Prof. Dr. Gemma Roig
M.Sc. Alperen Kantarcı
M.Sc. Gamze Akyol

Programmieren für Studierende der Naturwissenschaften

Lecture 9 – Algorithms

Contents

- L6: External Packages, Introduction NumPy and SciPy
  P6: Exercises

- L7:External Packages 2
  P7: Exercises

- L8: Handling external data and visualization
  P8: Exercises

- L9: Design of algorithms
  P9: Exercises (not graded) and independent work in small groups

Algorithms

- How to write algorithms on paper?
- More about the complexity, than writing algorithm

"Algorithm design - that's the field where people talk about programs and prove theorems about programs instead of writing and debugging programs."

An Introduction to Algorithm Design, Jon Louis Bentley Carnegie-Mellon University, 1979.

Review of the first lecture

1. Describe and analyze a problem

2. Selection, development and description of the required algorithms

3. Transfer / conversion into a programming language

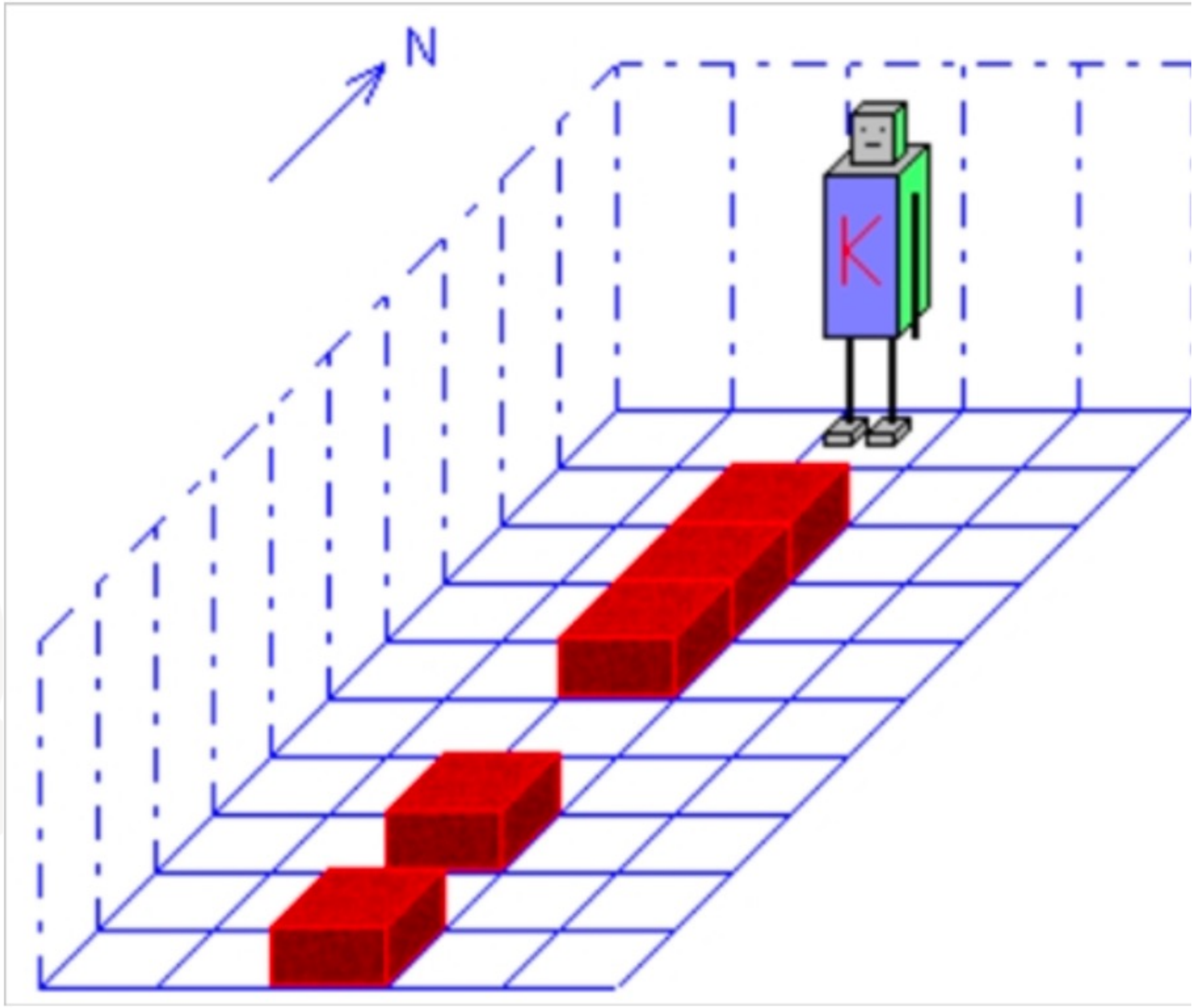
4. Testing

All this is obviously not trivial!

Possibilities before implementation

- Natural language (text/keywords)
- Pseudocode (Based on code, there are no fixed rules)
- Graphical(flowcharts, state transition diagrams)
- In practice, often a mixture.

Example



Source: http://www.inf- schule.de/algorithms/algorithms/algorithm_term/exkurs_darstellung, 16.09.2017

Robot example

As long as the wall has not yet been reached, repeat the following:
If there is a brick in the way, pick it up and go one step further.
Otherwise, go directly one step further.
Turn 180º degrees.
As long as the wall is not reached, go one step further.
Turn 180º degrees.

Robot example

As long as the wall has not yet been reached, repeat the following:
If there is a brick in the way, pick it up and go one step further.
Otherwise, go directly one step further. Turn 180º degrees.
As long as the wall is not reached, go one step further.
Turn 180º degrees.

```
While wall has not been reached:
    If there is a brick in the way:
        pick it up
        go one step further.
    else:
        go directly one step further
        turn 180º degrees.
        while wall is not reached:
            go one step further.
        Turn 180º degrees.
```

Division Example

Division

Algorithm 2: Division

function divide (x, y);

Input: Two n-bit integers x and y, where $y \geq 1$

Output: The quotient and remainder of x divided by y

```
if x = 0 then
|   return (q, r) = (0, 0)
else
|   set (q, r) = divide(⌊x/2⌋, y);
|   q = 2 × q, r = 2 × r;
|   if x is odd then
|   |   r = r + 1
|   end
|   if r ≥ y then
|   |   r = r − y, q = q + 1
|   end
|   return (q, r)
end
```

