FIT1045: Algorithms and Programming Fundamentals in Python

Lecture 8

Sorting

Announcements

Reminder: Test 1 this week

- covers only material from Weeks 1 – 3
- questions similar to the exam
  - similar to workshop and tutorial tasks
- opens August 27 2am
- closes August 28, 1pm
- timed (45m)

Objectives

- Become familiar with sorting problem/algorithms
- Practice understanding and describing a problem
- Practice to program planning and decomposition

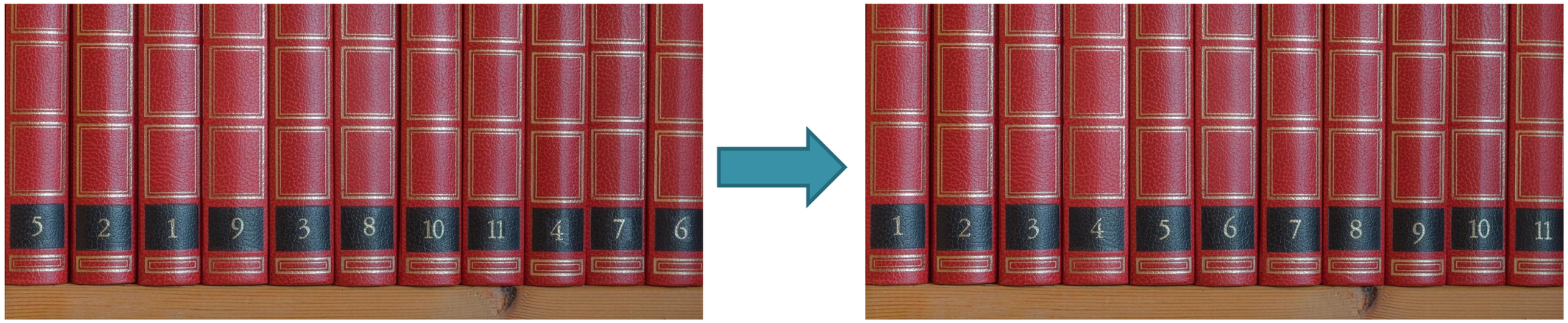
This covers learning outcomes:

- 1 – translate between problem descriptions and program design with appropriate input/output representations
- 2 – choose and implement appropriate problem solving strategies in Python
- 5 – decompose problems into simpler problems and reduce unknown to known problems

Where am I?

