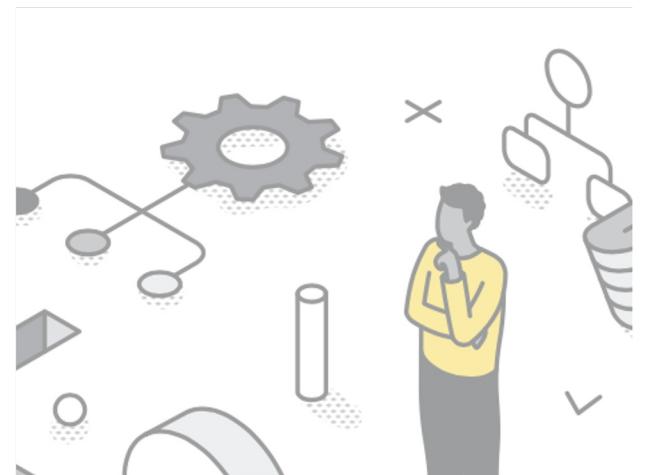




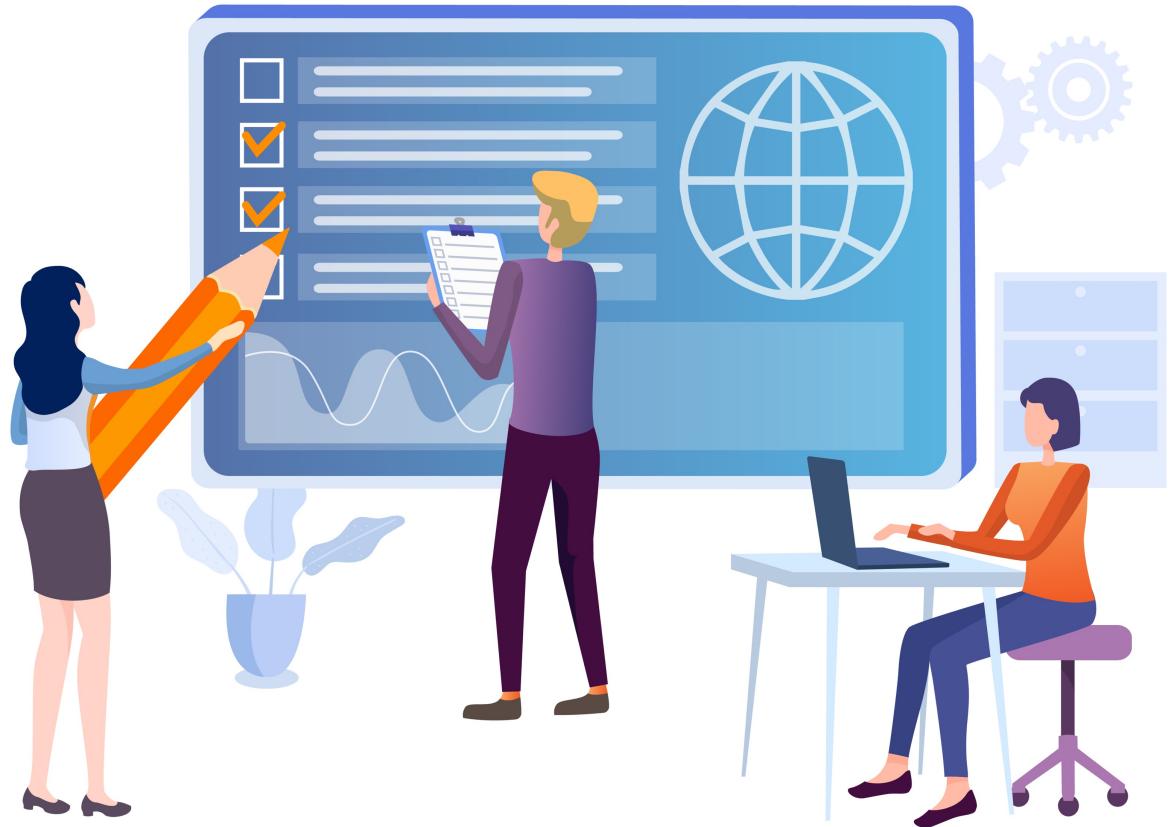
Stelios Sotiriadis

2. Introduction to algorithms



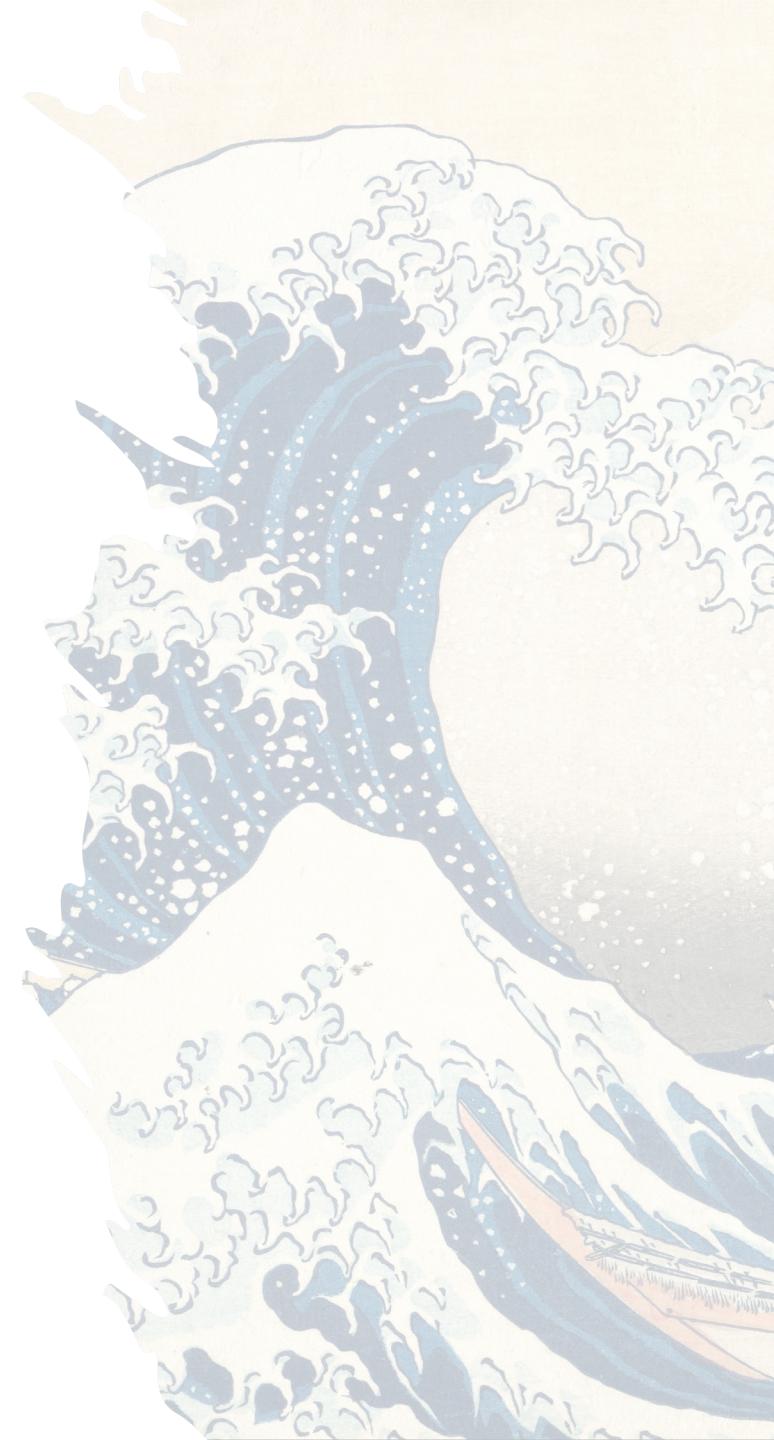
Quiz of the day

Get ready!



Agenda

- ▶ Algorithms for problem-solving!
- ▶ Computational complexity (second part)
- ▶ Sorting (bubble sort, merge etc.)
- ▶ Algorithmic classes:
 - Divide and conquer, Greedy and Dynamic programming
- ▶ **Race lab!**
 - Join me in 404-405 & online



Computational complexity

- ▶ Time
 - Number of operations to complete a task
- ▶ Space
 - Amount of extra memory used for completing a task



Big O (worst case)

- ▶ **O(1) [Constant time]**

- Accessing an element in an array by index.

- ▶ **O(n) [Linear time]**

- Linear search in an array.

- ▶ **O(logn) [Logarithmic time]**

- Binary search in a sorted array.

- ▶ **O(n^2) [Quadratic time]**

- Sort data by repeatedly swapping the adjacent elements if they are in the wrong order.

Quiz

► Write down all the possible permutations of ABC?

- **ABC** Can you think of a scenario where generating all possible permutations could be useful?
- **ACB**
- **BAC** Software testing
- **BCA** Can you think of a mathematical formula to generalise this example?
- **CAB**
- **CBA** $3! = 1*2*3 = 6$

$O(N!)$ [Factorial time]

- ▶ Generating all possible permutations of a set of elements is inherently factorial.
 - Consider a set of N elements
 - There are $N!$ permutations to be generated

Quiz

- ▶ You have been tasked with guessing a password.
 - The password includes three digits, each of which can be either 1 or 0. A digit may appear multiple times or not at all in the combination. Write down all the possible combinations!
 - Can you think of a mathematical formula to generalise this example?
 - 000
 - 001
 - 010
 - 011
 - 100
 - 101
 - 110
 - 111
- $2^3 = 8 = 2^n$

$O(2^N)$ [Exponential time]

- ▶ All possible combinations!
 - Scenario: In security testing or hacking, generating all possible passwords up to a certain length, given a specific character set (brute-force password cracking).

Question

- ▶ What is worse, factorial or exponential?

