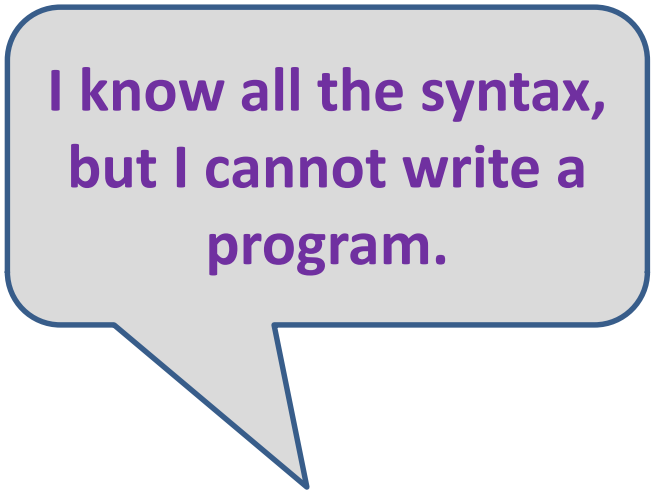
Computer Science & Programming

Lecture 3: Computational Thinking

Stephen Huang

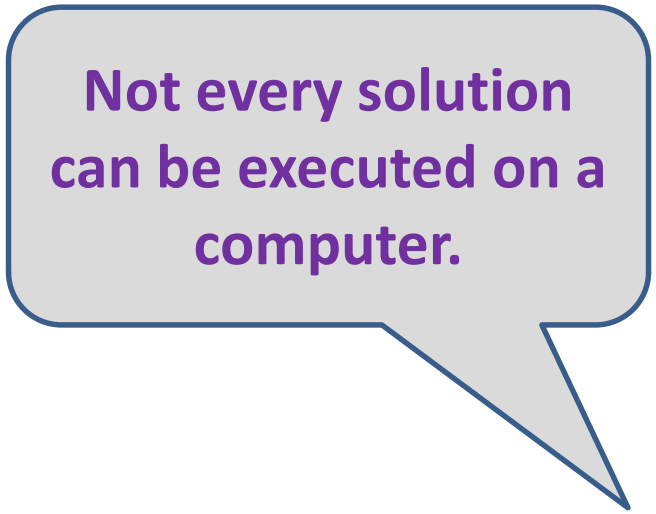
January 23, 2023

Contents



1. Computational Thinking

- Computational thinking is the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that an information-processing agent can effectively carry out.

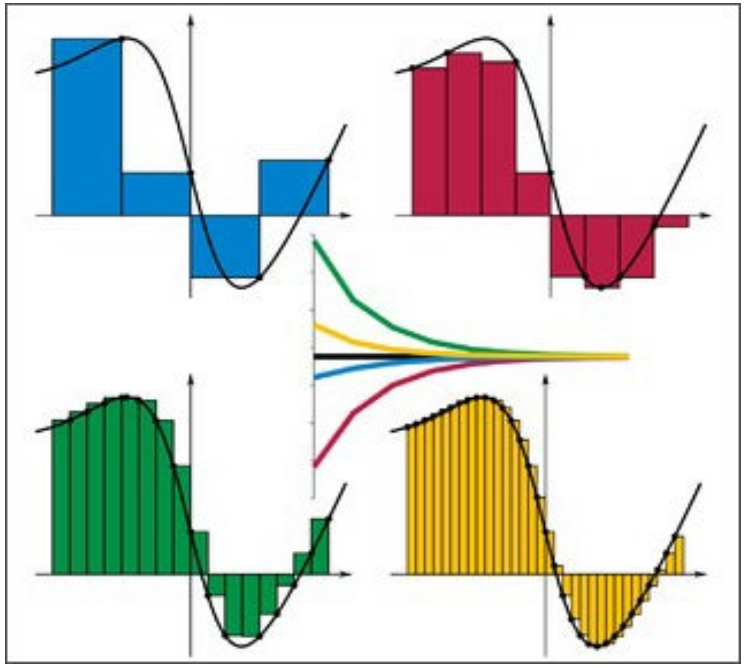


Computational Thinking?

- Symbolically in Calculus

$$A = \int_0^2 (x - \frac{1}{2}x^2)\,dx = \left[\frac{1}{2}x^2 - \frac{1}{6}x^3\right]_{x=0}^{x=2} = \frac{2}{3}$$

- Computationally



https://isquared.digital/blog/2020-05-27-riemann-integration/

What does CT allow us to do?

- Computational thinking allows us to take a complex problem, understand the problem, and develop possible solutions.
- We can then present these solutions so that a computer, a human, or both, can understand.
- Turning a complex problem into one we can easily understand is an extremely useful skill, programming or otherwise.

CT is not ...

- Programming
- Thinking in binary
- Thinking like a computer

What is CT?

- “Computational Thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to Computer Science.”

  – Jeannette M. Wing, CACM, Vol. 49, no. 3, March 2006, pp.33-35.

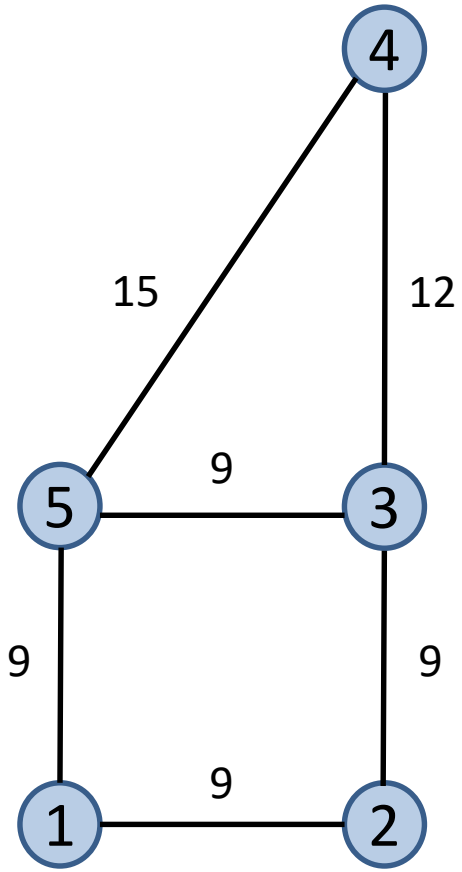
CT is

- Computational thinking is
  - reformulating a problem into one we know how to solve, by reduction, embedding, transformation, or simulation,
  - is thinking recursively,
  - using abstraction and decomposition when attacking a large complex task, or
  - using heuristic reasoning to discover a solution.

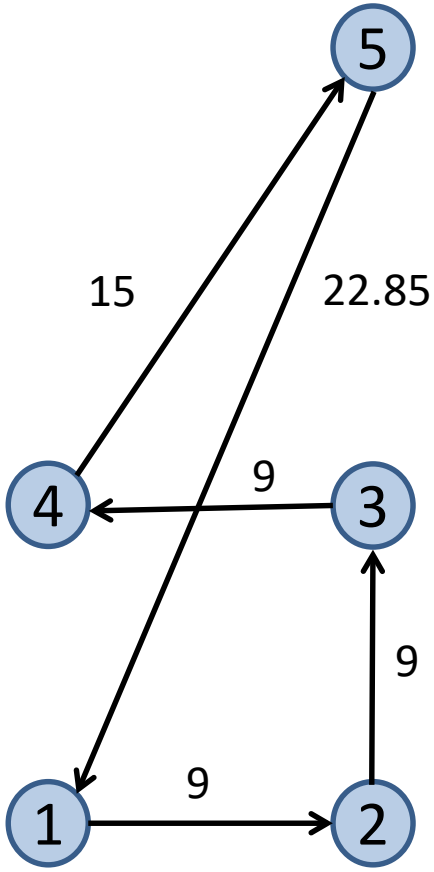
Traveling Salesman Problem

Starting from City 1, the salesman must travel to all cities once before returning home.

Minimize the total distance travelled.