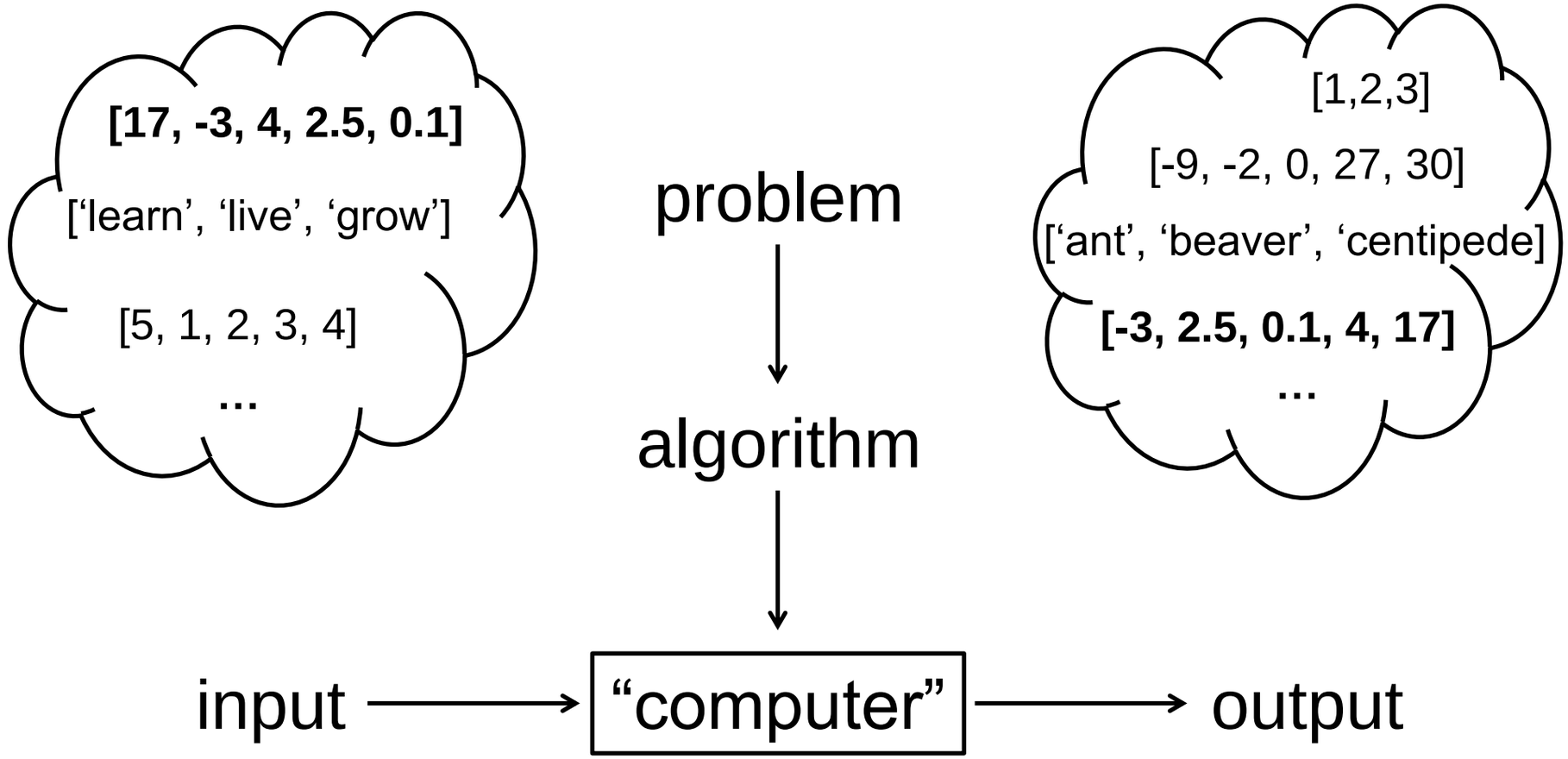
- The Problem of Sorting
- Selection Sort
- Insertion Sort

What is sorting and why studying it?



- Useful for producing human-readable output
- Important building block for many algorithms
  - Certain fast algorithms for optimization (cost minimisation)
  - Storing data in structure that allows efficient retrieval of information
- Illustrates many important algorithm design paradigms
- Practice to reason about algorithms and their properties
- Computers spend a lot of time (and energy) sorting; choosing "better" algorithm can save substantial amount of resources