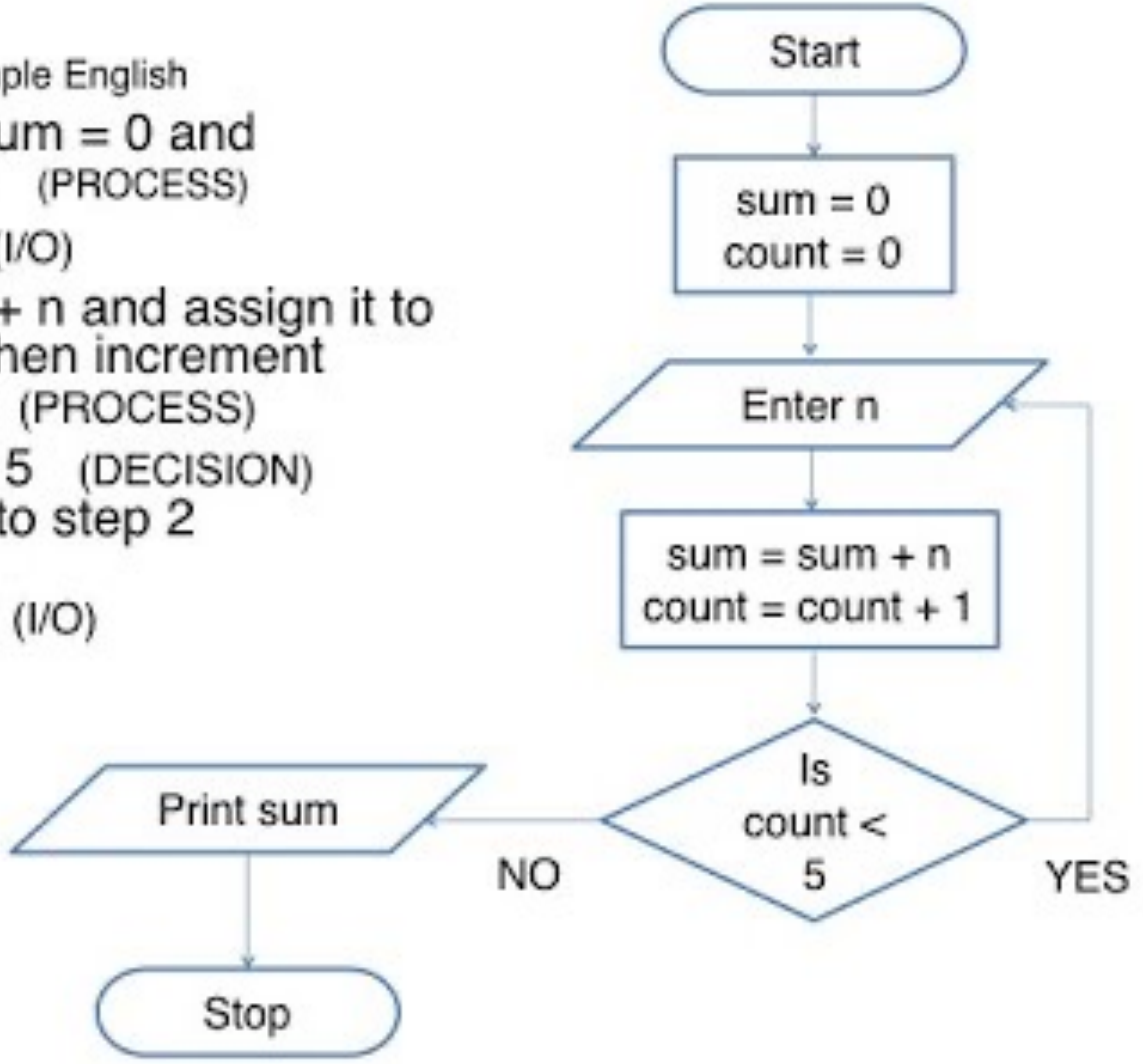
Flowcharts and Pseudocodes

Find the sum of 5 numbers

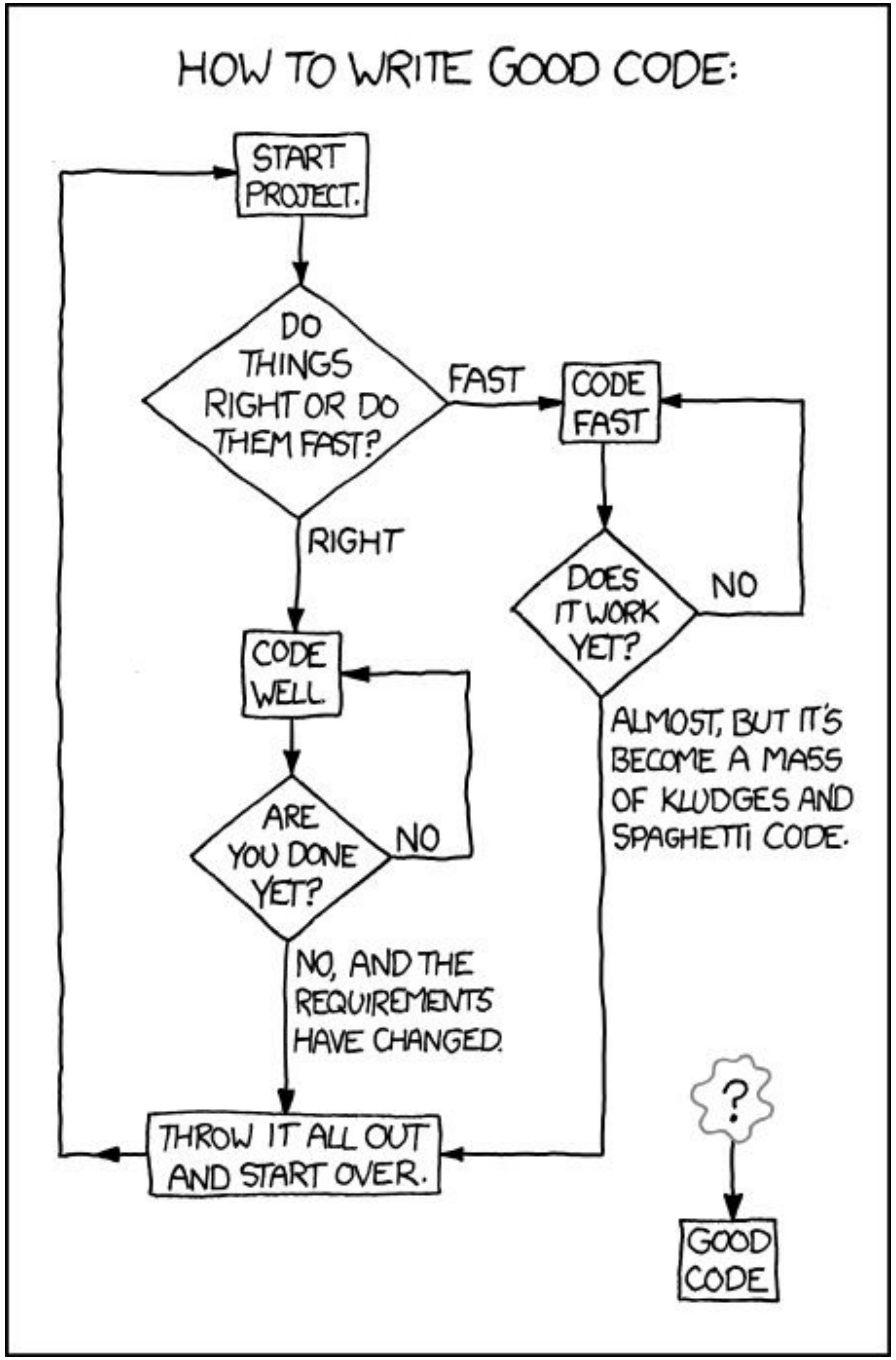
Algorithm in simple English

1. Initialize sum = 0 and count = 0 (PROCESS)
2. Enter n (I/O)
3. Find sum + n and assign it to sum and then increment count by 1 (PROCESS)
4. Is count < 5 (DECISION)
   if YES go to step 2
   else
   Print sum (I/O)

Flowchart



Flowcharts and Pseudocodes



Criticism of Flowcharts

- Already with medium-sized algorithms quickly confusing
  - Requires a lot of space
- Tends to use explicit jump instructions

- If one corrects an error in thinking, many things in the flow chart would have to be "tightened up" if necessary

- Rarely encountered in real-world algorithm writing

- **But:** are well suited to clarify elementary structures of programming and to facilitate the start!

Complexity of an algorithm

- Measure how "efficiently” a program works
- Non sensical instructions or inefficient structures can cost a lot of computing time and / or memory
- **Optimization:**
  - Often time gains with larger amounts of memory or memory optimization with longer computation times
  - Not frequently, time gains are at the expense of accuracy. An approximated solution in one hour is sometimes better than an exact one in 40,000 years.
- **Consequence:**
  - If solid efficiency optimization is the goal, then already in the conceptual phase
  - Subsequent efficiency improvements are often associated with a high level of effort.

When you found the algorithm

Algorithm properties

- Correctness
- Terminability
- **Complexity**
  - Computing time
  - Storage space

Algorithm Analysis

- **Computing time**
  - Number of elementary operations performed depending on the input size
- **Storage space**
  - The maximum memory consumption during the execution of the algorithm depends on the complexity of the input.
- **Data transfer**
  - How large is the data that needs to be transferred

Computing Time

- **1st approach**
  - Direct measurement of the runtime (e.g.in ms): Depends on many parameters, like computer configuration, computer load, compiler, operating system...
  - Therefore: hardly transferable and inaccurate

- **2nd approach:**
  - Counting the required elementary “operations of the algorithm depending on the size of the input.
  - The algorithmic behavior is represented as a function of the number of elementary operations required.
  - Categorization into different quantities

Charactirize input data

- Most of the time it is very difficult to find an exact distribution of the data that corresponds to the real case.

- Therefore, we usually have to **consider the worst case** and find an upper bound for the running time
  - If this **upperbound** is correct, we guarantee that the running time of our algorithm for any input data is always less than or equal to this bound.

- **Example:**
  - I want to sort a list. What is the maximum number of steps I need depending on the input list?

What operations are considered?