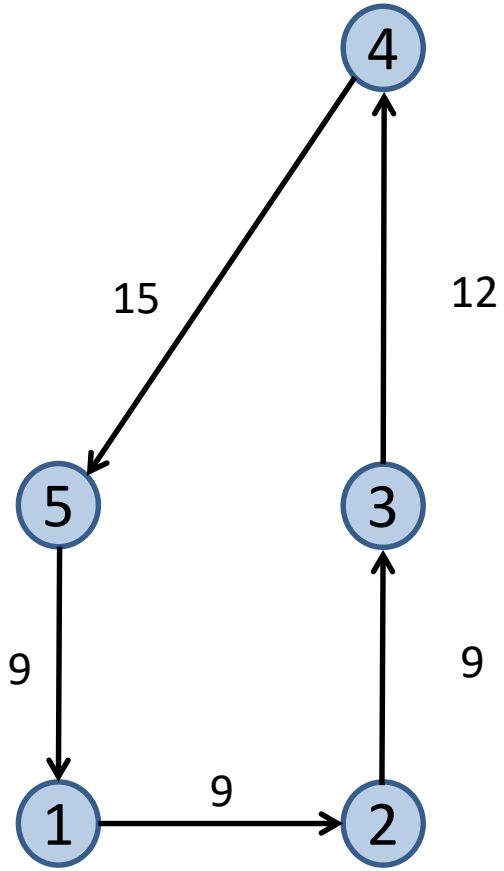


Heuristic Solution: TSP



Greedy

Nearest Neighbor Tour



Optimal

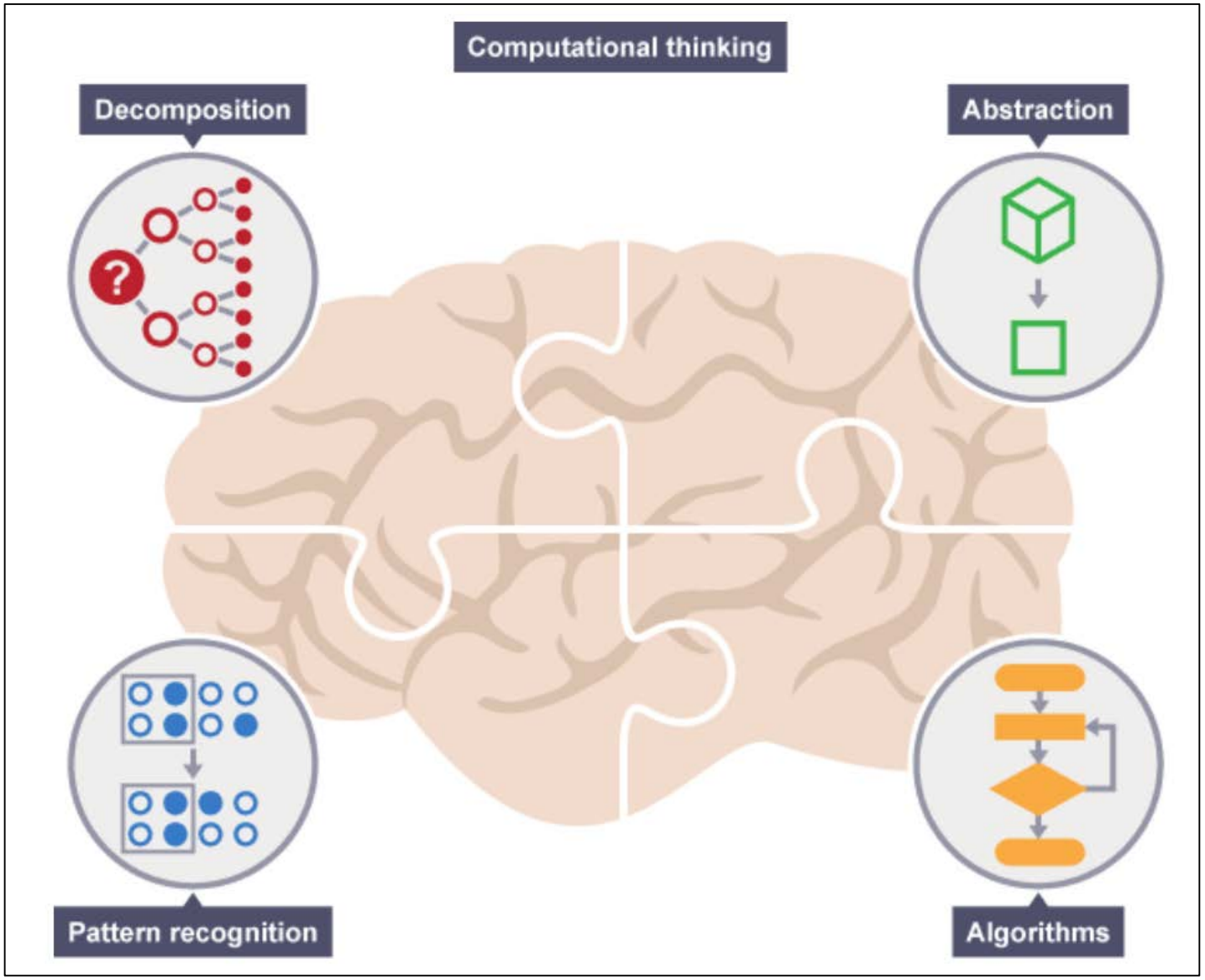
Characteristics

- Computational thinking thus has the following characteristics:
  - Conceptualizing, not programming.
  - Fundamental, not a rote skill.
  - A way that humans, not computers, think.
  - Complements and combines mathematical and engineering thinking.
  - Ideas, not artifacts.
  - For everyone, everywhere.

Key Concepts of CT

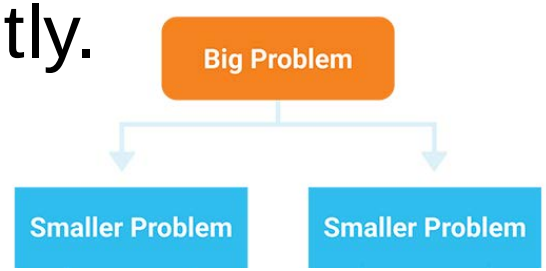
- **Problem Decomposition** - breaking down a complex problem or system into smaller, more manageable parts
- **Pattern Recognition** – looking for similarities among and within problems
- **Abstraction** – focusing on the important information only, ignoring irrelevant detail
- **Algorithm Design** - developing a step-by-step solution to the problem, or the rules to follow to solve the problem

CT

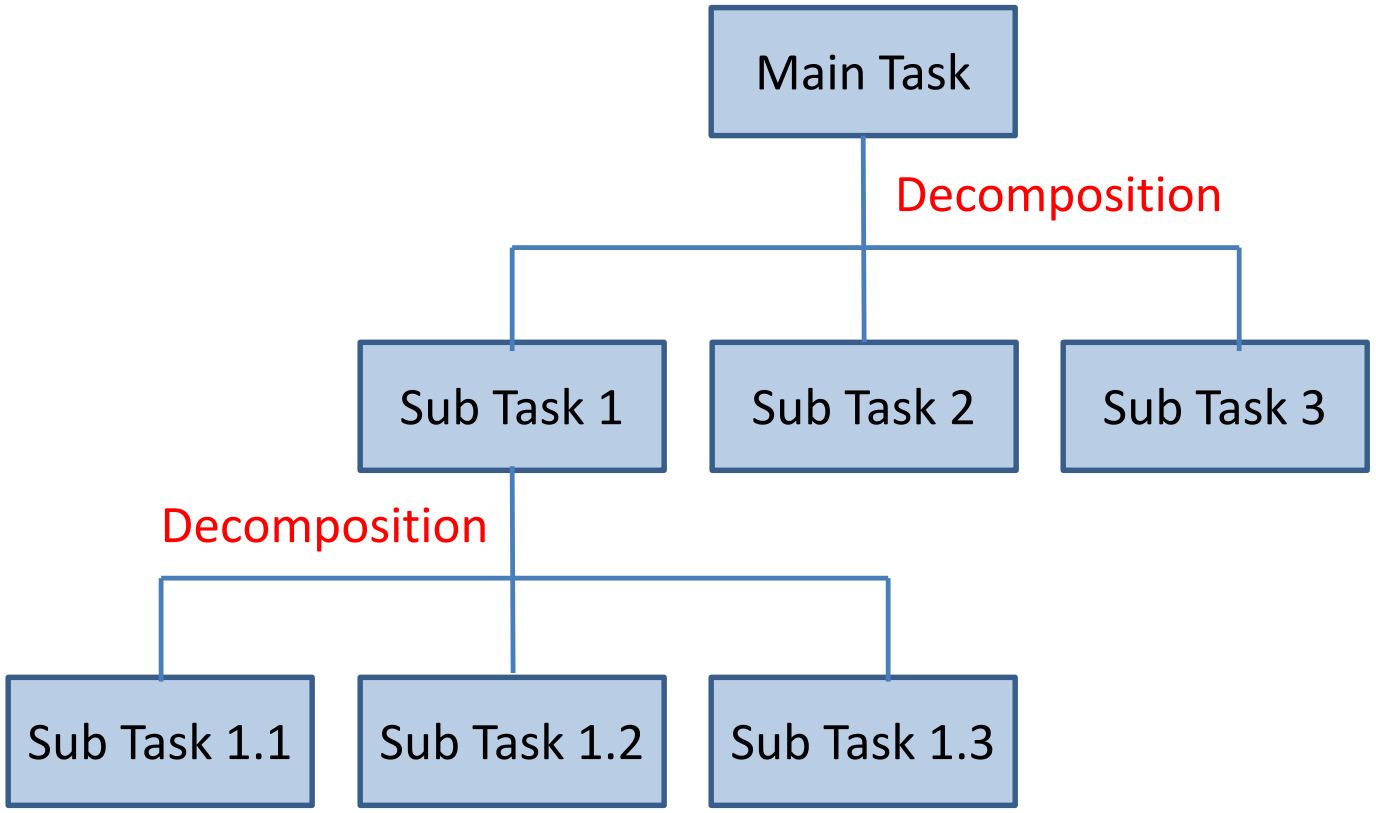

Computational thinking
Decomposition
Abstraction
Pattern recognition
Algorithms

a. Problem Decomposition

- The process of breaking down a complex problem into smaller, more manageable parts.
- Dividing a problem into smaller problems until they are small enough to be solved.
- The decomposition process can be used repeated, one level at a time, until the parts are small enough to be solved directly.