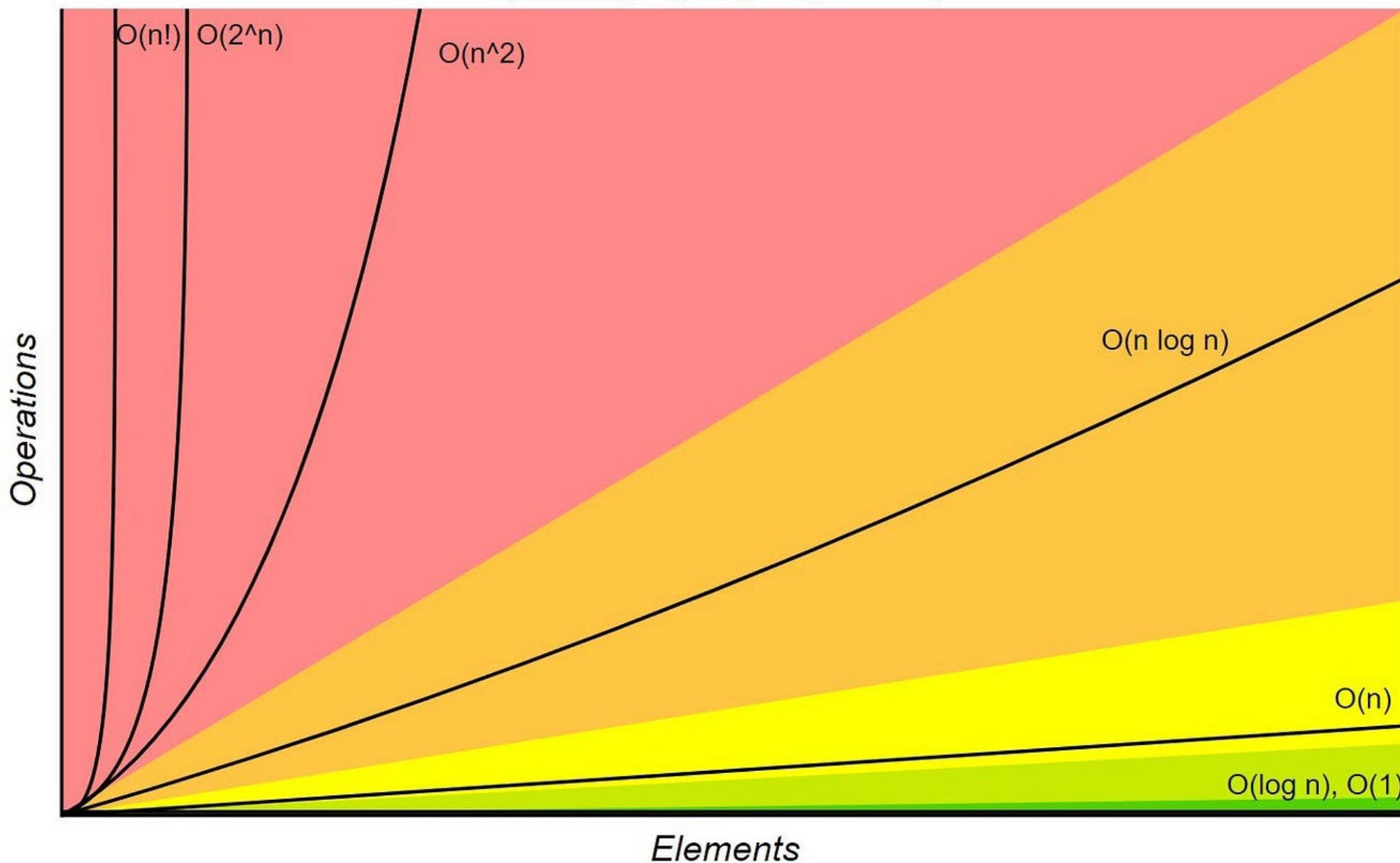
$$2^5 = 2*2*2*2*2 = 32$$

$$5! = 1*2*3*4*5 = 120$$

- By the time you reach $n=10$,
 - $10!=3,628,800$
 - $2^{10}=1,024$
- The factorial function grows much faster!

Big-O Complexity Chart

Horrible Bad Fair Good Excellent



Group discussion

- ▶ What is the most efficient way to swap elements of a list?

- `alist = [2,4,6,8,12,14]`
- `swap(alist,6,8) → [2,4,8,6,12,14]`

[2,4,8,6,12,14]

↑ ↑ ↑ ↑ ↑ ↑

Time complexity? $O(n/2) \rightarrow O(n)$

Space complexity? $O(1)$ ☺

Sorting methods

What is sorting?

- ▶ Sorting refers to ordering data in an increasing or decreasing manner according to some linear relationship among the data items.
- ▶ Bubble sort:
 - Repeatedly steps through the list to be sorted, compares each pair of adjacent items, and swaps them if they are in the wrong order.

Bubble sort

#1



#2



#3



#4



#5



#6



#7



#8



What is the complexity of bubble sort?

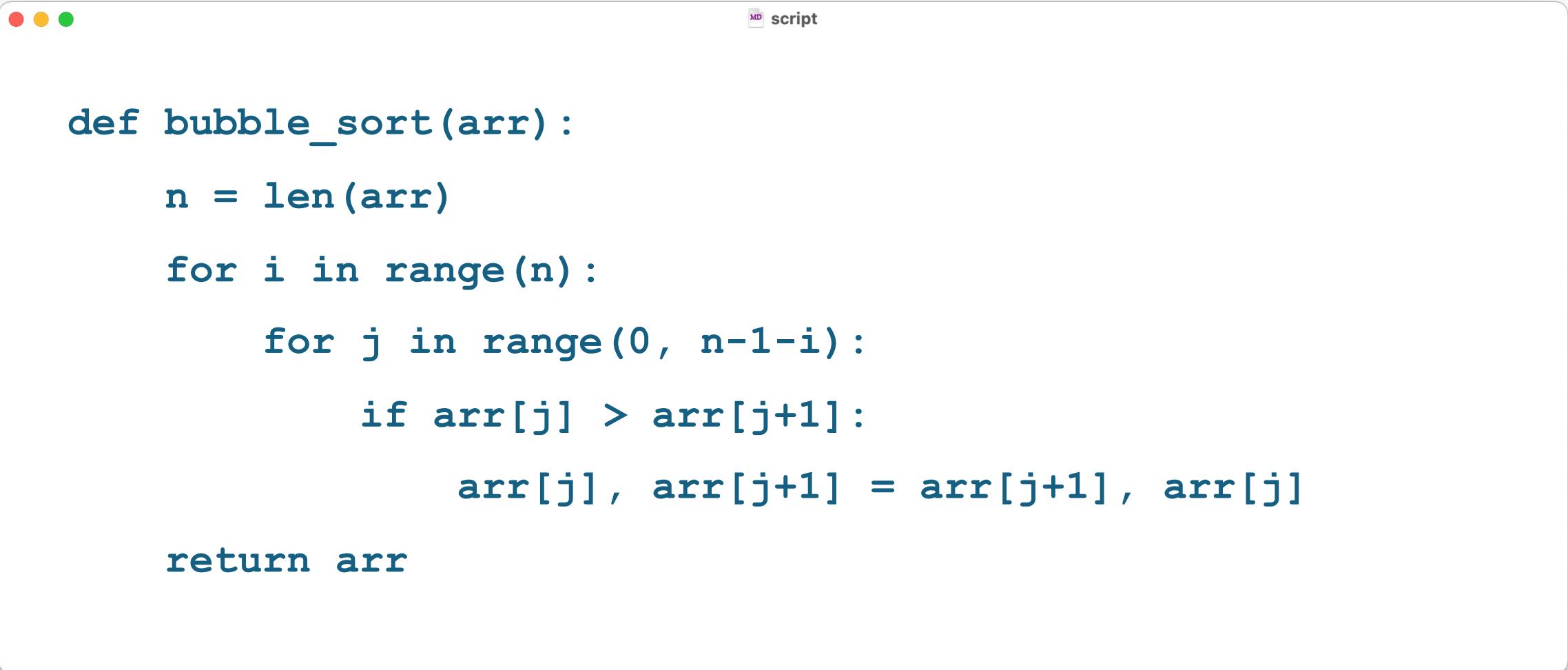
► Time:

- Repeatedly traversing the array and comparing adjacent items, swapping them if they are in the wrong order.
 - Each complete pass through an array of length **n** compares elements in pairs, leading to **n-1** comparisons (first pass), **n-2** (second), and so on, down to **1** comparison in the last pass.
 - $(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2} \approx O(n^2)$

► Space:

- **O(1)** – in-place swapping, no extra space is used ☺

Bubble sort

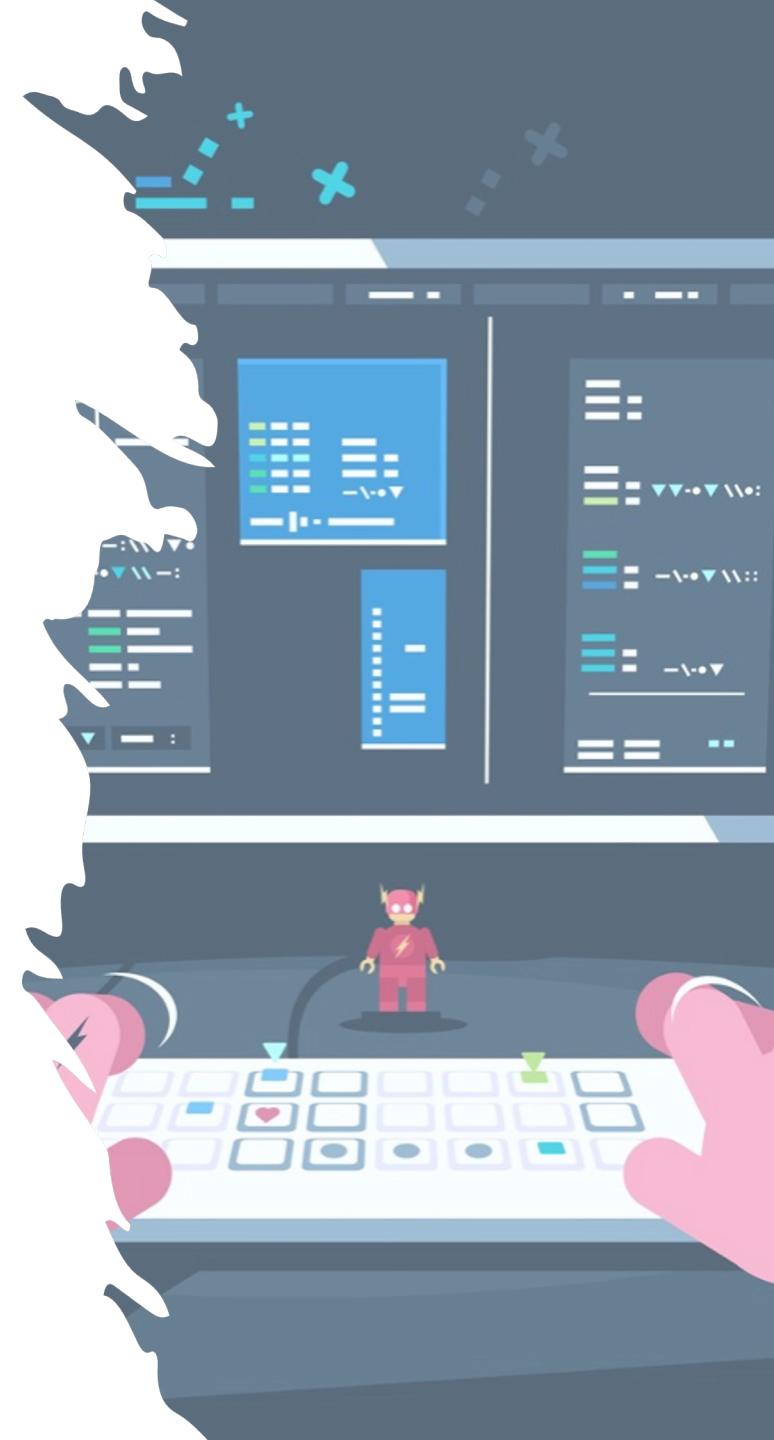


A screenshot of a terminal window on a Mac OS X system. The window title bar shows three colored window control buttons (red, yellow, green) on the left and a "script" icon with a small "MD" logo on the right. The main area of the terminal contains the following Python code:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-1-i):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

Insertion sort

- ▶ Iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list.
 - It is a stable sorting algorithm (elements with equal values maintain relative order).



Insertion sort

3 1 6 4

3 1 6 4

3 3 6 4

1 3 6 4

1 3 6 4

1 3 4 6



temp=1

temp=1

3 3 6 4

1 3 6 4

1 3 6 4

Time: $O(n^2)$
Space: $O(1)$

Merge sort

38 27 43 10

38 27

43 10

27

38

10

43

27 38

10 43

10 27 38 43

Time: $O(n \log n)$

Space: $O(n)$

What is $O(n \log n)$?