- Small operations that combine several smaller operations that are executed individually in constant time, but significantly contribute to the total time required by the algorithm due to their frequent occurrence.
- Runtime of individual elementary operation is then dependent on the computer hardware used
- **Example:**
- For sorting algorithms:
    - Compare

- In other algorithms:
    - Memory accesses
    - Number of multiplications
    - Number of bit operations
    - Number of loop passes
    - Number of function calls

O notation

•With the O-notation, computer scientists have found a way to characterize the asymptotic complexity (in terms of runtime or memory requirements) of an algorithm.

•O-notation allows algorithms to be compared at a higher level of abstraction (independent of implementation details such as programming language, compiler, and hardware properties)

• O(...) denotes the set of all functions that have the same complexity class with respect to the input (i.e. except for a constant factor)

$$2\text{-}n \in O(n)$$
$$n/2 + 1 \in O(n)$$

At this point no definition, but an example

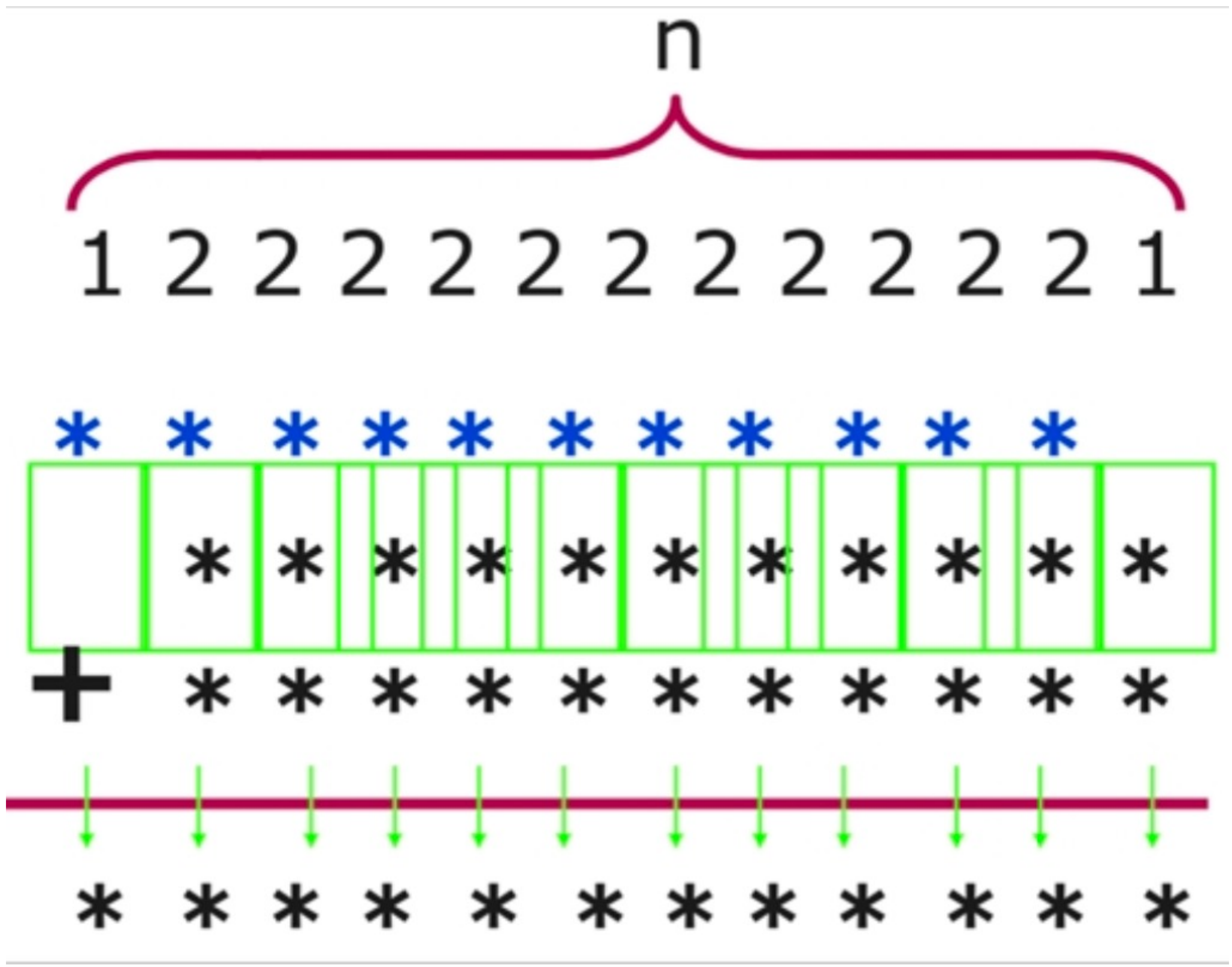
Written addition and multiplication

- **Inputsize:** n = digits of the number

- **Calculation step:**
- Addition of 2 digits
- Carry forward if necessary

- **Complexity Analysis:**
    - T(n) = number of calculation steps to add two numbers with n digits

- **Worst case:**
    - T(n) = 2n (additions)
    - →linear→classO(n)