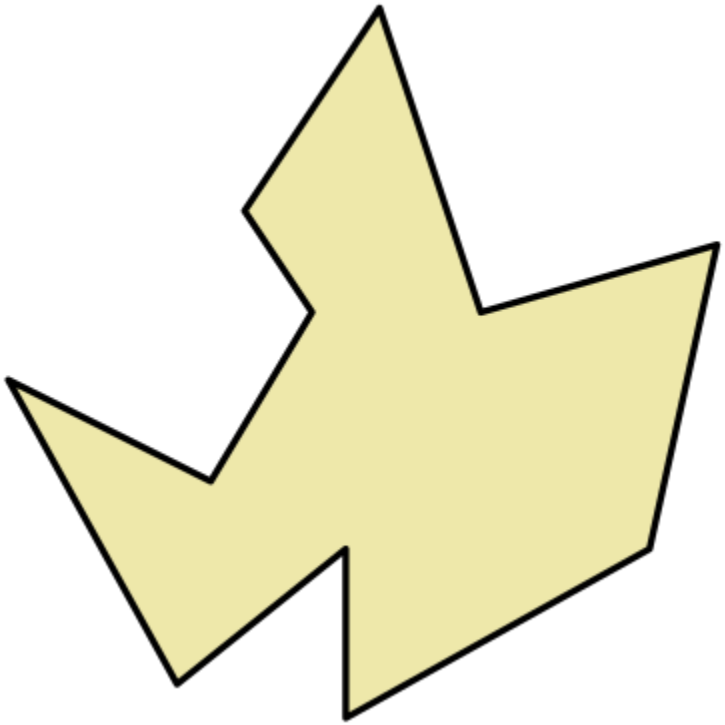
Decomposition



To solve this problem,

you must solve all these problems.

Decomposition of Polygon



Recursion

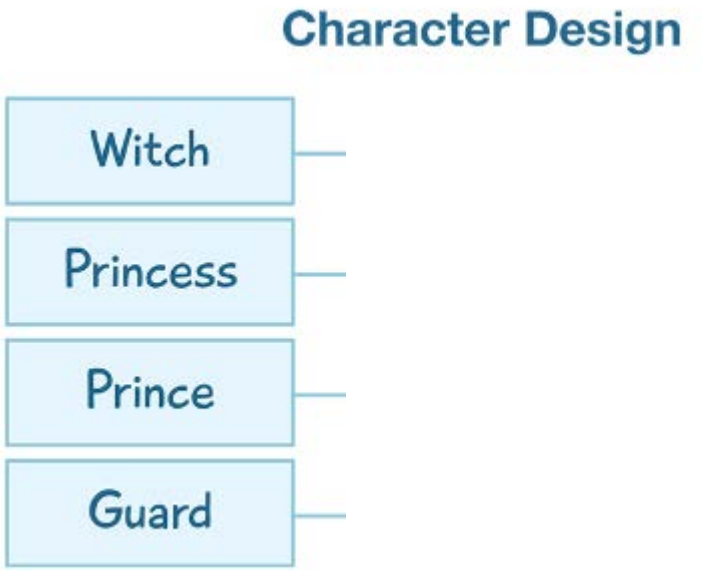
- Recursion is a special case of decomposition when a smaller part of problem is of the same type of the problem as the original one except the size of the problem is smaller.
- In this case, the same strategy can be used to solve the smaller problems until the smaller problems are small enough to be solved directly.

b. Pattern Recognition

- Patterns can help us to solve complex problems more efficiently.
- Finding the similarities (or patterns) of several problems.
- Patterns may exist among different problems or within individual problems.
- Example: Sorting numbers vs. words.

Pattern Recognition

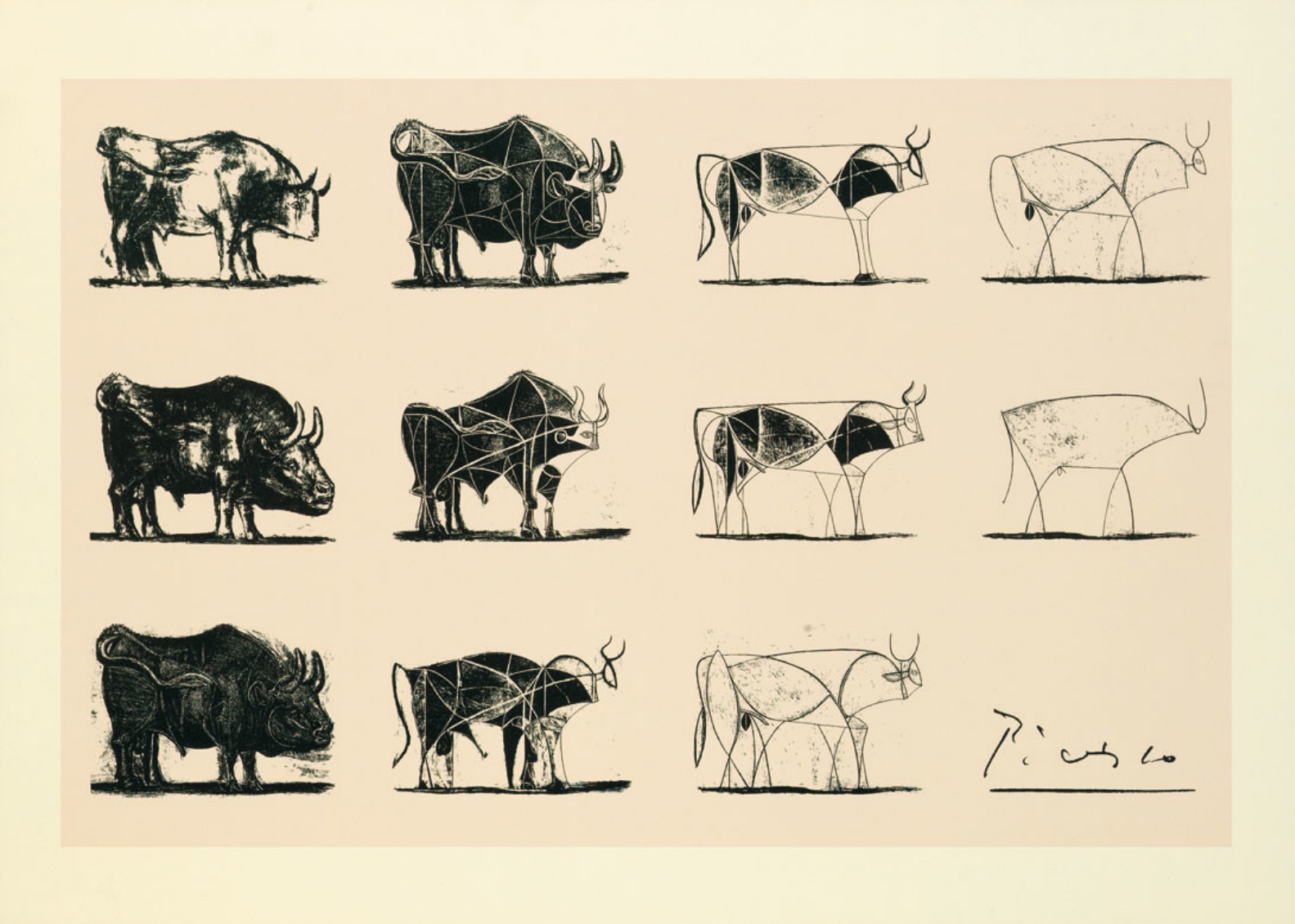
- Pattern recognition is all about recognizing patterns.



c. Abstraction

- This process of filtering out the extraneous and irrelevant pieces of information to identify what's most important and connects each decomposed problem.