- ▶ Merge Sort:

- Divides the array into two halves, sorts each half, and then merges the sorted halves.
- Each divide (and subsequent merge) step takes linear time, and the division happens $\log n$ times (the depth of the recursion tree).



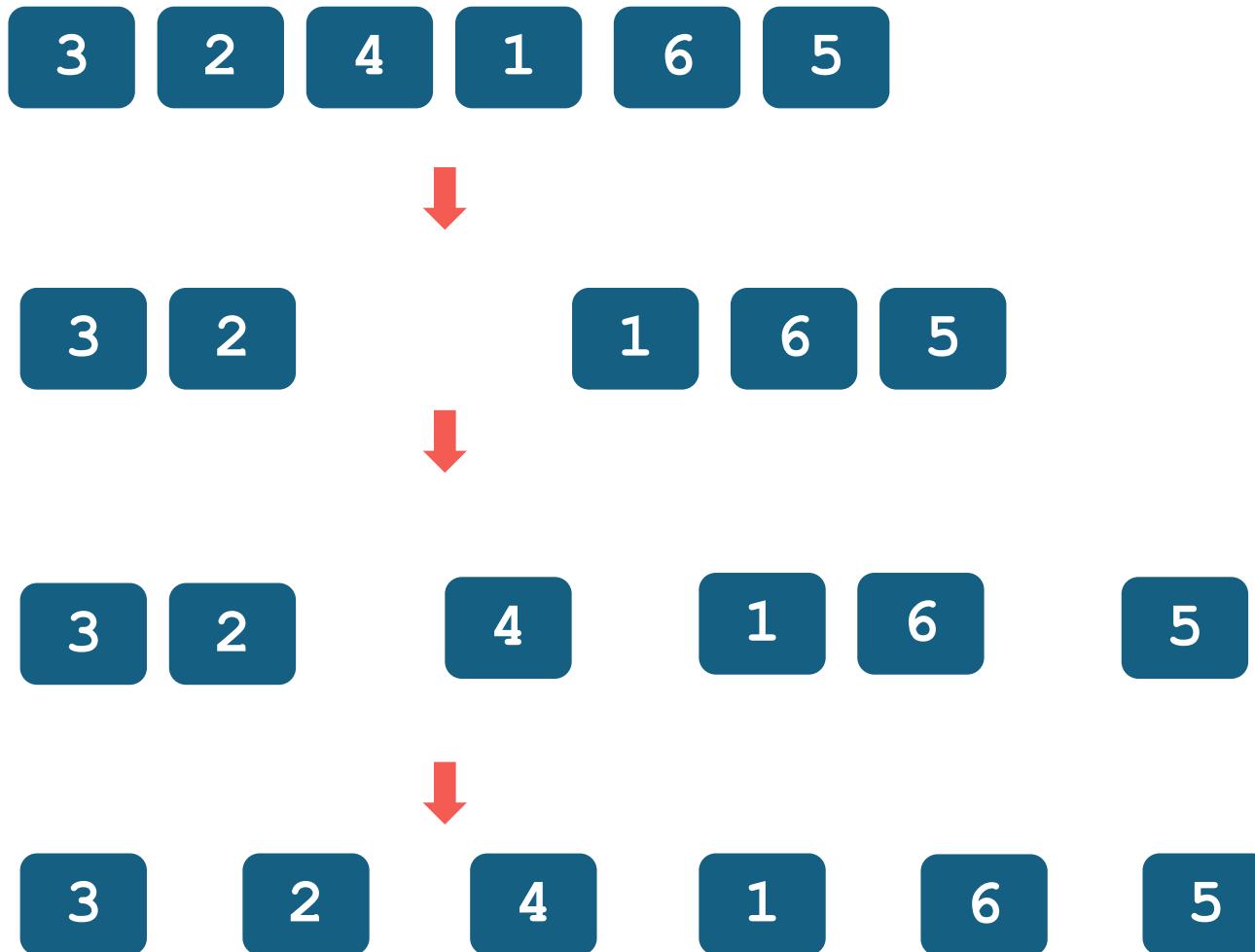
Quiz

- ▶ Using pen and paper sort the following numbers using merge sort!

3 2 4 1 6 5

- ▶ First, split the array into smaller sub-arrays until each sub-array has only one element.
- ▶ Start merging them back together in a sorted order.
- ▶ Merge the two sorted sub-arrays

Step 1: Split the Array



First, split the array into smaller sub-arrays until each sub-array has only one element.

Step 2: Merge and Sort

3 2 4 1 6 5



2 3 4 1 6 5



2 3 4 1 5 6

Step 3: Final merge

2 3 4

1 5 6



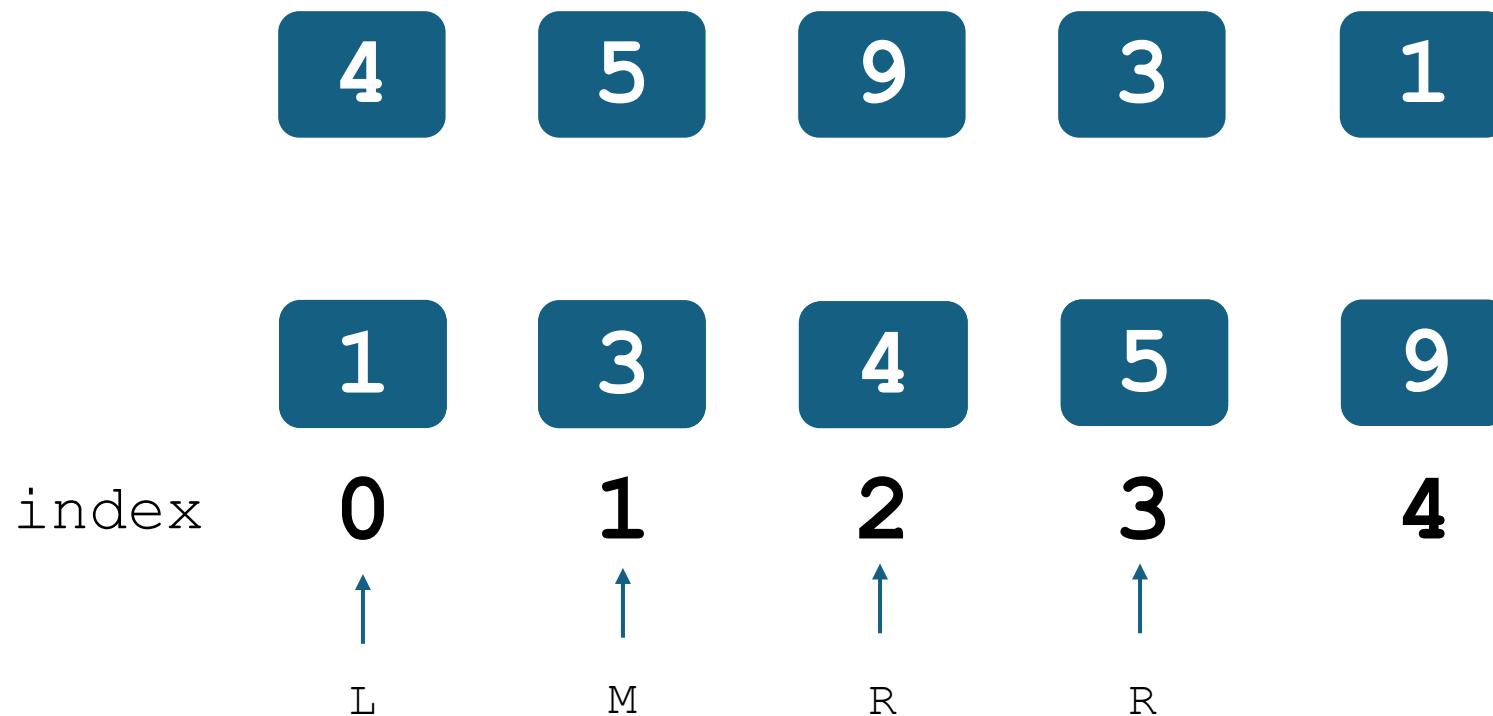
1 2 3 4 5 6

Tim sort

- ▶ Tim Sort is a hybrid sorting algorithm derived from merge sort and insertion sort. It was designed to perform well on many kinds of real-world data.
 - Tim Sort is the default sorting algorithm used by Python's `sorted()` and `list.sort()` functions.
- ▶ Time:
 - $O(n * \log(n))$
- ▶ Space
 - $O(n)$

TimSort uses binary search insertion

- ▶ Instead of using linear search to find the location where an element should be inserted, it uses binary search.



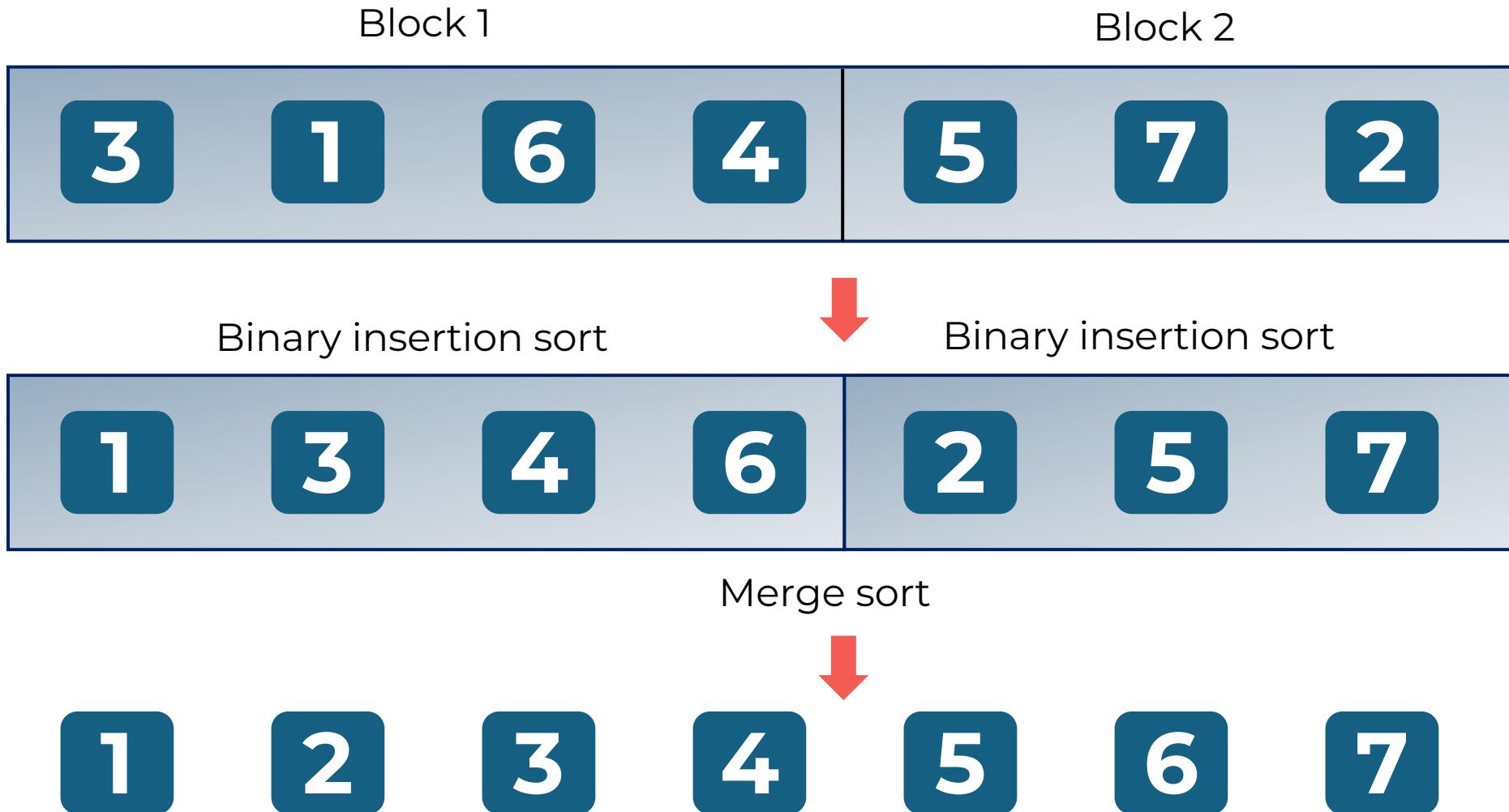
Tim sort steps

- ▶ Tim sort assumes that some data is already ordered...
- ▶ We call this a run:
 - **arr = {5,7,8,9,3, ... }**
 - **Run1 is {5,7,8,9}**
- ▶ Each run has a minimum length of elements (min run):
 - **arr = {8,12,3,9,14, ... }**, with min run = 3
 - Run is {8,12,3}
 - Tim sort performs binary insertion, so the run becomes {3,8,12}