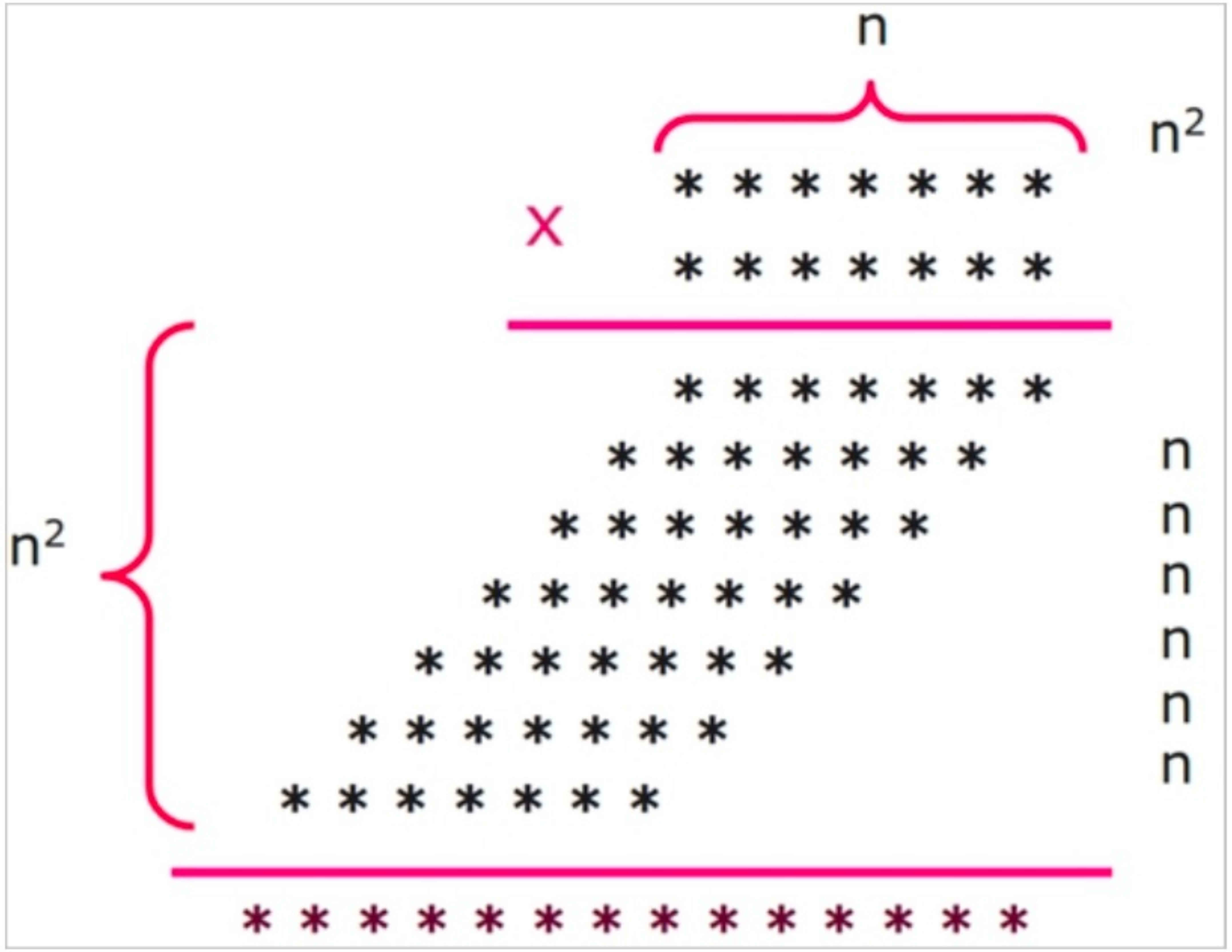


Written addition and multiplication

- **Input size**: n = digits of the number
- **Calculation step:**

- Multiplication of 2 digits
- Addition of the results

- **Worst case:**
- T(n)=n multiplications + n^2 + n additions
- →quadratic → class O(n^2)



Complexity Classes

	Sprechweise	Typische Algorithmen / Operationen
$O(1)$	konstant	Addition, Vergleichsoperationen, rekursiver Aufruf, …
$O(\log n)$	logarithmisch	Suchen auf einer sortierten Menge
$O(n)$	linear	Bearbeiten jedes Elementes einer Menge
$O(n \cdot \log n)$		gute Sortierverfahren
$O(n \cdot \log^2 n)$		
…		
$O(n^2)$	quadratisch	primitive Sortierverfahren
$O(n^k)$, $k \geq 2$	polynomiell	