Picasa's Bulls



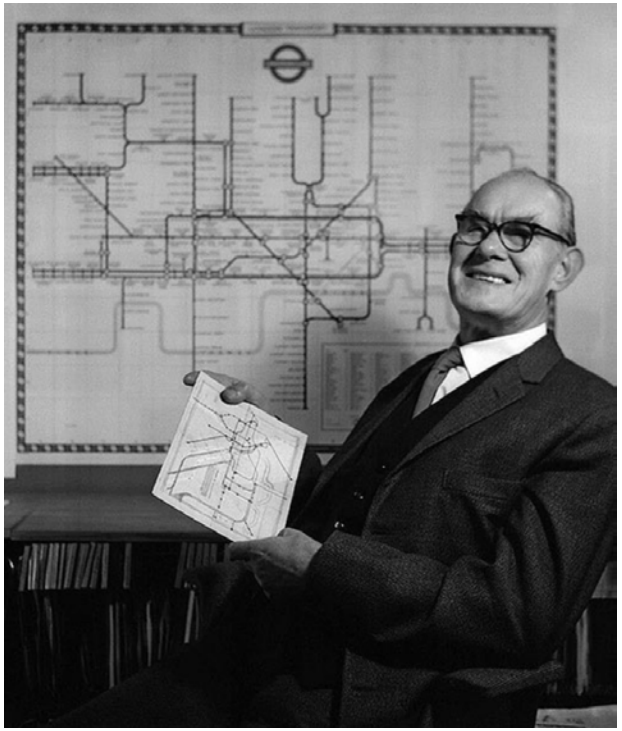
https://www.linkedin.com/pulse/picasso-his-bulls-lesson-simplicity-anna-sundt-1/

Abstraction

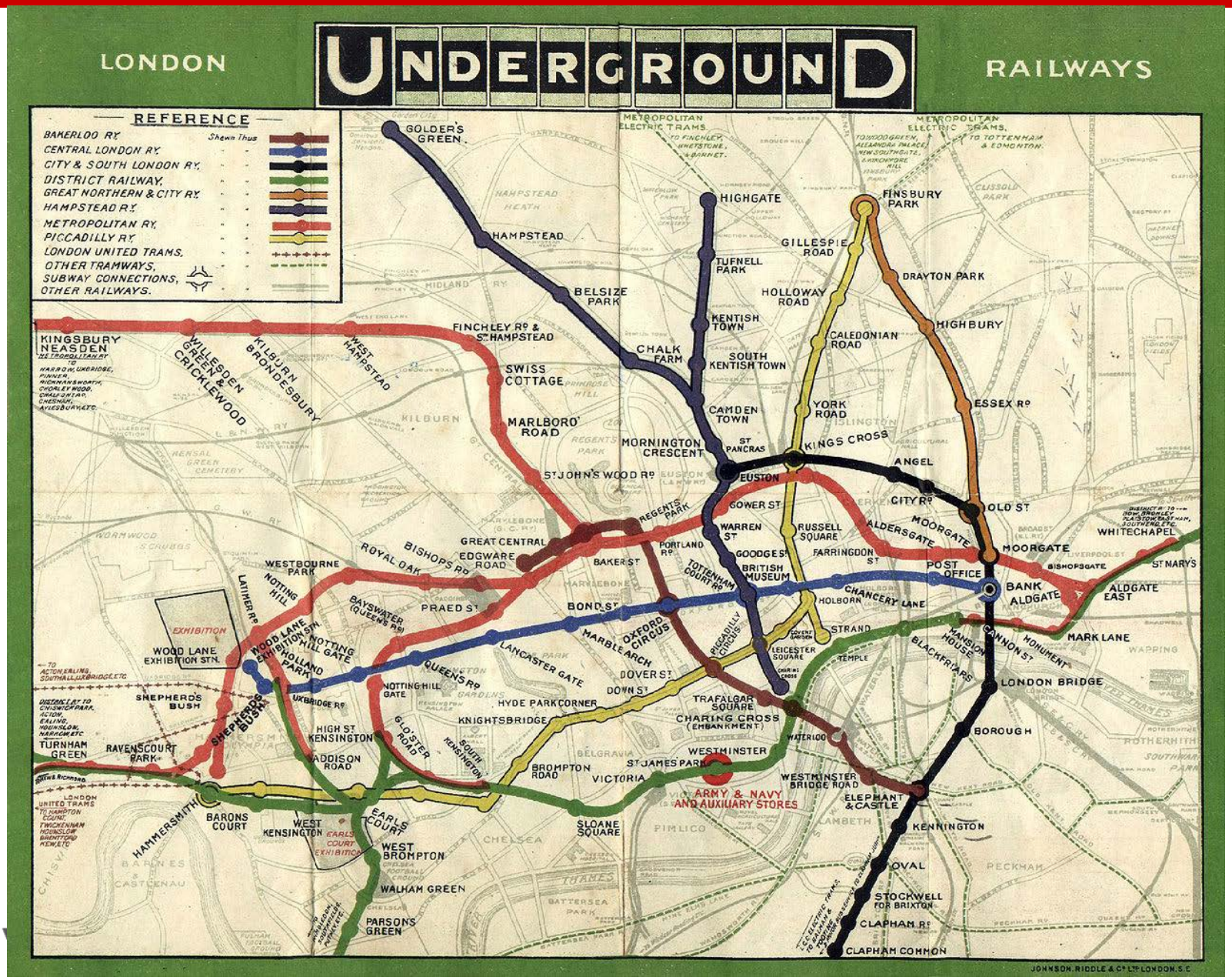
- Filtering out the characteristics that we don't need to concentrate on those that we do.
- From this, we create a representation of what we are trying to solve. This representation is known as a ‘model’.
- Abstraction allows us to create a model of the problem so we can solve it.
- Once we have a model of our problem, we can then design an algorithm to solve it.

Topological Map

- Henry Charles Beck (1902-74) created the London Underground topological map in 1931.



London Tube 1908



1933

1990


Key to lines
Bakerloo
Central
Circle
District
East London
Hammersmith & City
Jubilee
Metropolitan
Northern
Piccadilly
Victoria
Docklands Light Railway†
Network SouthEast
Restricted service
Peak hours and Sunday mornings
Peak hours only
Under construction
Key to symbols
Interchange stations
Connections with British Rail
Connections within walking distance
Closed Sundays
Closed Saturdays and Sundays
Served by Piccadilly line Sundays and early morning and late evening Mondays to Fridays
Closed for reconstruction until early 1991. Use Cannon Street
Closed for reconstruction. Use Edgware Road (Circle/District/H&C)
† These stations are open at the following times only:
Aldwych Monday to Friday peak hours
Barbican Mondays to Fridays all day. Saturdays 07 15 to 23 50. Sundays 08 15 to 23 50
Chancery Lane Exit only Mondays to Fridays 08 00 to 10 00 during escalator replacement. Closed Sundays
Chigwell Daily until 20 00
Docklands Light Railway Replaced by buses at weekends and after 21 30 Mondays to Fridays
Drayton Park and Essex Road Mondays to Fridays until 21 00 Closed Saturdays and Sundays
Grange Hill Daily until 20 00
Kensington (Olympia) Mondays to Saturdays 07 00 to 20 45. Sundays during exhibitions
North Weald Mondays to Saturdays until 22 00 Sundays 09 00 to 21 00
Ongar Mondays to Saturdays until 22 00 Sundays 09 00 to 21 00
Roding Valley Mondays to Fridays until 20 00 Closed Saturdays and Sundays
Shoreditch Monday to Friday peak hours. Sundays 08 00 to 14 30
Waterloo & City Line Mondays to Fridays 06 45 to 21 50. Saturdays 07 15 to 13 40. Closed Sundays
Certain stations are closed during public holidays
UNDERGROUND

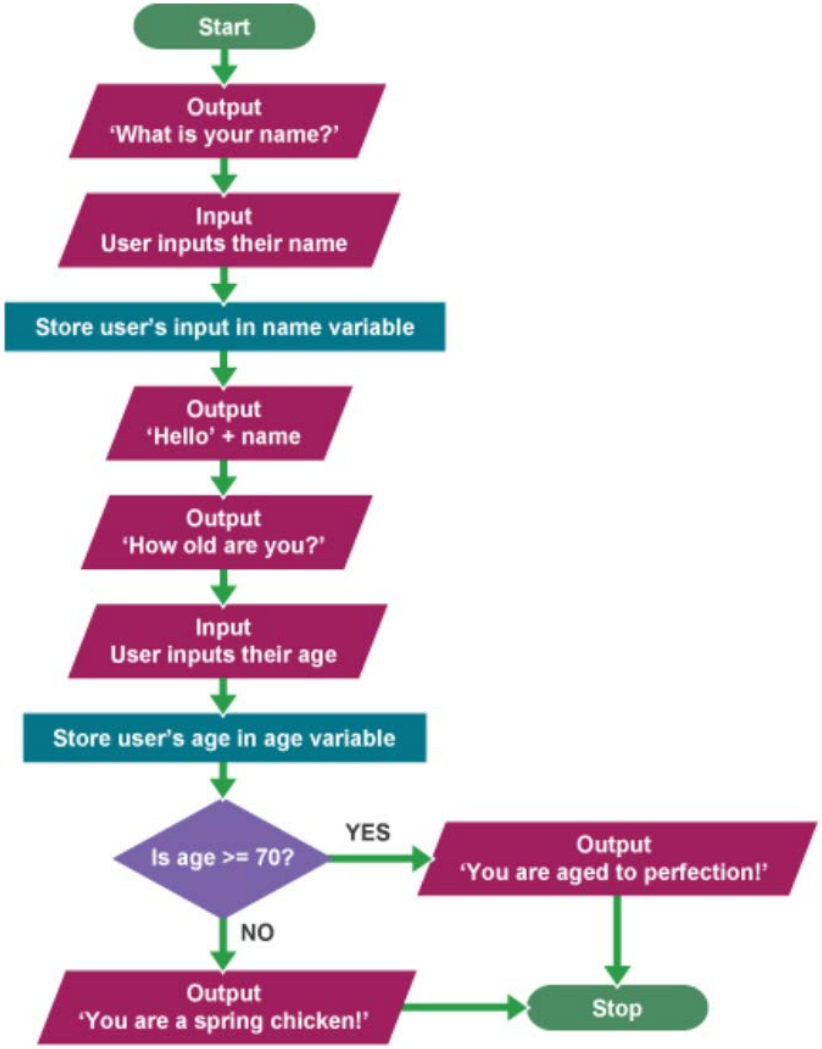
d. Algorithm Design/Thinking

- A process that automates the problem-solving process by creating a series of systematic, logical steps that
  - intake a defined set of inputs and
  - produce a defined set of outputs based on these.