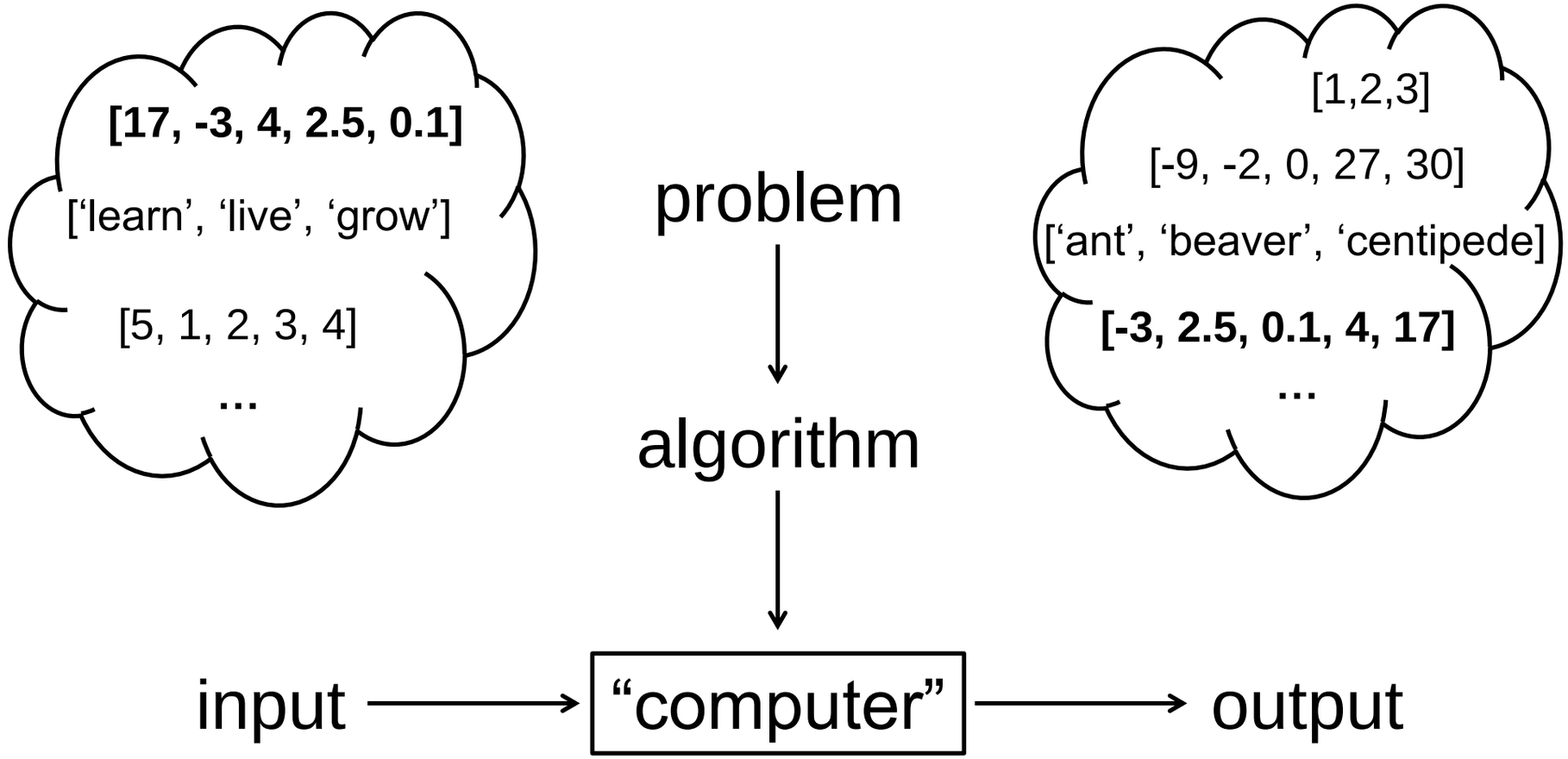
Sorting problem



Input: list

Output: sorted list ← Any list?