…		
$O(2^n)$	exponentiell	Ausprobieren von Kombinationen

Another example

```
def sum(n):
  summe =0
  for i in range(n+1):
    summe += i
  return summe
```

```
def  sum(n):
  return n*(n+1)/2
```

More examples

- Loop passes n times.
- In the loop: constant operations (addition and assignment).
- → n * 1 operations
- → O(n)

```
def sum(n):
  summe =0
  for i in range(n+1):
    summe += i
  return summe
```

- According to Gauss
- →3 constant operations
- → O(1)

```
def  sum(n):
  return n*(n+1)/2
```

More examples

Fibonacci-Series: 1,1,2,3,5,8,13,21,34...

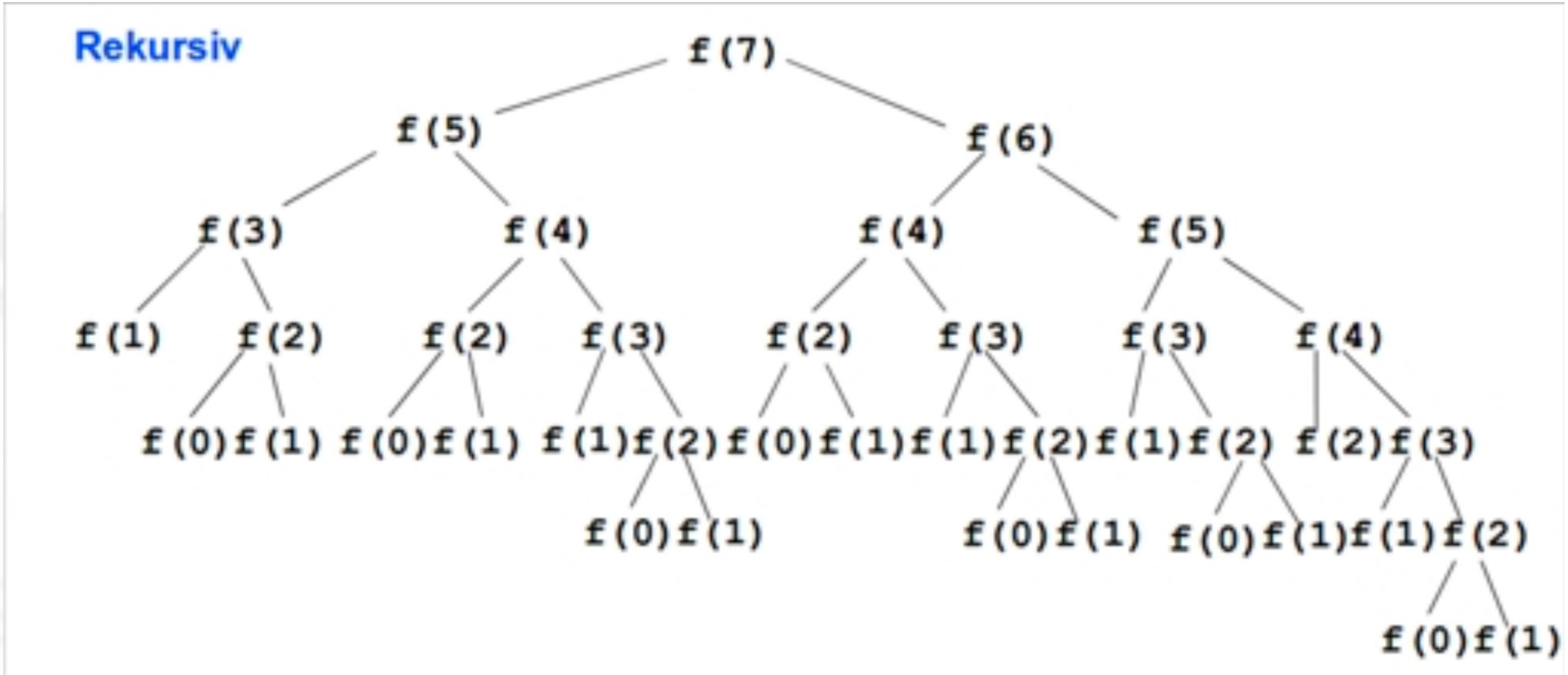
- The next sequence element is always the sum of the two preceding ones.