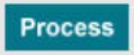
Algorithms

- An algorithm is a plan, a set of step-by-step instructions to resolve a problem.
- It must have a starting point, a finishing point, and a set of precise instructions.
- The plan can be presented as sentences, pseudocode, flowchart, or (but not necessarily) a program.
- Pseudocode is not a programming language; it is a simple way of describing a set of instructions that do not have to use specific syntax.

Flow Chart


Start
Output
'What is your name?'
Input
User inputs their name
Store user's input in name variable
Output
'Hello' + name
Output
'How old are you?'
Input
User inputs their age
Store user's age in age variable
Is age >= 70?
YES
Output
'You are aged to perfection!'
NO
Output
'You are a spring chicken!'
Stop

Flow Chart

Name	Symbol	Usage
Start or Stop		The beginning and end points in the sequence.
Process		An instruction or a command.
Decision		A decision, either yes or no.
Input or Output		An input is data received by a computer. An output is a signal or data sent from a computer.
Connector		A jump from one point in the sequence to another.
Direction of flow		Connects the symbols. The arrow shows the direction of flow of instructions.

Computationally?

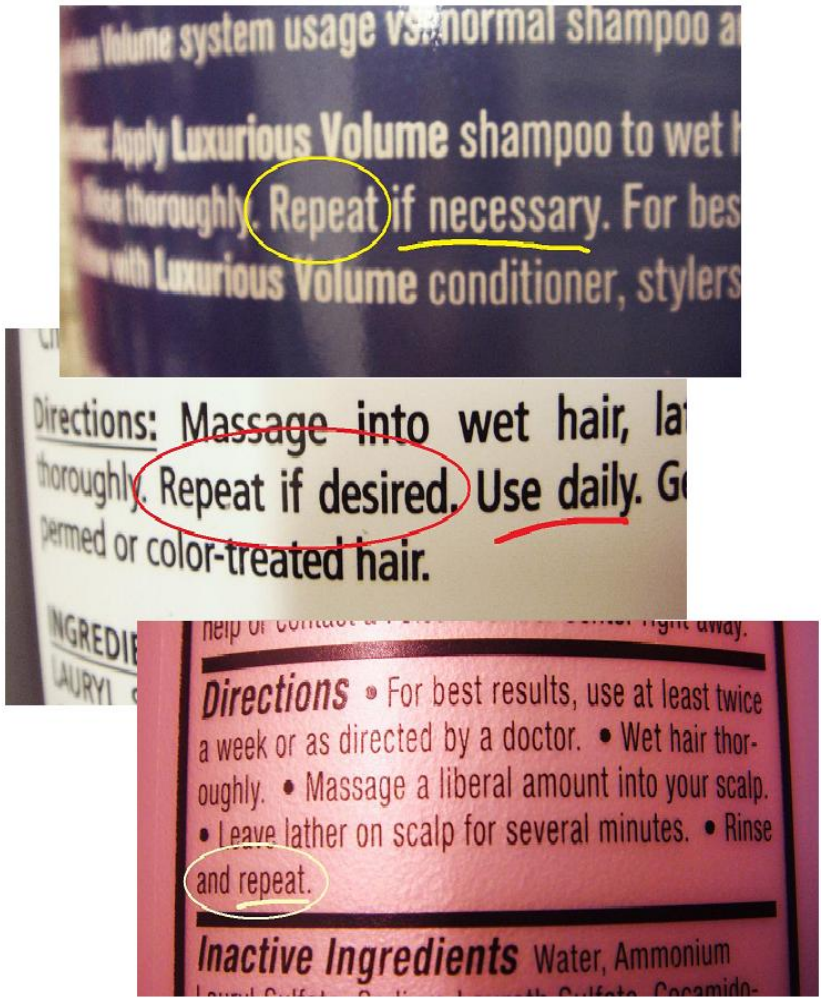
- To solve a problem computationally we must generate a solution in a series of precise steps.
- Algorithm:
  - A set of steps to accomplish a task.
  - A series of steps for a computer program to achieve a task.

Solving Problems

- Having to solve a particular problem, we might ask:
  - How difficult is it to solve? and
  - What's the best way to solve it?

Algorithmic Thinking

- Not algorithms:



Skills

- Thinking
  - Logically
  - Algorithmically
  - Recursively

Algorithms are used everyday

Useful Algorithms

- Google Map: finding the best route from one place to another.
- YouTube Video: compress a video so that you can download it faster.
- UPS: packing as many boxes into a truck for delivery.
- Amazon: suggesting a book that you may be interested in reading.

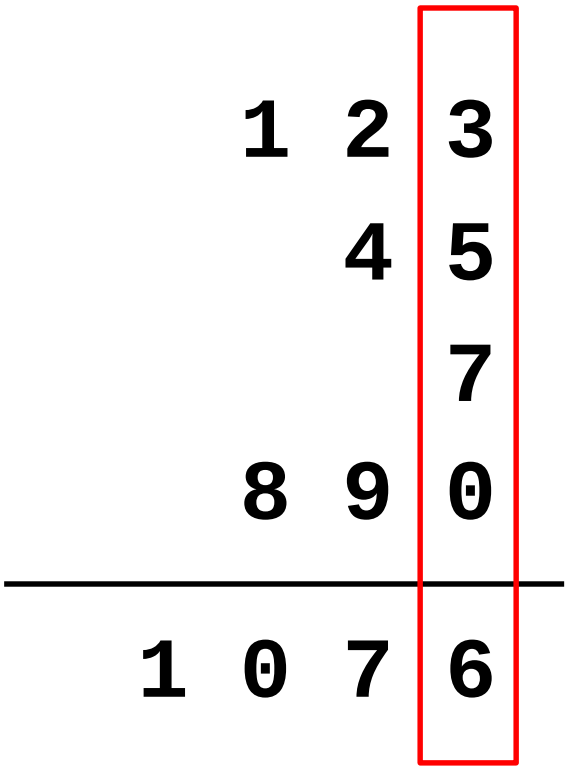
Others

- Structured organization, modularization, encapsulation

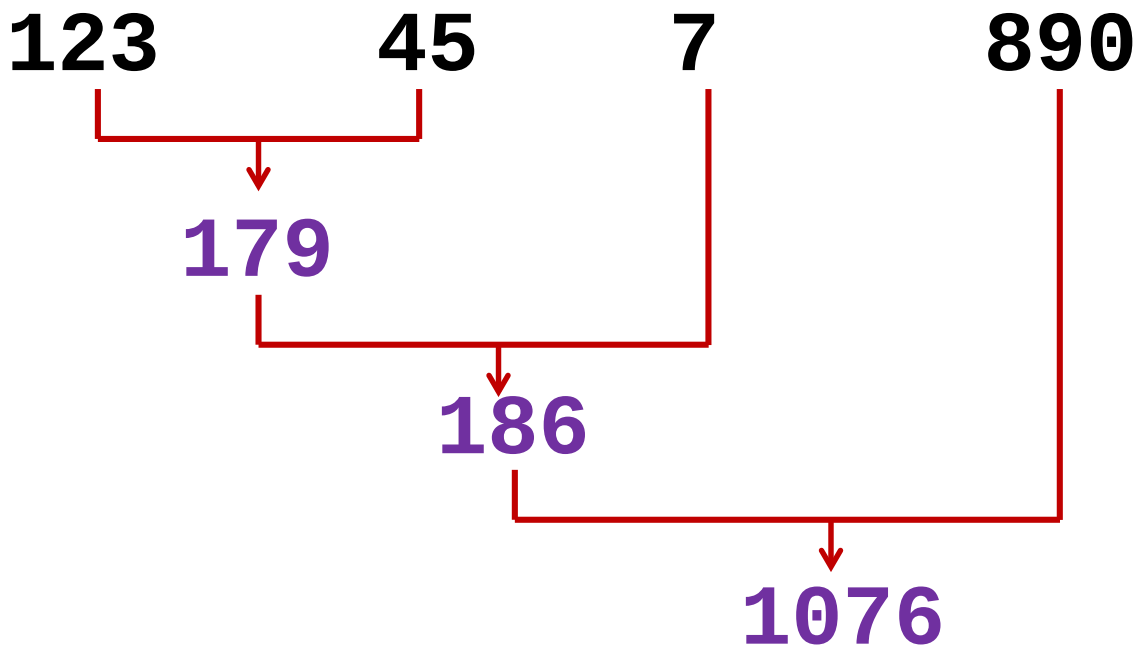
2. An Example

- The problem: given a list of numbers, add them up.
  - Works for any length (say, 1 or 1,000,000)
  - Computer can only do a small task at a time
  - Decomposition
  - How do I remember the “state” of my computation?