Sorting problem



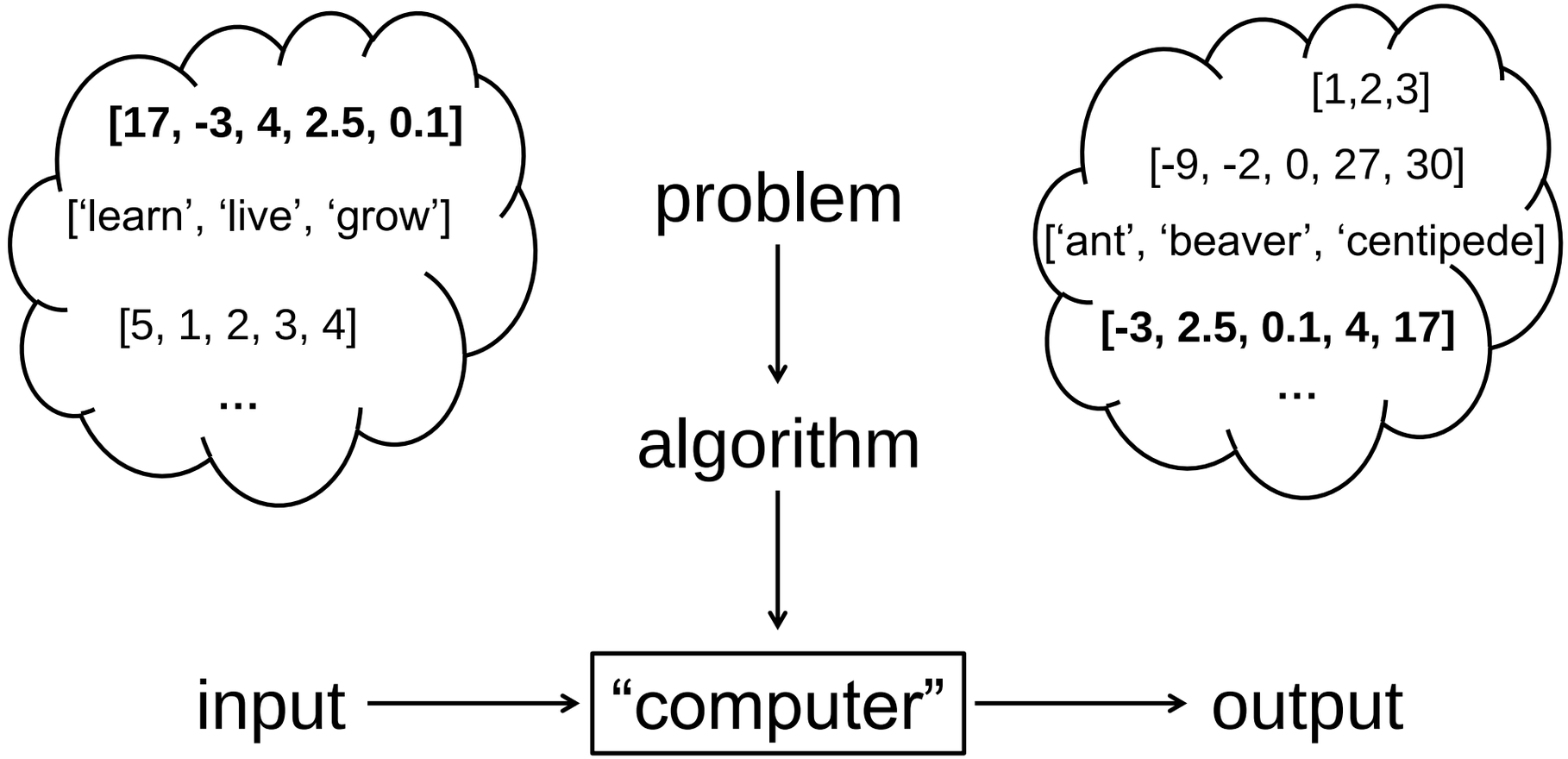
What does that mean?

Input: list `inlst`

Output: sorted list `outlst` containing exactly same elements as `inlst`

technically: a *permutation*

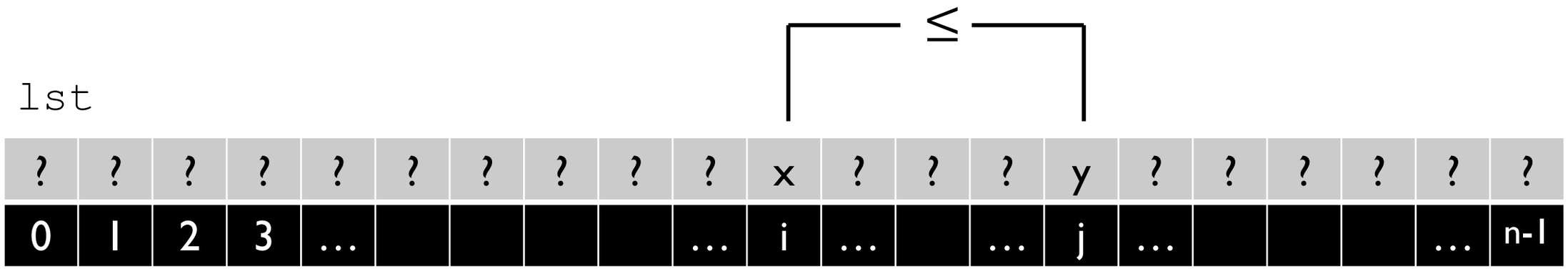
Sorting problem



Input: list `inlst` of comparable elements (`x<y` defined for `x, y` in `inlst`)