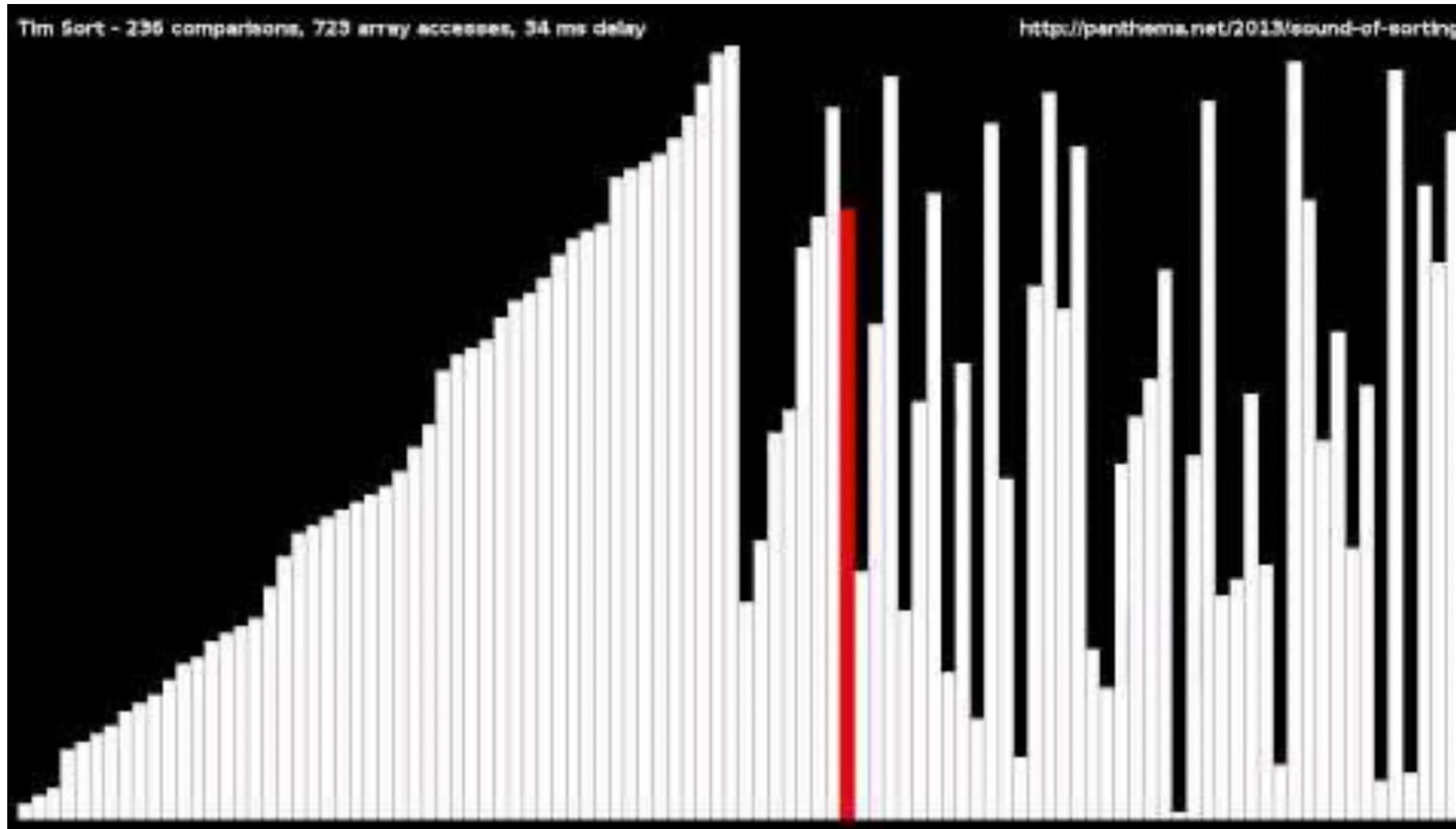
Tim sort steps

- ▶ Tim sort has some cool feature
 - If the min run is met, and the next element follows the same pattern (ascending order), then increase the size of the run...
 - `arr = {5,7,8,9}, min run = 3`
 - `run1 = {5,7,8} → {5,7,8,9}`
- ▶ If the run is in descending order, then blindly reverse!
 - `run1 = {8,4,2,1} becomes {1,2,4,8}`

The whole picture!



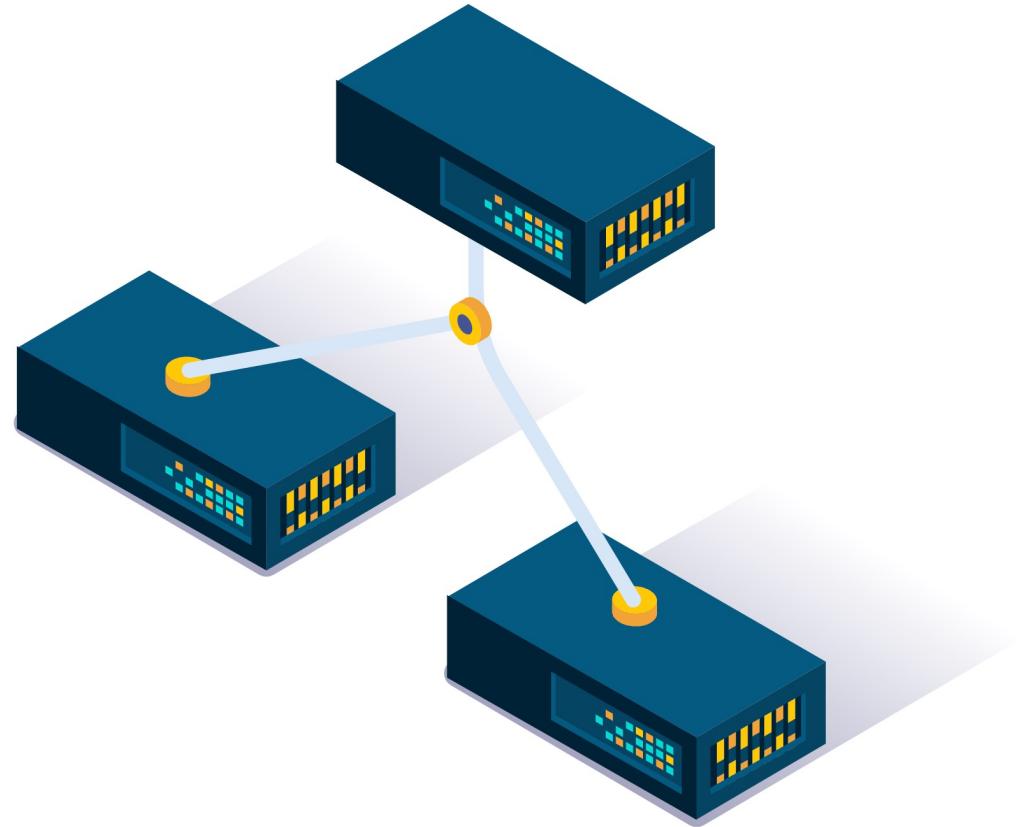
Tim sort: The fastest sort!



Let's compare them!

Algorithm	Time	Space
Linear search	$O(n)$	$O(1)$
Binary search	$O(\log n)$	$O(1)$
Bubble sort	$O(n^2)$	$O(1)$
Insertion sort	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n)$
Timsort	$O(n \log n)$	$O(n)$

Divide and conquer!



Divide and conquer

- ▶ A general algorithm design paradigm
 - Divide: divide the input data **s** in two or more disjoint subsets **s₁**, **s₂**, ...
 - Recur: solve the subproblems recursively
 - Conquer: combine the solutions for **s₁**, **s₂**, ..., into a solution for **s**
- ▶ Examples:
 - Binary search
 - Merge sort

Sometimes, we cannot do better!

- ▶ Divide and conquer cannot be used.
- ▶ **Greedy algorithms,**
 - Unlike divide and conquer, make a series of choices that seem best at the moment with the hope of finding a global optimum.



Problem

- ▶ You must run nine Python scripts, each lasting **3, 5, 6, 10, 11, 14, 15, 18, and 20** minutes.
- ▶ You have three computers on which you can run these scripts.
- ▶ What is the best combination to run your scripts?

Discussion

- ▶ You must run nine Python scripts, each lasting **3, 5, 6, 10, 11, 14, 15, 18,** and **20** minutes.
- ▶ You have three computers on which you can run these scripts.
- ▶ What is your strategy for running the tasks?



Bin packing problem!

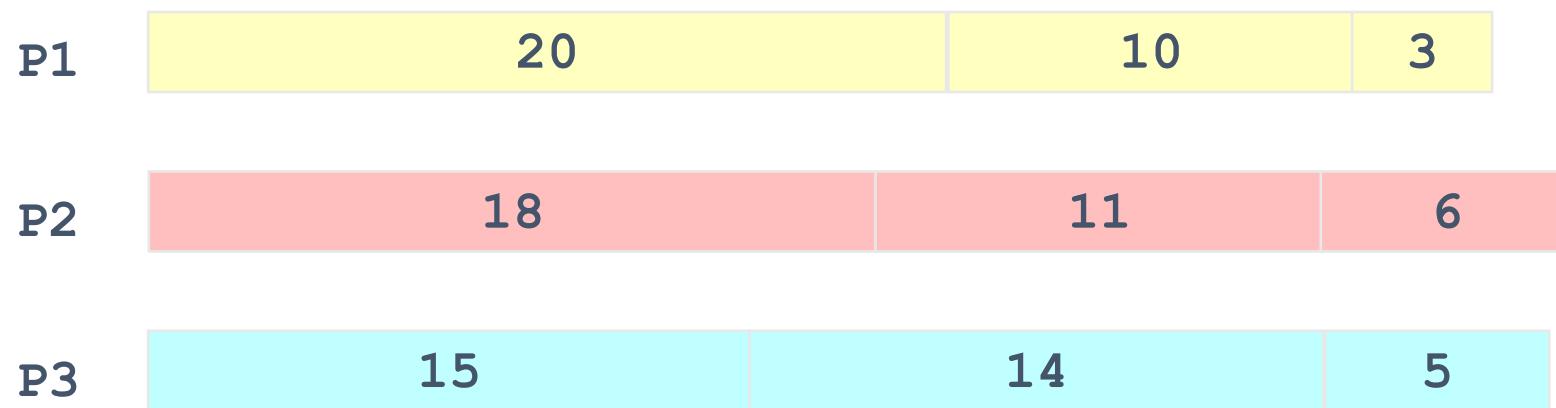
- ▶ Objects of different volumes must be packed into a finite number of bins each with a specific capacity in a way that minimises the number of bins used.
- ▶ Can you think of real-world examples?
 - Loading Trucks
 - Filling Containers in Shipping
 - Resource Allocation in Computing
 - Cutting Stock Problem
 - Packing Goods in Warehouses



Solution 1

- ▶ You decide to do the longest-running jobs first on whatever processor is available.