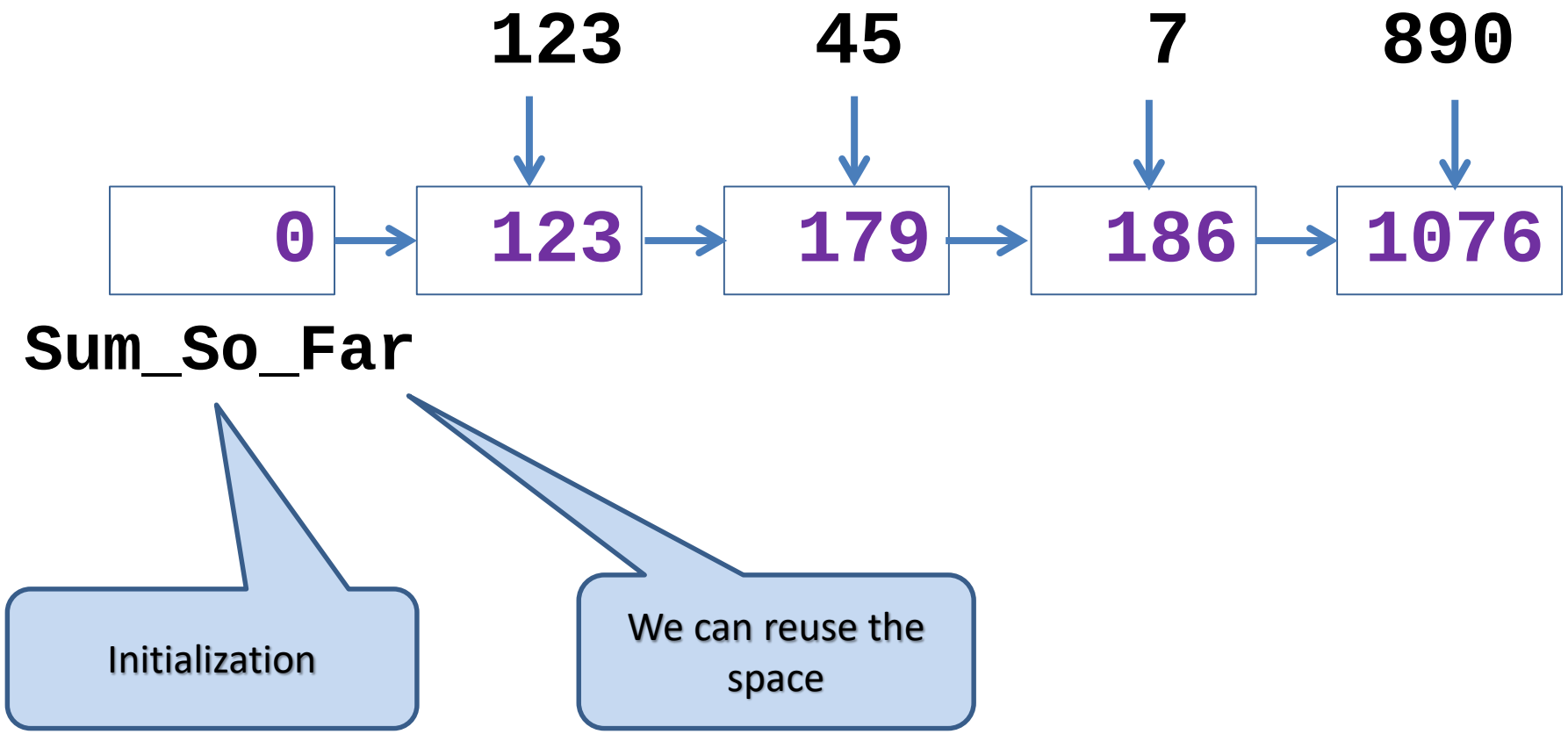
Decomposition



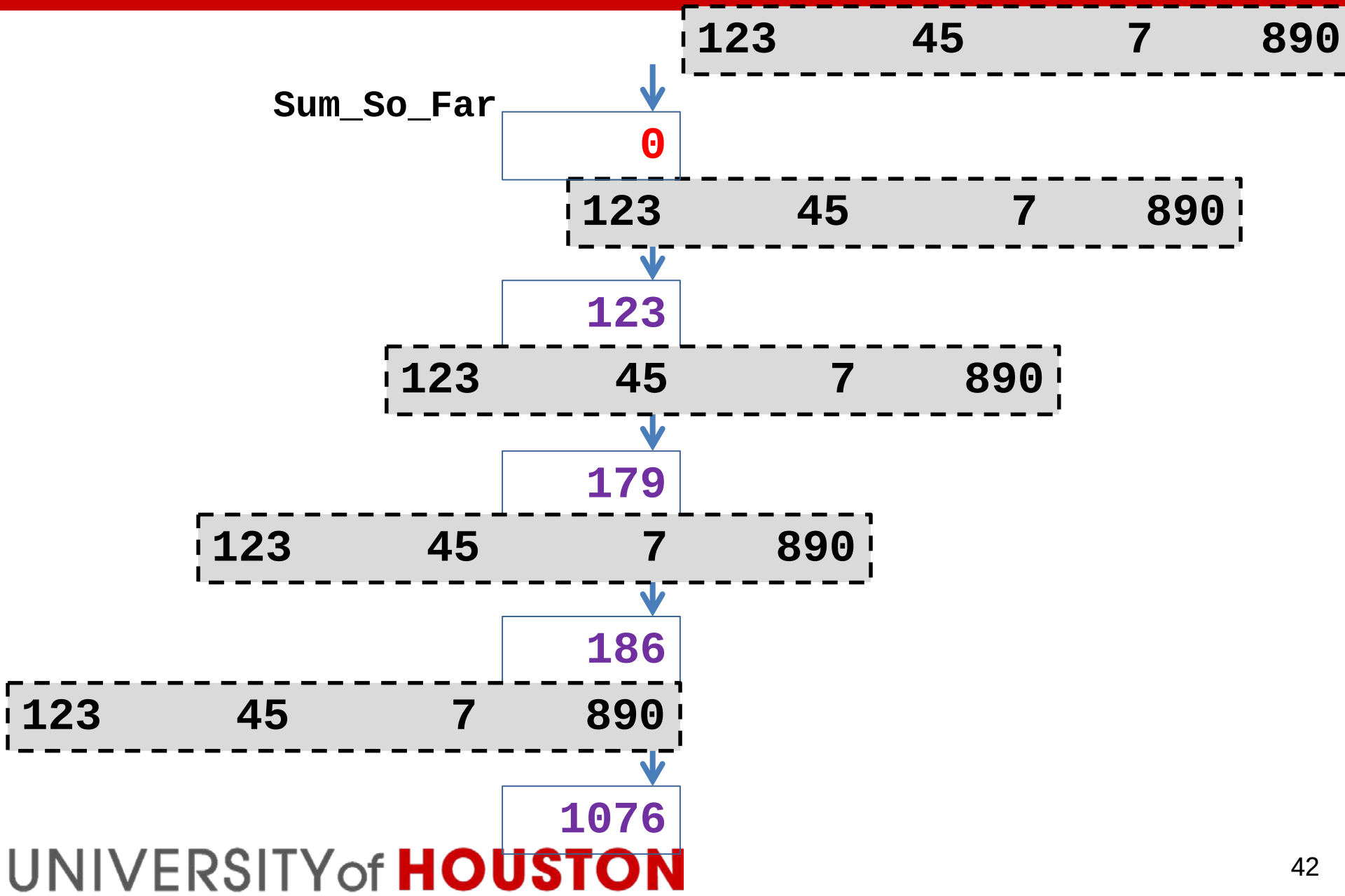
Decomposition



Pattern Recognition



Pattern Recognition



Algorithm Design

Algorithm/Pseudo-code:

```
Initialize Sum_So_Far = 0
For each number in the list
    Add the number to Sum_So_Far
```

Python Code:

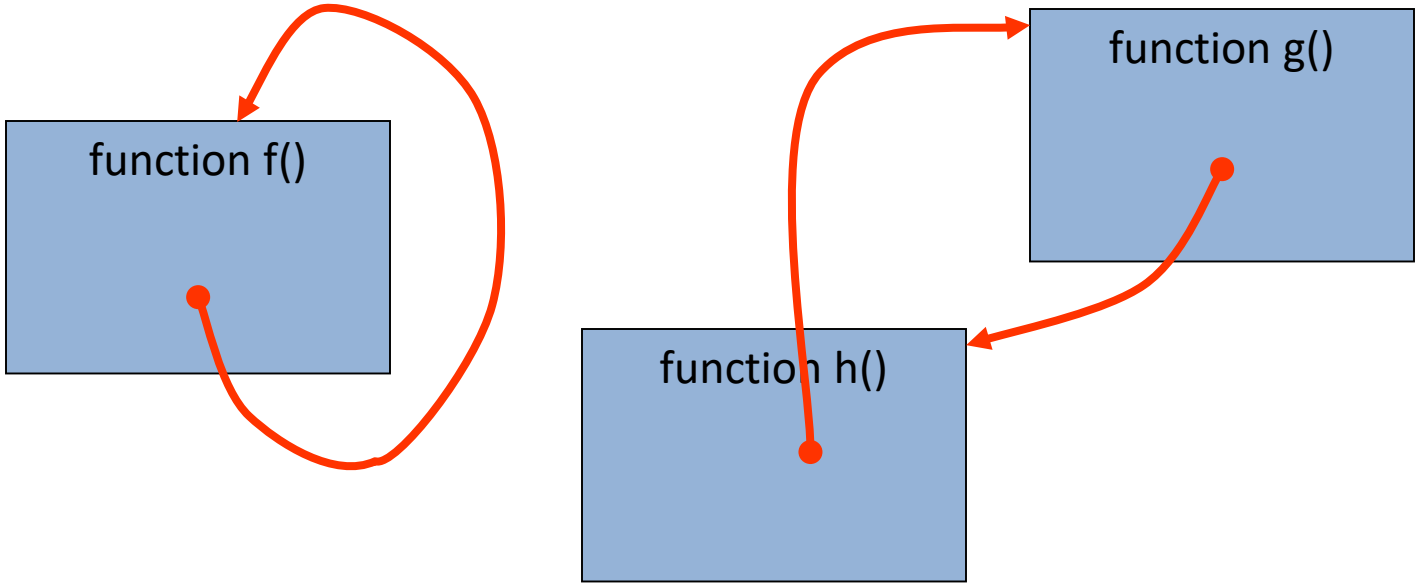
```
sum = 0
for num in myList:
    sum = sum + num
```

What did we learn?

- Decomposition
- Pattern recognition and Abstraction
- Algorithm
  - Use (and reuse) variables to save the result
  - Proper initialization of variables
  - Start with a (not so good) solution and gradually refine it to a solution

3. Recursion

- In Python (and most other languages), a function can call itself within the function. This type of call is named a recursive call.
  - Directly
  - Indirectly



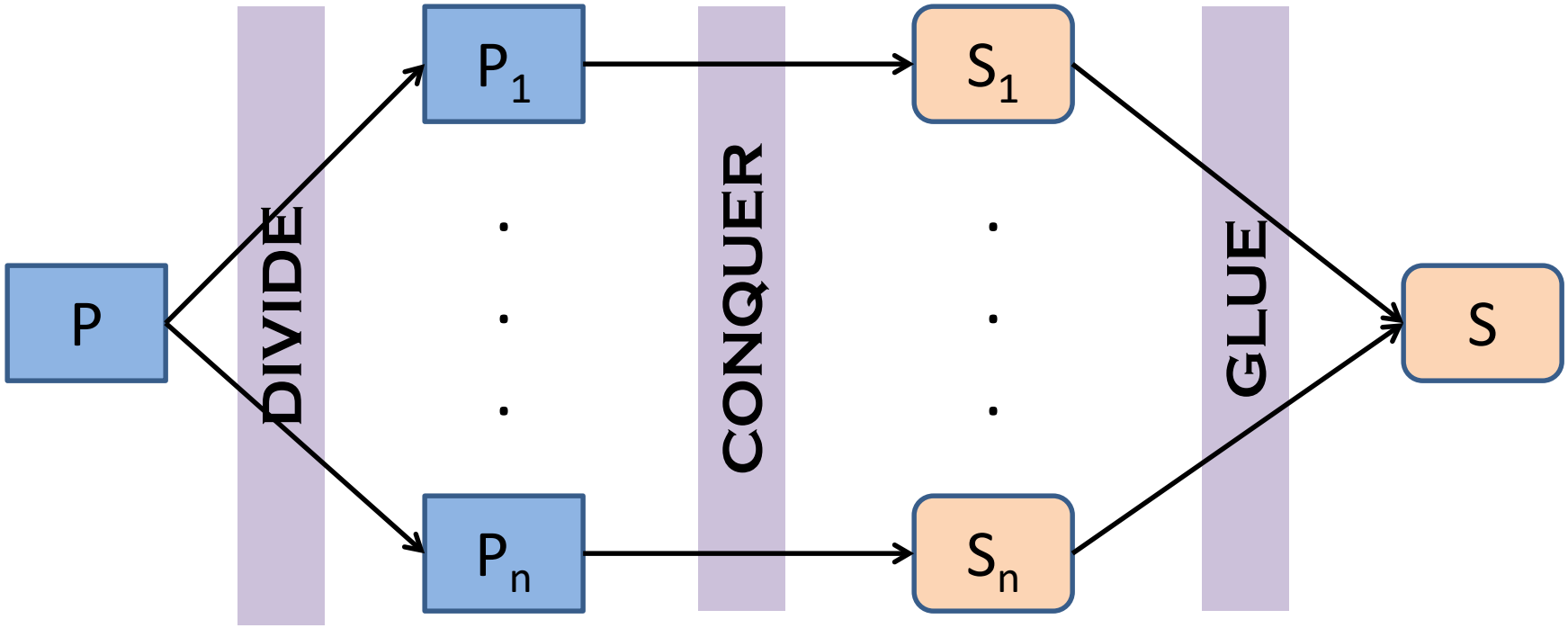
Recursion

- Recursive thinking is one of the profound ideas in CS1.
  - So, it is okay if you don't understand the codes now.
  - We use some syntax that we have not discussed yet in the examples.
- "A journey of a thousand miles begins with a single step."

Decomposition

- Divide, Conquer, and Glue (DCG)
  - Divide a problem P into subproblems P_1, P_2, ..., P_n
  - Conquer the subproblems by solving them, yielding subsolutions S_1, S_2, ..., S_n
  - Glue subsolutions S_1, S_2, ..., S_n together into the solution S to the whole problem P.

DCG



Decomposition

- Sub-problem 1 and sub-problem 2 do not have to be symmetric, i. e., solved the same way, but they usually do.
- One of the solutions must be solved non-recursively.
  - Make the first step of your journey.
  - Take the journey, which is now one step shorter.

Recursive Functions

- Termination of recursive programs: make sure some cases do not make recursive calls.
  - Infinite recursive calls
- So typically, there is an IF statement in the program to distinguish the two cases:
  - Trivial case: no recursions,
  - General case: makes a recursive call. Make sure the problem size is reduced.

Summing Numbers

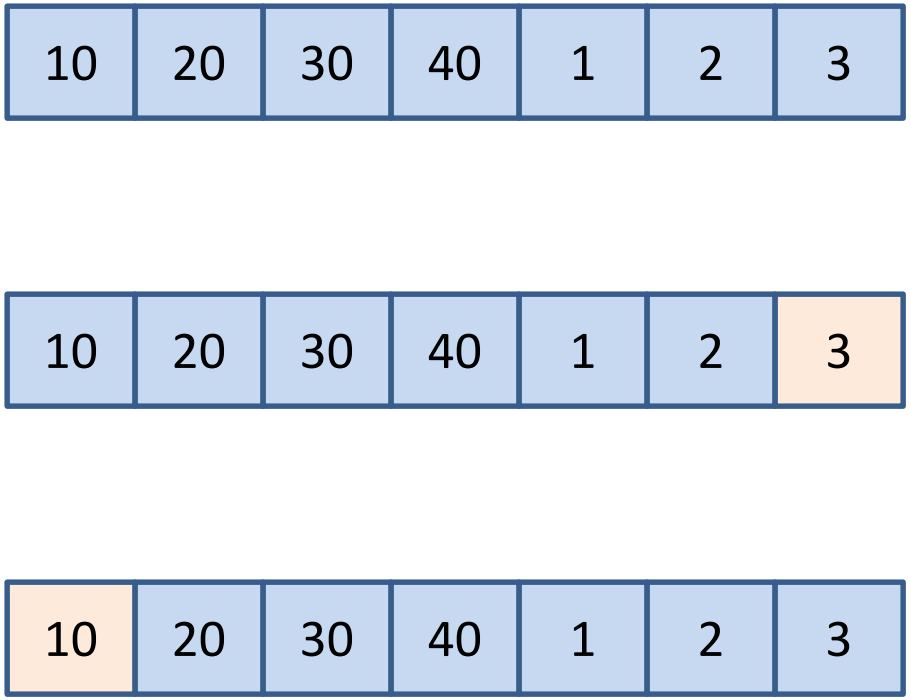
- The following code adds all numbers in the list without using recursion.

```
myList = [10, 20, 30, 40, 1, 2, 3]
sum = 0
for num in myList:
    sum = sum + num
print (sum)
```