Output: sorted list `outlst` containing exactly same elements as `inlst`

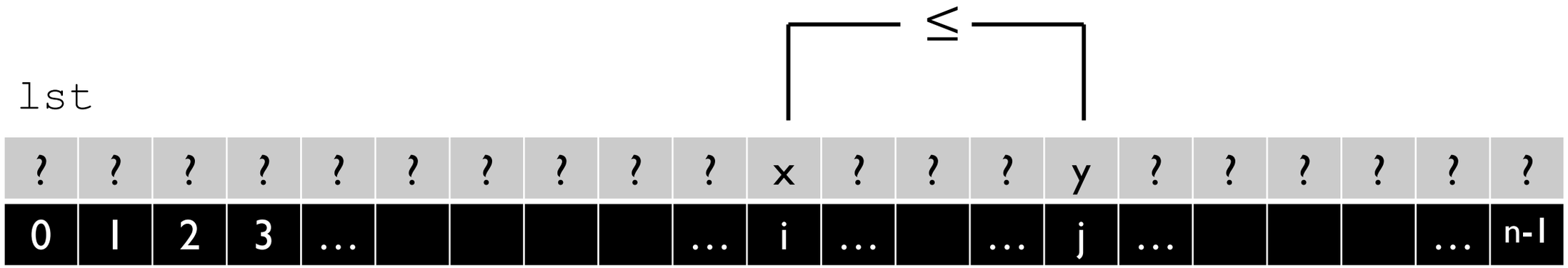
Sorting problem



Input: list `inlst` of comparable elements (`x<y` defined for `x, y` in `inlst`)

Output: sorted list `outlst` containing exactly same elements as `inlst`

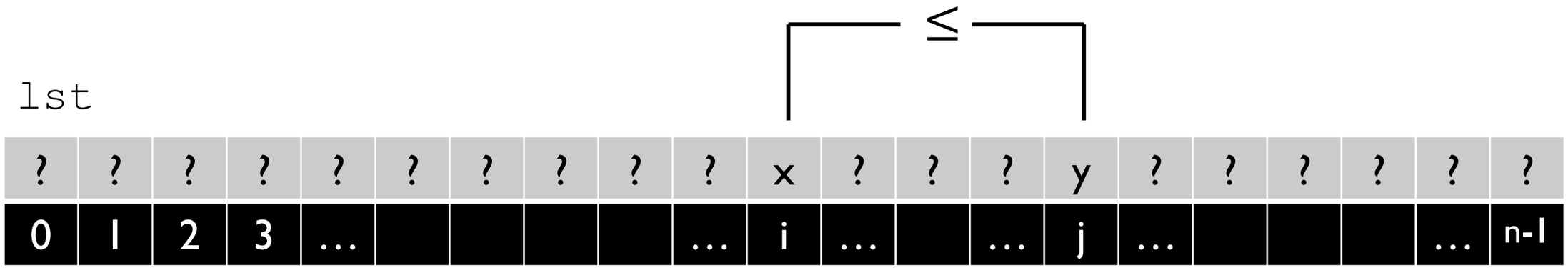
Sorting problem



Input: list `inlst` of comparable elements (`x<y` defined for `x, y` in `inlst`)

Output: list `outlst` containing exactly same elements as `inlst` that is sorted (for all `i` in `range(1, n):outlst[i-1]<=outlst[i]`)

Sorting problem



https://flux.qa
Clayton: AXXULH
Malaysia: LWERDE

Input: list `inlst` of comparable elements (`x<y` defined for `x, y` in `inlst`)

Output: list `outlst` containing exactly same elements as `inlst` that is sorted (for all `i in range(1, n):outlst[i-1]<=outlst[i]`)