{3, 5, 6, 10, 11, 14, 15, 18, 20}

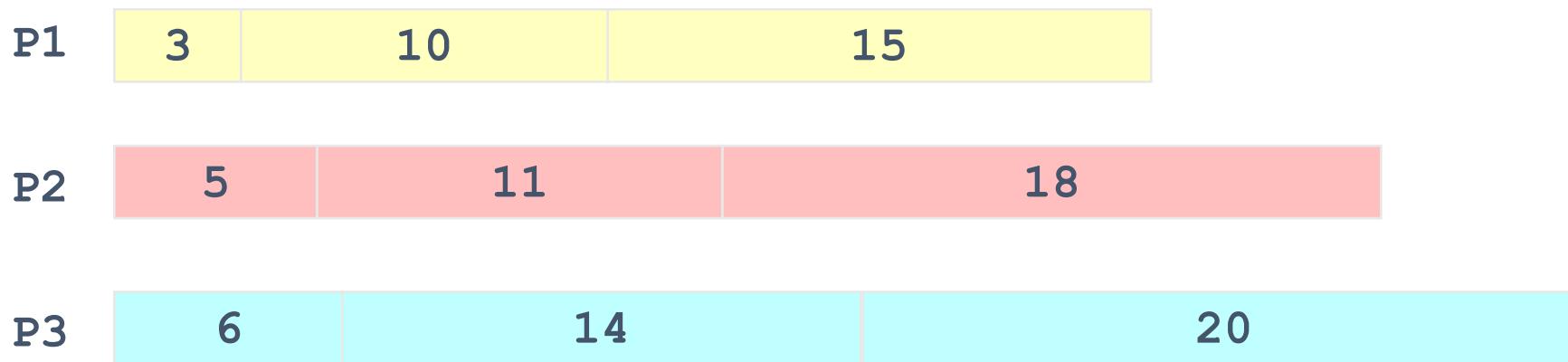


- ▶ Time to completion: $18 + 11 + 6 = 35$ minutes
- ▶ This solution isn't bad, but we might be able to do better

Solution 2

- ▶ You decide to do the shortest-running jobs first.

{3, 5, 6, 10, 11, 14, 15, 18, 20}



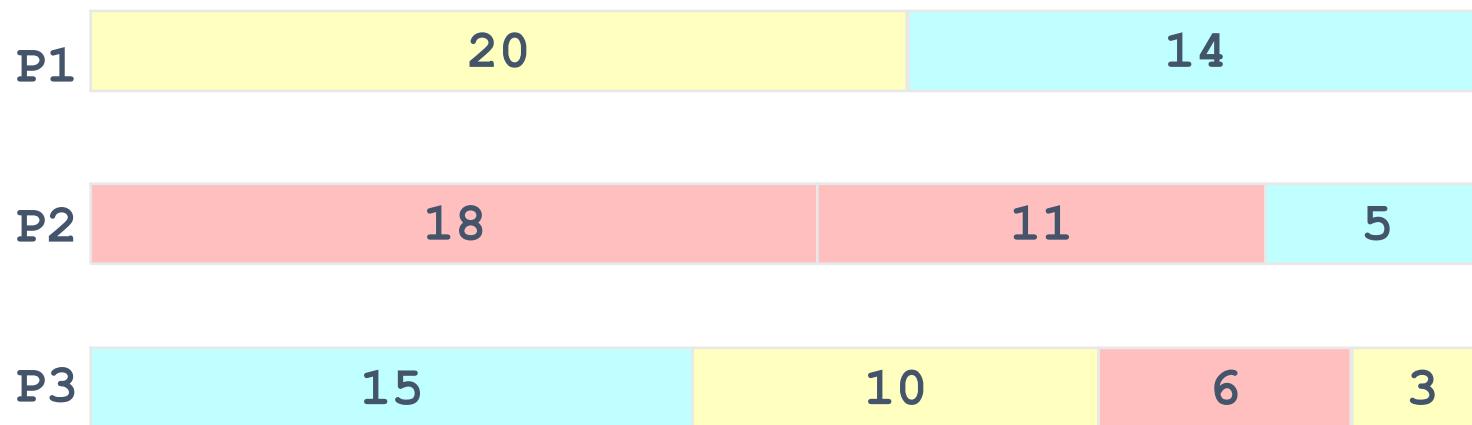
- ▶ That wasn't such a good idea; time to completion is now

$$6 + 14 + 20 = 40 \text{ minutes}$$

Solution 3

- ▶ Better solutions do exist:
 - Greedy: at each stage, pick the maximum values.

$\{3, 5, 6, 10, 11, 14, 15, 18, 20\}$



- ▶ Time to completion: $20 + 14 = 34$ minutes

How to find the optimal solution?

- ▶ One way:

- Try all possible assignments of jobs to processors
- $\{3, 5, 6, 10, 11, 14, 15, 18, 20\}$ has nine elements.
 - The number of possible combinations: $2^9 = 512$
- What if I had 100 jobs?
 - $2^{100} = 1,267,650,600,228,229,401,496,703,205,376$

- ▶ Unfortunately, this approach takes exponential time!

- Let's select the best option available at the moment.

What is optimization?

- ▶ Do you have a problem that requires too many operations (quadratic, exponential, or factorial)?
- ▶ Find a near-optimal or acceptable solution!
 - If it is costly, time-consuming, or technically infeasible



Fintech example

► Resource Allocation in Finance:

- You want to invest £10,000
 - You decided to build a portfolio and make various investments with the hope of maximising your profit.
 - There are some risks:
 - Volatility levels of stocks (price can change dramatically in a short period)
- How do we model this scenario?
 - Each investment option has a specific return (value) and risk (weight), and the goal is to maximise the return on investment.

Ali Baba and the 40 thieves...

- ▶ Ali Baba is a woodcutter who discovers the secret cave hideout of a band of forty thieves while working in the forest.
- ▶ He overhears the thieves' leader, often named Cassim, using a magical phrase to open the cave where they store their stolen treasures. Ali Baba learns the phrase and gains access to the cave.
 - Open sesame!
- ▶ Ali Baba enters the cave with a bag and takes some of the treasure for himself...
- ▶ What is the best strategy for Ali Baba to get as much as possible out of the cave?



Knapsack fractional: A maximisation problem!

7 objects ($n=7$)

15 is the size of a bag ($m=15$)

Objects	O	1	2	3	4	5	6	7
Profits	P	10	5	15	7	6	18	3
Weight	W	2	3	5	7	1	4	1

- ▶ The question is: How do we fill the bag so that the profit is maximized?

Knapsack fractional: A maximization problem!

Objects:	o	1	2	3	4	5	6	7
Profits:	P	10	5	15	7	6	18	3
Weight:	W	2	3	5	7	1	4	1
Profit by weight:	$\frac{P}{W}$	$\frac{10}{2} = 5$	$\frac{5}{3} = 1.66$	$\frac{15}{5} = 3$	$\frac{7}{7} = 1$	$\frac{6}{1} = 6$	$\frac{18}{4} = 4.5$	$\frac{3}{1} = 3$

Knapsack fractional: A maximization problem!

Objects:	o	1	2	3	4	5	6	7
Profits:	P	10	5	15	7	6	18	3
Weight:	w	2	3	5	7	1	4	1
Profit by weight:	$\frac{P}{W}$	5	1.66	3	1	6	4.5	3

How much should we take?

0 < amount <1

Knapsack fractional: A maximization problem!

Objects:	o	1	2	3	4	5	6	7	Bag size: 15
Profits:	P	10	5	15	7	6	18	3	$15-1=14$
Weight:	w	-2	<u>3</u>	-5	7	-1	-4	-1	$14-2=12$
Profit by weight:	$\frac{P}{w}$	5	1.66	3	1	6	4.5	3	$12-4=8$
		1	2/3	1	0	1	1	1	$8-5=3$
									$3-1=2$

Knapsack fractional: A maximization problem!

7 objects, 15 bag size

Objects: O	1	2	3	4	5	6	7
Profits: P	10	5	15	7	6	18	3
Weight: W	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
x	1	2/3	1	0	1	1	1

Total weight: $1*2 + 2/3*3 + 1*5 + 0*7 + 1*1 + 1*4 + 1*1 = 15$

Total profit: $1*10 + 2/3*5 + 1*15 + 0*7 + 1*6 + 1*18 + 1*3 = 55.33$

Knapsack fractional

- ▶ The algorithm sorts the items by their value-to-weight ratio in descending order.
- ▶ Time/Space depends on the sorting algorithm.
 - Tim sort:
 - Time: $O(n \log n)$
 - Space: $O(n)$
- ▶ Knapsack 0/1 is a different problem:
 - Each item can either be taken as a whole or left out entirely

We can still do
better!



Dynamic programming



The Romanesco flower!

- ▶ Romanesco has flowers that form perfect geometric spiral patterns that repeat.
- ▶ Each spiral bud is composed of a series of smaller buds
 - All buds are arranged in yet another spiral that continues at smaller levels
- ▶ The number of spirals on Romanesco's head follows the **Fibonacci sequence!**



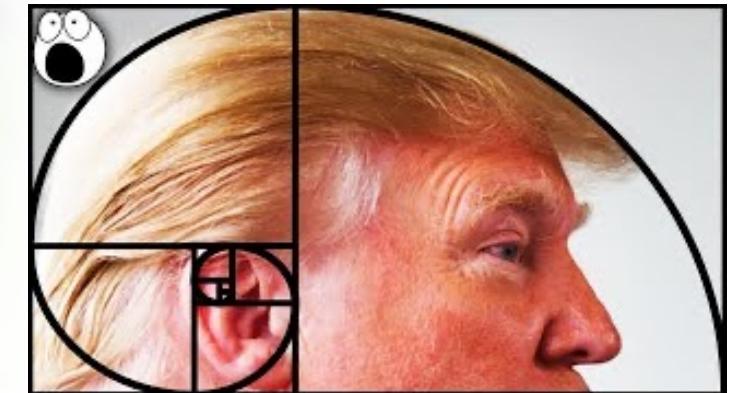
Other examples

Fibonacci symmetry
algorithm underlies our
perception of attractiveness





Reflects the
growth processes
in nature



What is the Fibonacci sequence?