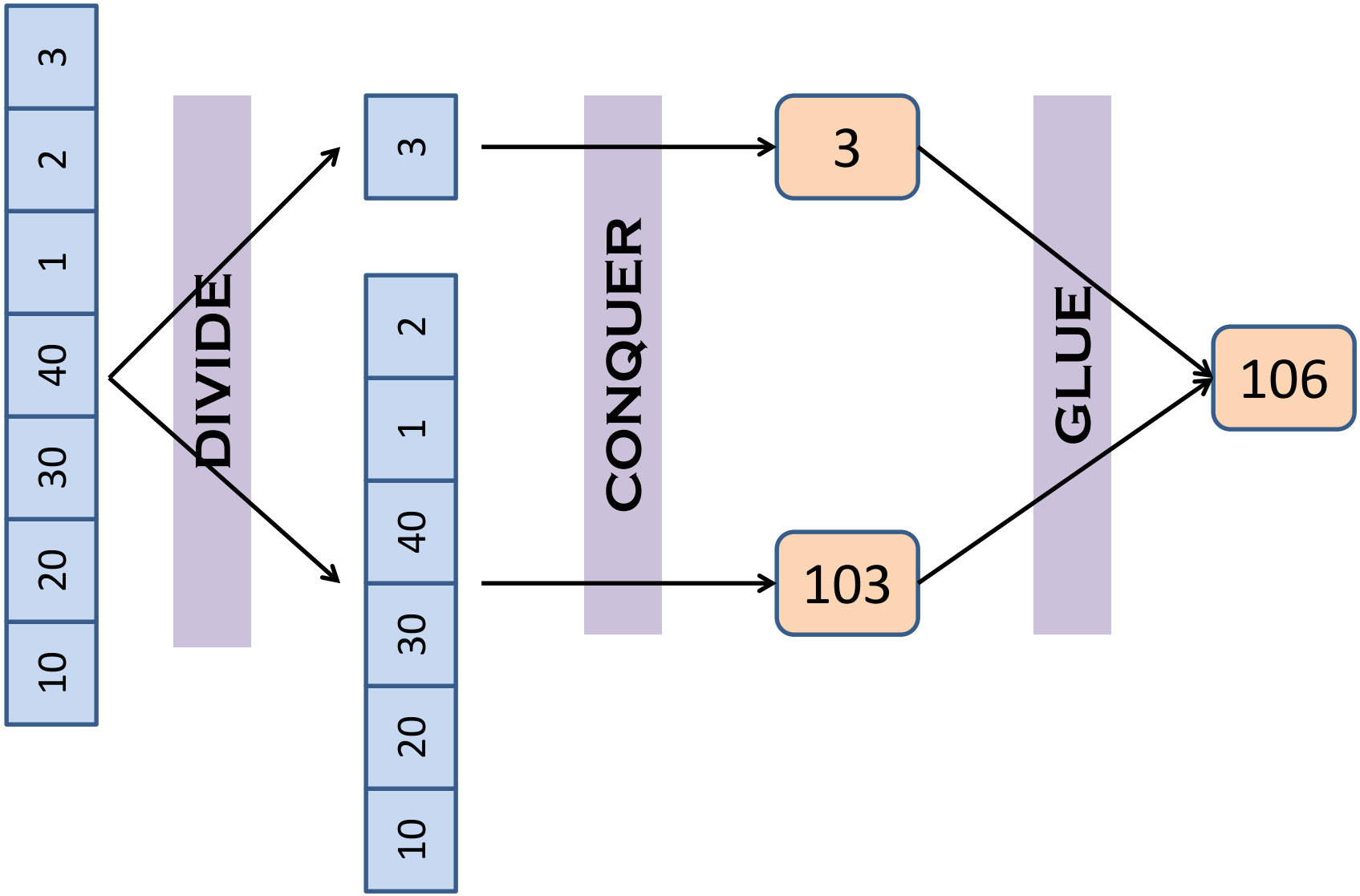
Think Recursively

- Step 1: What is the trivial case(s)?
  - Solve it non recursively.
- Step 2: Identify the exact problem but is smaller is size.
  - Solve it.
- Step 3: Combine the solution(s) (of smaller size) to form the solution for this size.

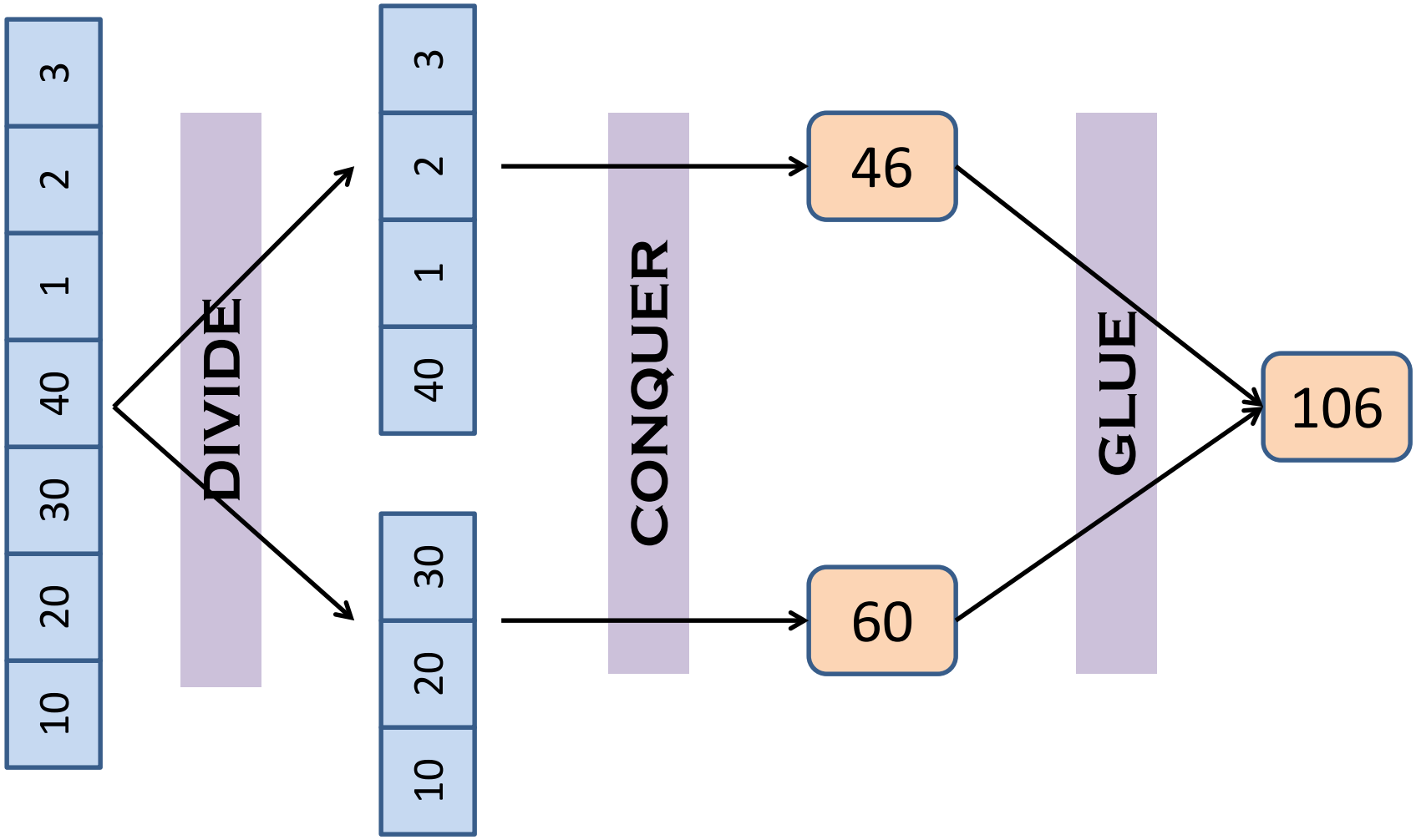
Think Recursively



DCG



DCG



Final Remarks

- Recursion is a great way to solve a problem at the conceptual level.
- There is a significant overhead associated with recursive codes.
- What we presented here is probably as much recursion as you will see in this course. Understand the concept, not the coding.