condition that relates list positions and values

Lots of algorithms solving the sorting problem in literature…

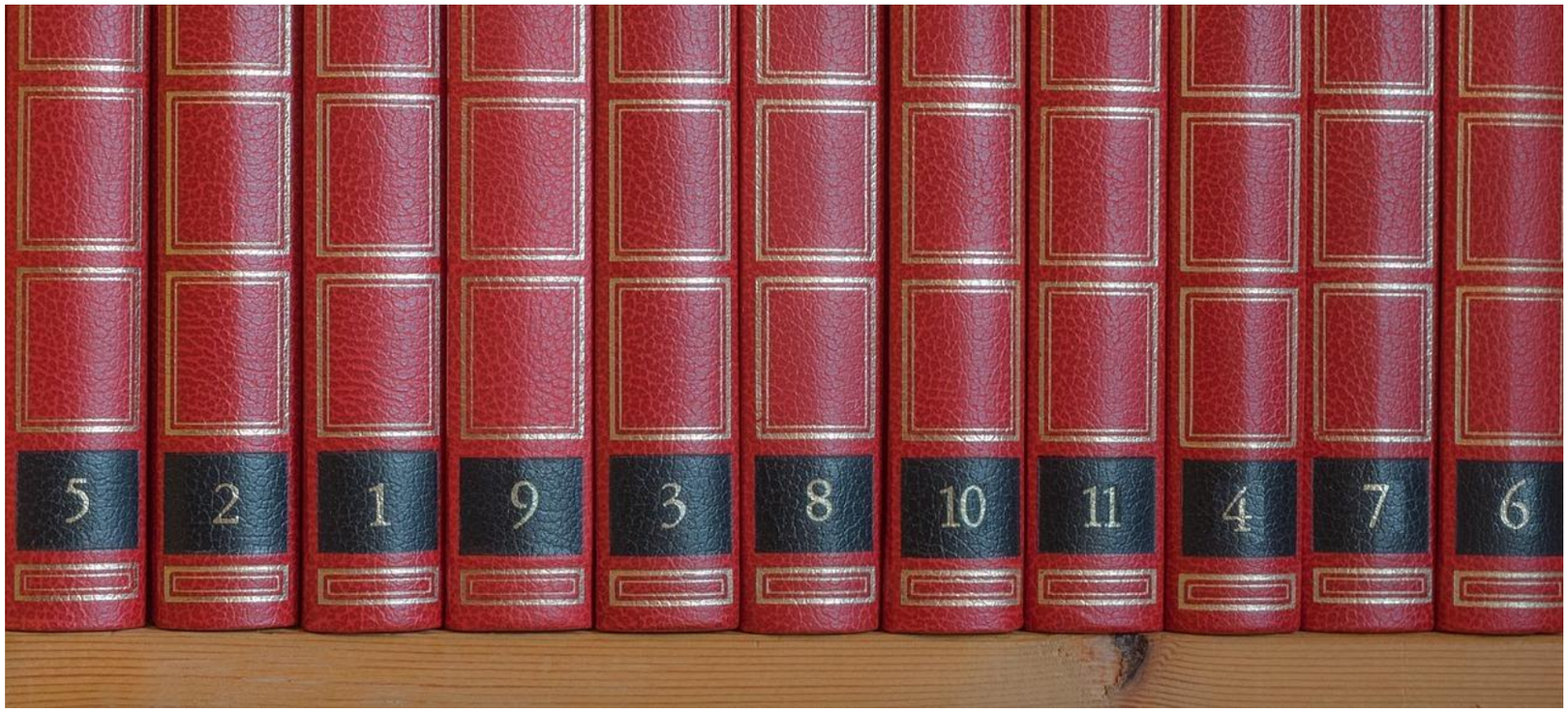
- **Selection Sort**
- **Insertion Sort**
- Merge Sort
- Quick Sort
- Heap Sort
- Counting Sort
- …

Great problem to study algorithms, design paradigms, and their properties!

Where am I?

- The Problem of Sorting
- Selection Sort
- Insertion Sort

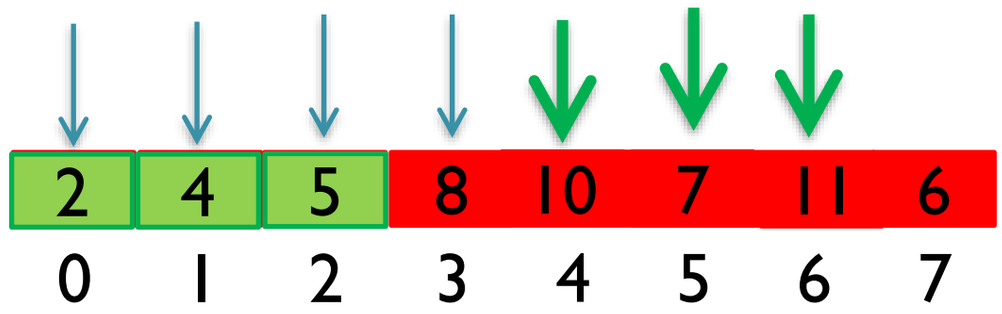
How would *you* solve it?