- Occurs frequently in nature (e.g., golden ratio, growth rates in plants, population growth, etc.).

- Is often described recursively (i.e. as a function that calls itself):

```
fib(n) = fib(n-1)+fibn(n-2)
```

More examples



Conclusion

- The choice of the algorithm can have influence on runtime of the program

- Computer science has found a system to categorize these differences.

- With small programs you may not notice any differences, but as soon as problems become bigger, runtime can become important.

Project Ideas

1. **Hangman Game (A game to play from terminal, with small hang)**

```
  +---+
  O   |
 /|\  |
 /    |
     ===

Missed letters:  s e r t d
_ o _
Please guess a letter.
n


  +---+
  O   |
 /|\  |
 / \  |
     ===

Missed letters:  s e r t d n
_ o _
You have run out of guesses!
After 6 missed guesses and 1 correct guesses, the word was "fox"
Would you like to play again? (y)es or (n)o
```

Project Ideas

1. **Hangman Game (A game to play from terminal, with small hang)**
2. **Password Strength Checker or Password Generator**

```
Enter password length: 20
Choose character set for password :
          1. Digits
          2. Letters
          3. Special characters
          4. Exit
Pick a number: 1
Pick a number: 2
Pick a number: 3
Pick a number: 4
```

Project Ideas

1. **Hangman Game (A game to play from terminal, with small hang)**
2. **Password Strength Checker or Password Generator**
3. **Tic-tac-toe Game**

```
You are X: Choose number from 1-9: 3

          X | O | X
         -----------
            | X |
         -----------
          O | X | O

Computer choosing move...
choices:  [3, 5]

          X | O | X
         -----------
            | X | O
         -----------
          O | X | O

You are X: Choose number from 1-9: 4

          X | O | X
         -----------
          X | X | O
         -----------
          O | X | O

Game Over. Nobody Wins
```

Project Ideas

1. **Hangman Game (A game to play from terminal, with small hang)**
2. **Password Strength Checker or Password Generator**
3. **Tic-tac-toe Game**
4. **Simple calculator (with memory function)**
5. **Palindrome checker**
6. **Contact Book**

```
Python 3.9.0 (tags/v3.9.0:9
Type "help", "copyright", "
>>>
=============== RESTART: C:\
Enter your string: madam
madam is a palindrome
>>>
=============== RESTART: C:\
Enter your string: sir
sir is not a palindrome
>>>
```

```
Welcome to your favorite address book!
What do you want to do?
        | List       | Lists all users
        | Add        | Adds an user
        | Delete     | Deletes an user
        | Delete all | Removes all users
        | Search     | Search or a user
        | Close      | Closes the address book
list
No contacts found
What do you want to do?
        | List       | Lists all users
        | Add        | Adds an user
        | Delete     | Deletes an user
        | Delete all | Removes all users
        | Search     | Search or a user
        | Close      | Closes the address book
```