- ▶ The Fibonacci Sequence is a series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

- ▶ The sequence starts with **zero** and **one** and is a steadily increasing series where each number equals the sum of the preceding two numbers.
- ▶ Example: Fibonacci F_3 is found by adding $F_2 + F_1$

$$F_3 = F_2 + F_1 = 1+1 = 2$$

$$F_2 = F_1 + F_0 = 1+0 = 1$$

$$F_1 = 1 \quad F_0 = 0$$

Quiz!

- ▶ What is the Fibonacci of 7?
 - Can you do it without pen and paper? ☺

$$F_3 = F_2 + F_1 = 1+1 = 2$$

$$F_2 = F_1 + F_0 = 1+0 = 1$$

$$F_1 = 1 \quad F_0 = 0$$

$$F_0 = 0$$

$$F_1 = 1$$

$$F_2 = 1$$

$$F_3 = 2$$

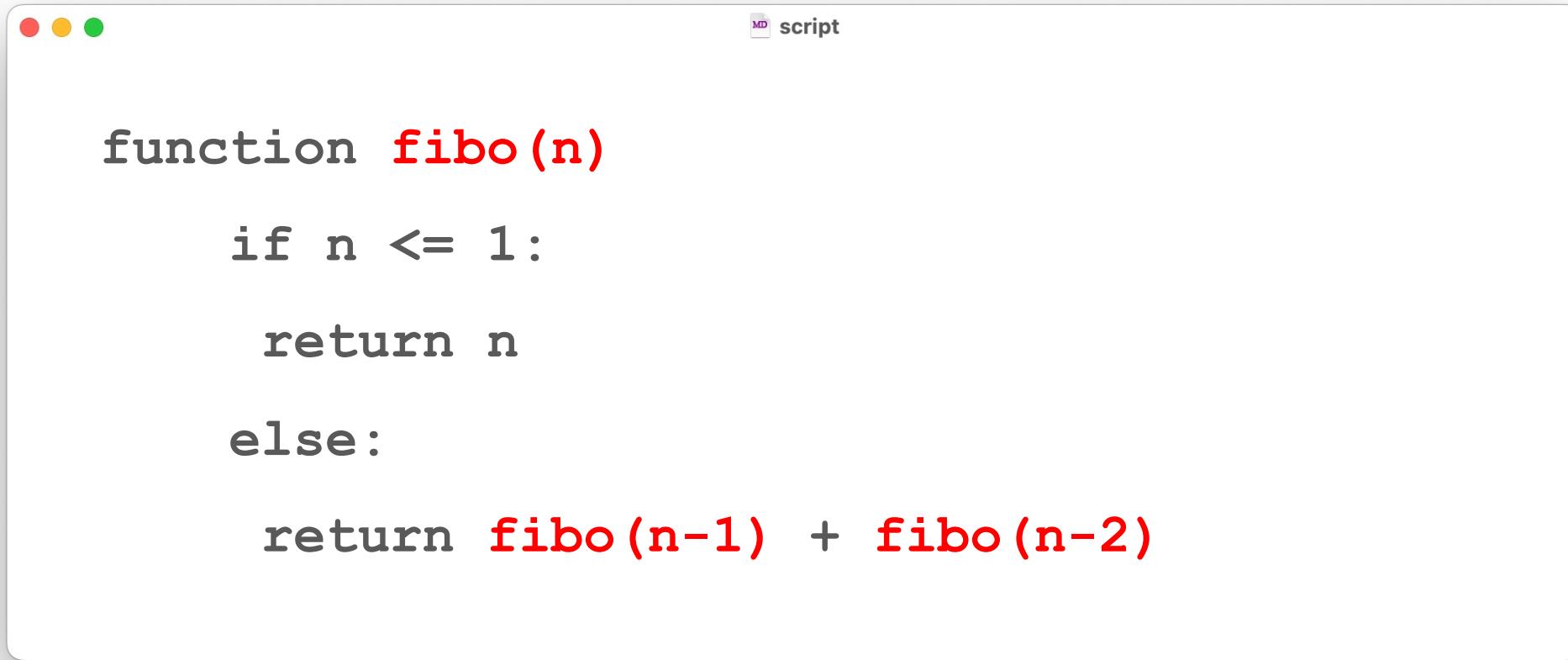
$$F_4 = 3$$

$$F_5 = 5$$

$$F_6 = 8$$

$$\mathbf{F_7 = 13}$$

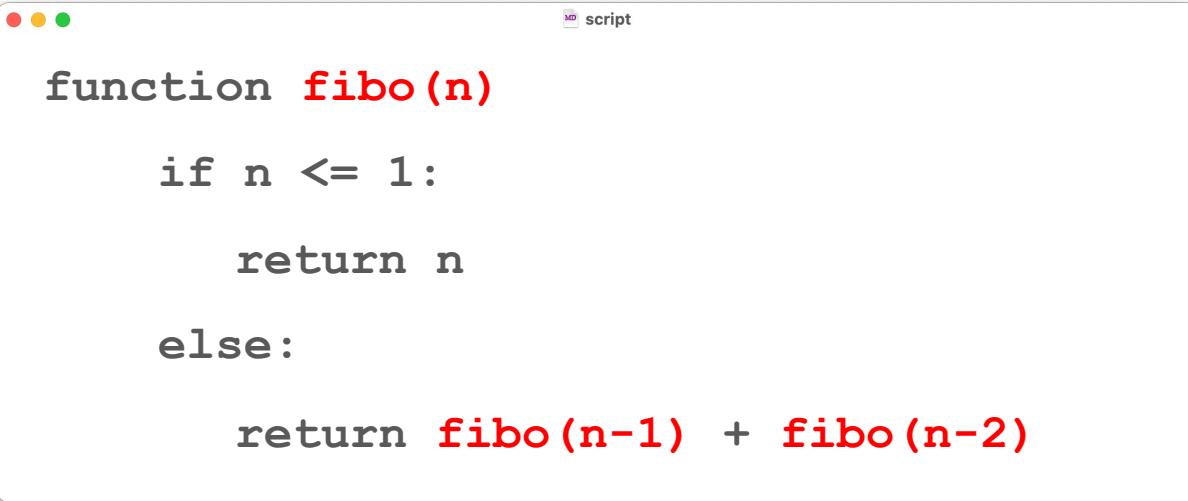
How to calculate the Fibonacci sequence?



A screenshot of a code editor window titled "script". The window has a light gray background and rounded corners. In the top-left corner, there are three small colored circles (red, yellow, green). The title bar "script" is located at the top right. The code itself is written in Python:

```
function fibo(n)
    if n <= 1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```

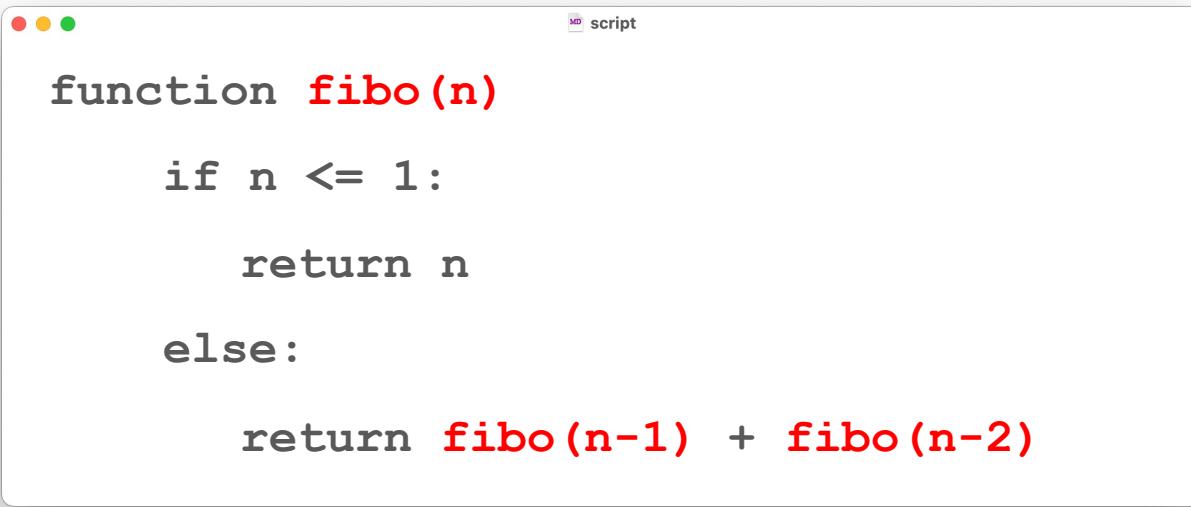
How to calculate the Fibonacci sequence?



```
function fibo(n)
    if n <= 1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```

- ▶ If $n = 0$, then what is the **fibo(0)**?
 - Answer: 0
- ▶ If $n = 1$, then what is the **fibo(1)**?
 - Answer: 1

How to calculate the Fibonacci sequence?

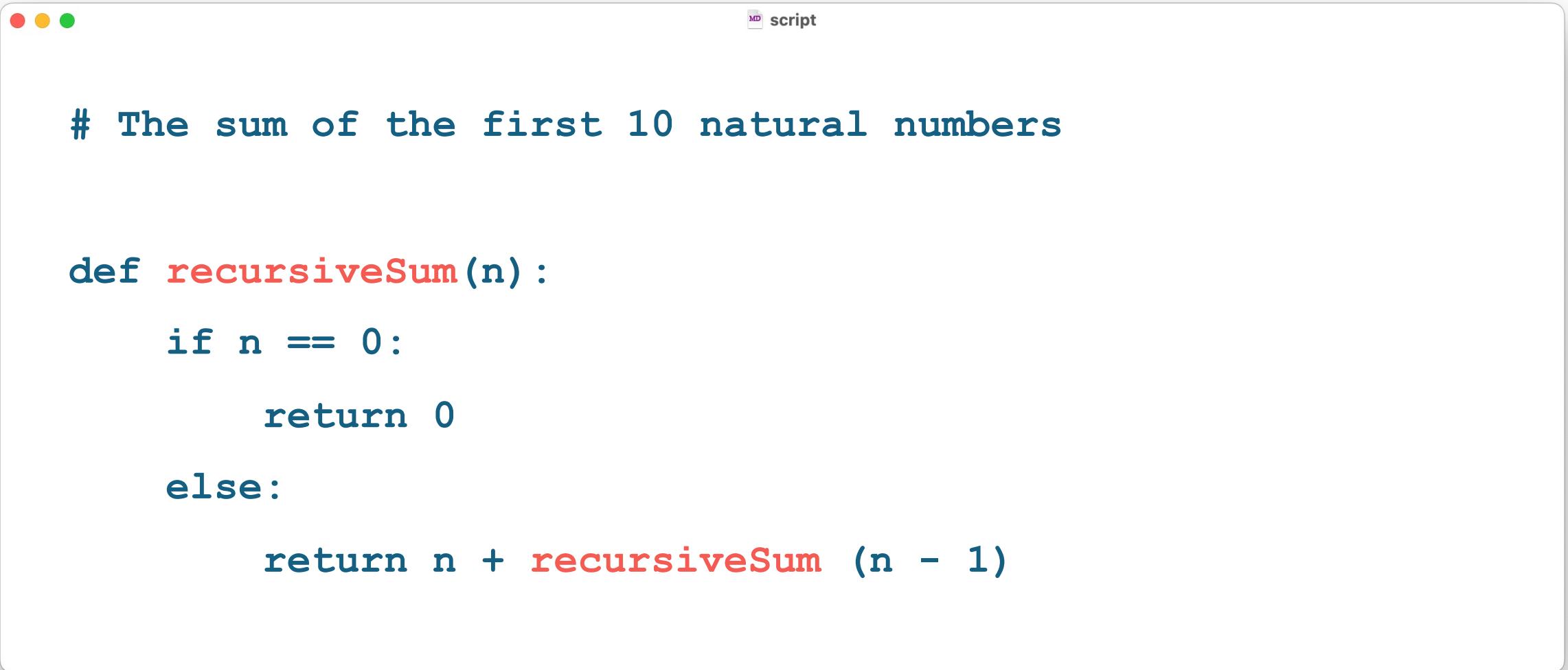


```
function fibo(n)
    if n <= 1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```

- ▶ If $n = 2$, then what is the **fibo(2)**?
 - Answer: $\text{fibo}(1) + \text{fibo}(0)$
 $= 1 + 0 = 1$

- ▶ If $n = 3$, then what is the **fibo(3)**?
 - Answer: $\text{fibo}(2) + \text{fibo}(1)$
 $= 1 + 1 = 2$

What is recursion in programming?



A screenshot of a code editor window titled "script". The window has three colored window control buttons (red, yellow, green) at the top-left. The code inside the editor is as follows:

```
# The sum of the first 10 natural numbers

def recursiveSum(n):

    if n == 0:

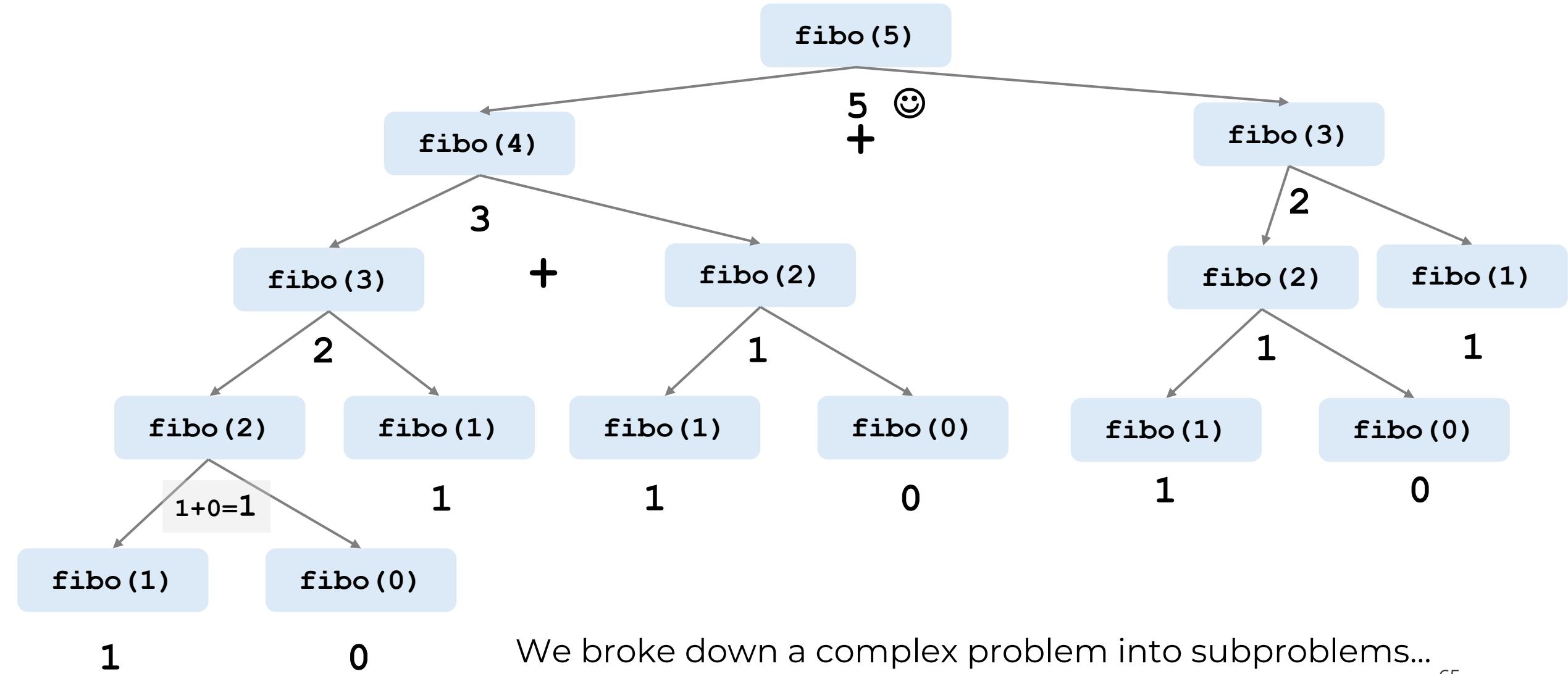
        return 0

    else:

        return n + recursiveSum (n - 1)
```

Let's see in practice!

We solve it recursively!

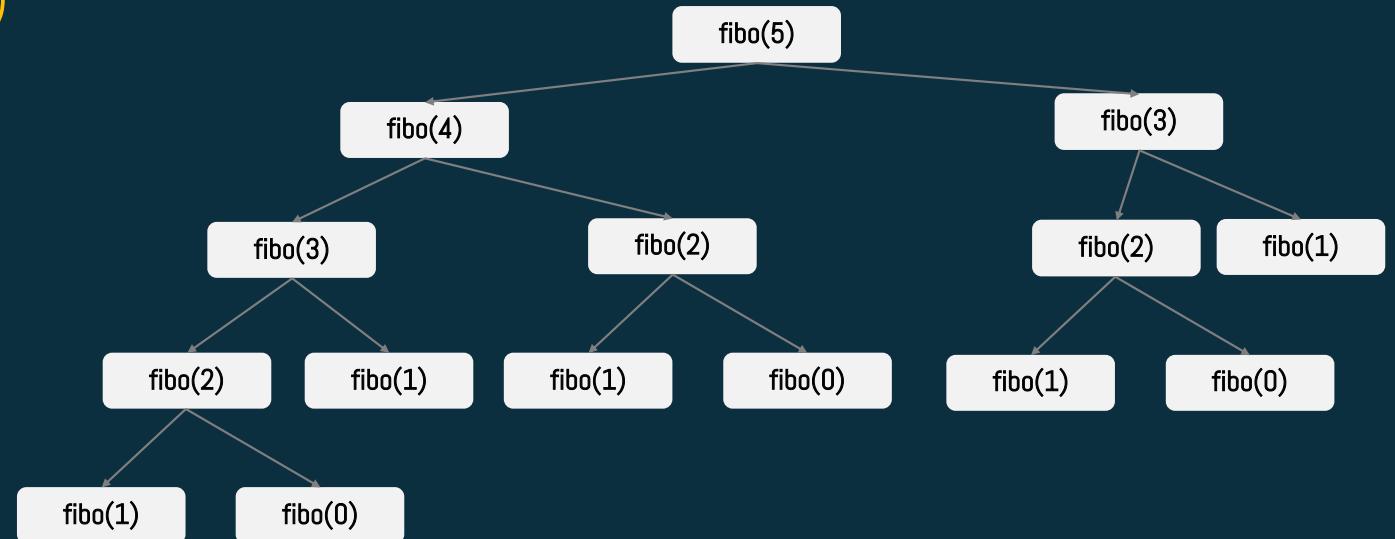


Quiz!

- ▶ What about computational complexity?
 - What is the time complexity of the Fibonacci method we just explored?
 - Answer: $O(2^n)$ (exponential)

- ▶ Is this good?

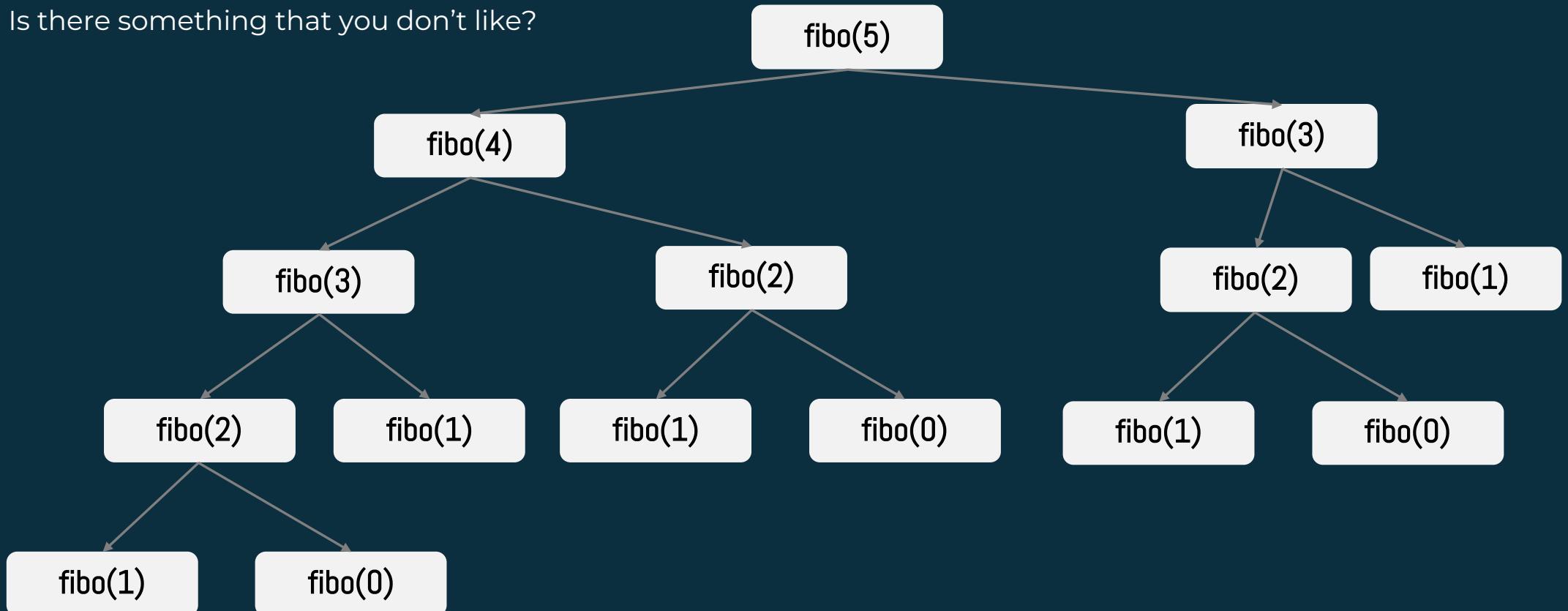
- Answer: $O(No!)$



Quiz!

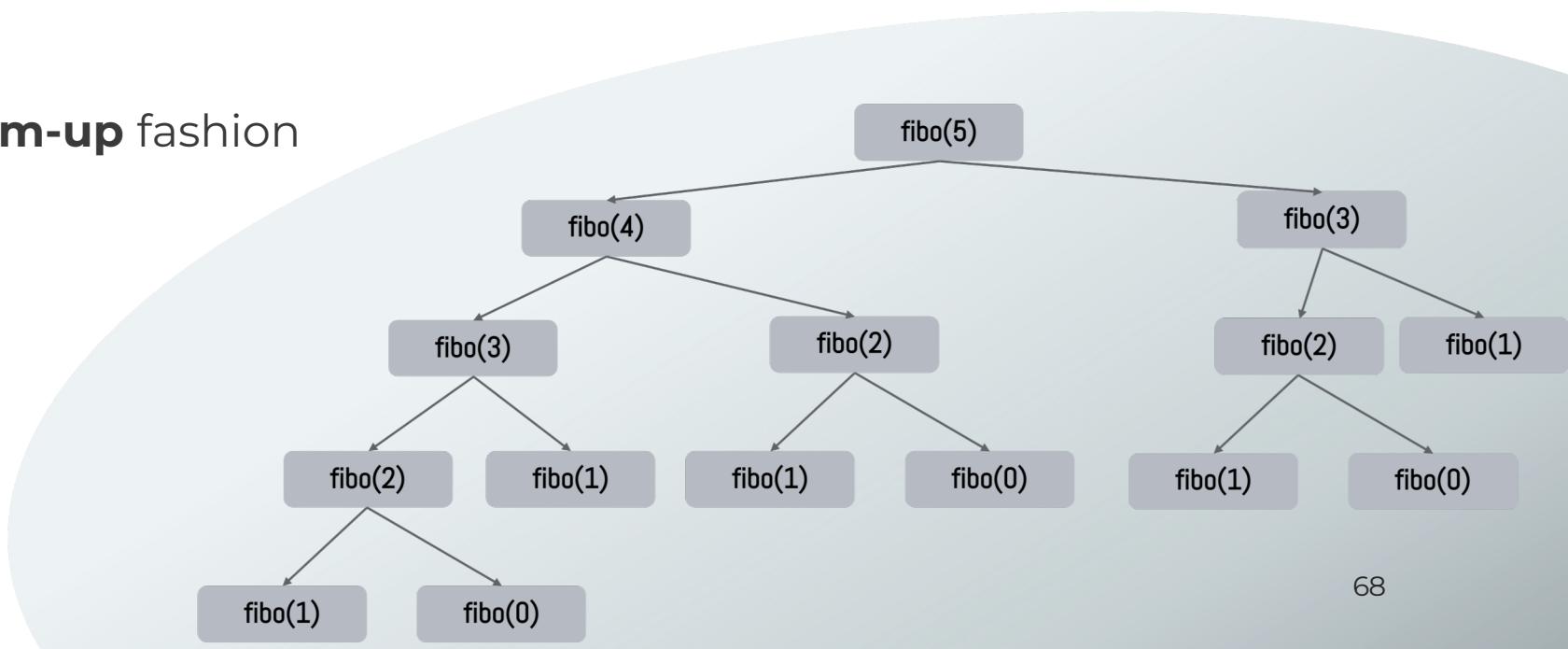
- ▶ Can you do better?

- Is there something that you don't like?



Can we do it better?

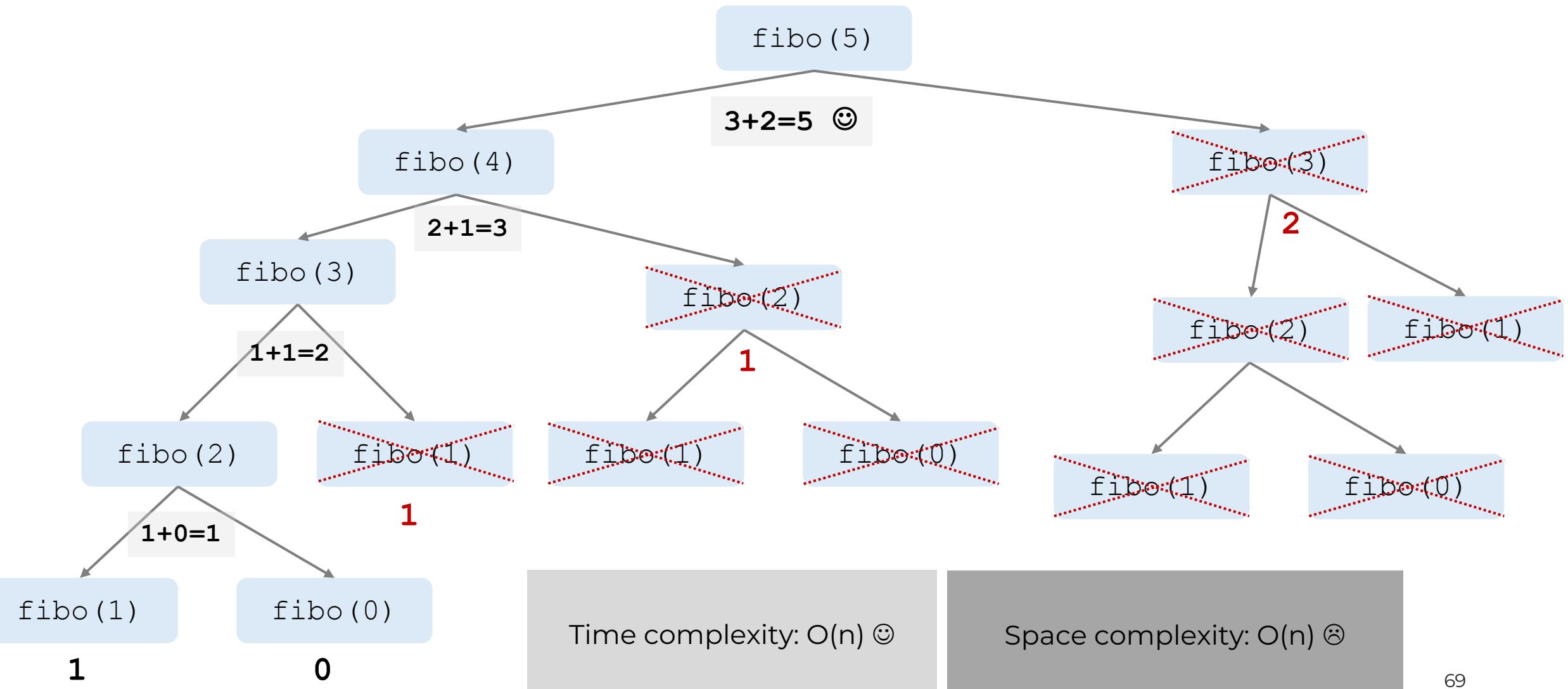
- ▶ **What if we avoid calculating the same Fibonacci numbers again and again?**
- ▶ What if we memorise the results of an existing calculation and use it in the next iteration?
 - Let's solve it in a **bottom-up** fashion



0	1	1	2	3	5
---	---	---	---	---	---

Let's use dynamic programming!

0 1 2 3 4 5



Let's recap

- ▶ Dynamic programming is a technique that breaks problems into sub-problems 😊
 - But it stores intermediate results, and that's a tradeoff 😞
 - There is no guarantee that the stored value will be used later in the execution

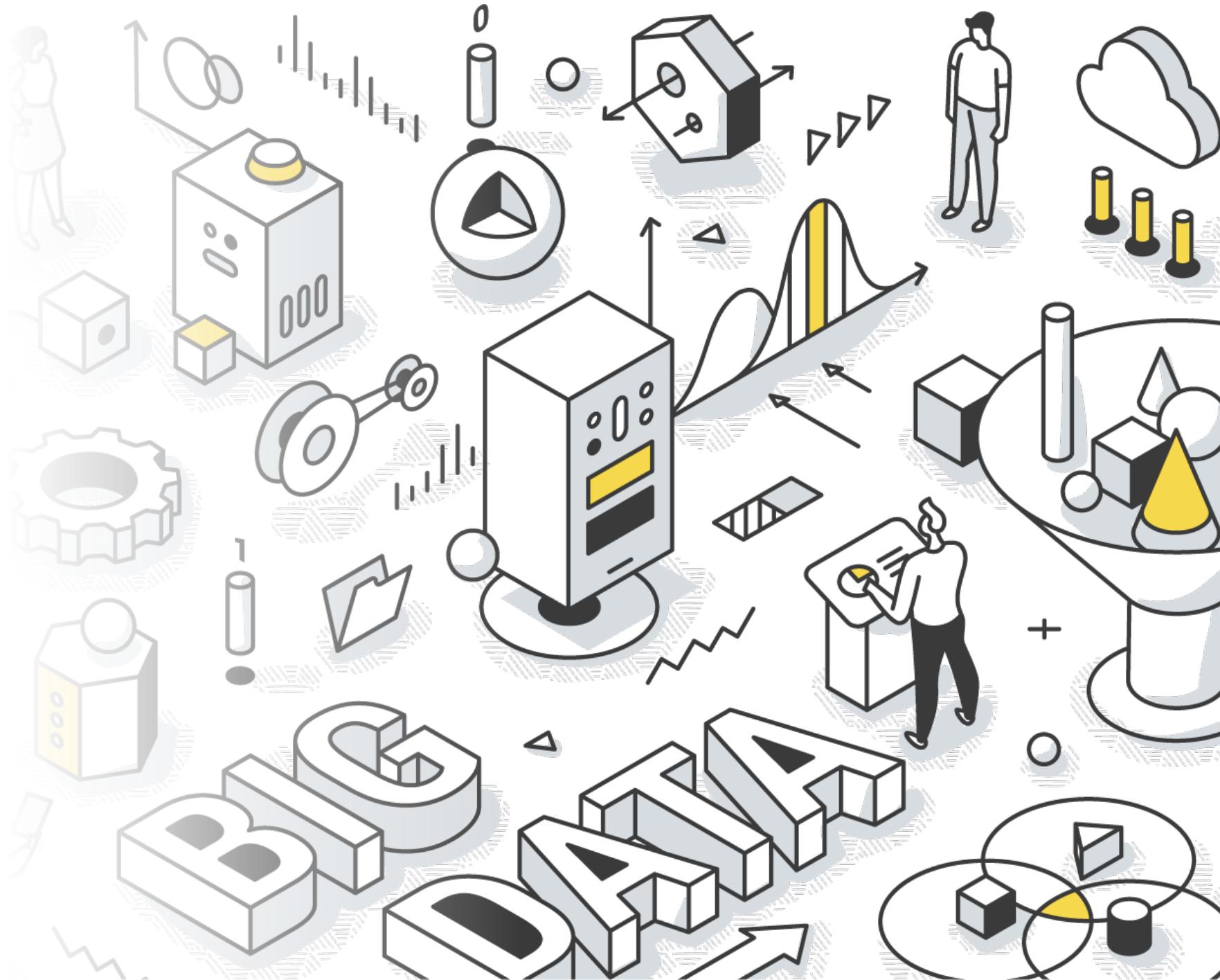
Dynamic programming vs. Greedy

- ▶ **Divide and conquer** breaks down a problem into two or more sub-problems
 - Until these become simple enough to be solved directly.
- ▶ **Greedy methods** (top-down) solve a problem as quickly as possible.
 - Make whatever choice seems best now and then solve the sub-problems arising after.
- ▶ **Dynamic programming** (bottom-up) solves a problem as efficiently as possible.
 - Store the result for future purposes so you do not need to compute it again.

Thank you!

- ▶ The lab starts soon!
(404-405)

O(no!)



Lab 2

Big Data Analytics

Lab activities

- ▶ Complete lab 2 activities and exercises.
- ▶ Use your preferred python IDE.