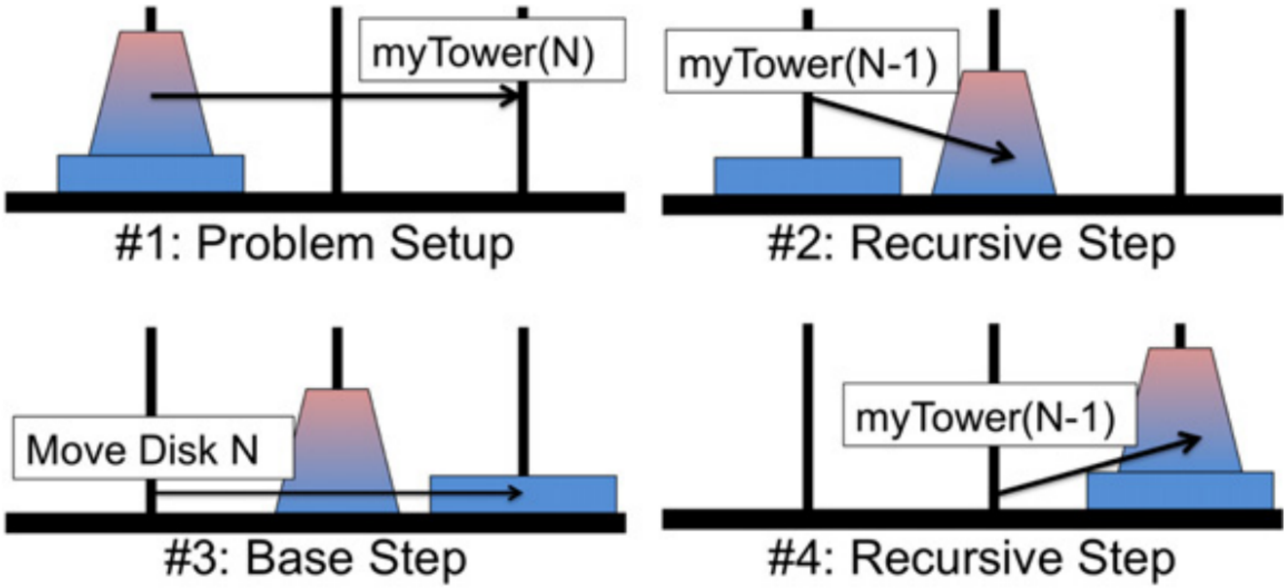
Rules:

- Only one disk can be moved at a time.
- Only the disk at the top of a stack may be moved.
- A disk may not be placed on top of a smaller disk.



Recursion: Towers of Hanoi

Simple to solve using recursive thinking: the key is breaking the problem down into smaller **subproblems**.



Recursion: Towers of Hanoi code

```
function hanoiTowers(N, from, to, alt)

if N~=0
    hanoiTowers(N-1, from, alt, to) % move N
        -1 towers from the start tower to the
        alternating tower
    display(sprintf('move disk %d from tower
        %d to tower %d', N, from, to)) %
        display on the screen the movement of
        bottom disk to final tower
    hanoiTowers(N-1,alt, to, from) % start
        over with the alternating tower as the
         'from' tower
end

end
```

Sorting algorithms

Sorting algorithms: sorted arrays

An array a of length n is sorted if, for every $1 < k \leq< n$,

$$a(k-1) \leq a(k) \text{ (ascending order)}$$

or

$$a(k-1) \geq a(k) \text{ (descending order)}$$

In other words, the elements of a are in 'order'.

Example:

$$a = [4, 20, 20.1, 32, 32, 56, 70]$$

is sorted in ascending order.

Sorting algorithms: sorting an array

Consider:

$$a = [56, 20, 32, 4, 20.1, 70, 32]$$

A simple sorting algorithm (ascending order):

1. $[56, 20.1, 32, \mathbf{4}, 20, 70, 32]$
2. $[4, 20.1, 32, 56, \mathbf{20}, 70, 32]$
3. $[4, 20, 32, 56, \mathbf{20.1}, 70, 32]$
4. $[4, 20, 20.1, 56, \mathbf{32}, 70, \mathbf{32}]$
5. $[4, 20, 20.1, 32, 56, 70, \mathbf{32}]$
6. $[4, 20, 20.1, 32, 32, 70, \mathbf{56}]$
7. $[4, 20, 20.1, 32, 32, 56, 70]$

This is known as the **Selection Sort** algorithm.

Sorting algorithms: the Selection Sort algorithm (iterative)

```
function out = selectionSort(arr)
n = length(arr);
for i = 1:n
    % Find the minimum element in the
       unsorted array
    [~,idx]= min(arr(i:end));
    min_idx = idx + i-1;
    % Swap the minimum element with the first
    if min_idx ~= i
        aux = arr(i);
        arr(i) = arr(min_idx);
        arr(min_idx) = aux;
    end
end
out = arr;
end
```

Sorting algorithms: the Quick-Sort algorithm

The quicksort algorithm starts with the observation that sorting a list is hard, but comparison is easy. So instead of sorting a list, we:

- Separate the array by comparing to a pivot
- The input array is divided into three parts: elements that are smaller, equal, and larger than the pivot.
- A recursive call is made on the two subproblems: the array of elements smaller than the pivot and the array of elements larger than the pivot.
- Eventually the subproblems are small enough (i.e., array size of length 1 or 0) that sorting the list is trivial.

Sorting algorithms: the Quick-Sort algorithm

```
function sorted = quickSort(arr)
if length(arr) <= 1
    sorted = arr; % lenght 1 already sorted
else
    pivot = arr(1); %first element as pivot
    bigger = []; smaller = []; same = [];
    for i = 1:length(arr)
        if arr(i) > pivot
            bigger = [bigger arr(i)];
        elseif arr(i) < pivot
            smaller = [smaller arr(i)];
        else
            same = [same arr(i)];
        end
    end
    sorted = [quickSort(smaller), same,
       quickSort(bigger)]; % recursive call
end
```

Sorting algorithms: sorting in MATLAB

`MATLAB` has the built-in function `sort` for sorting.

It is based on the Quick-sort algorithm.