Idea

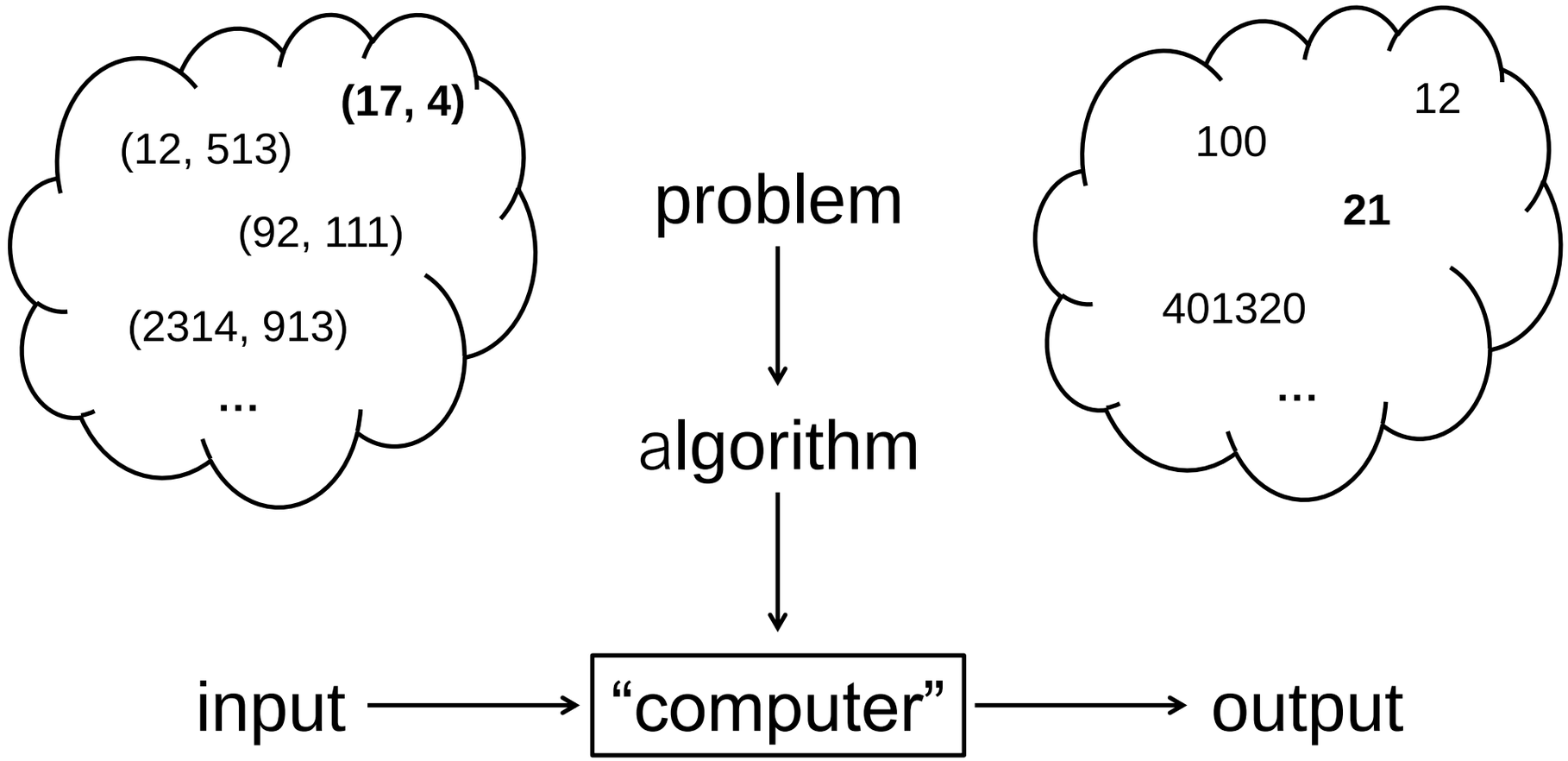
- **Select** smallest item and move it to 1st position.
- **Select** 2nd smallest item and move it to 2nd position.
- **Select** 3rd smallest item and move it to 3rd position.
- …
- **Select** n-th smallest item and move it to n-th position.

Selection Sort algorithm

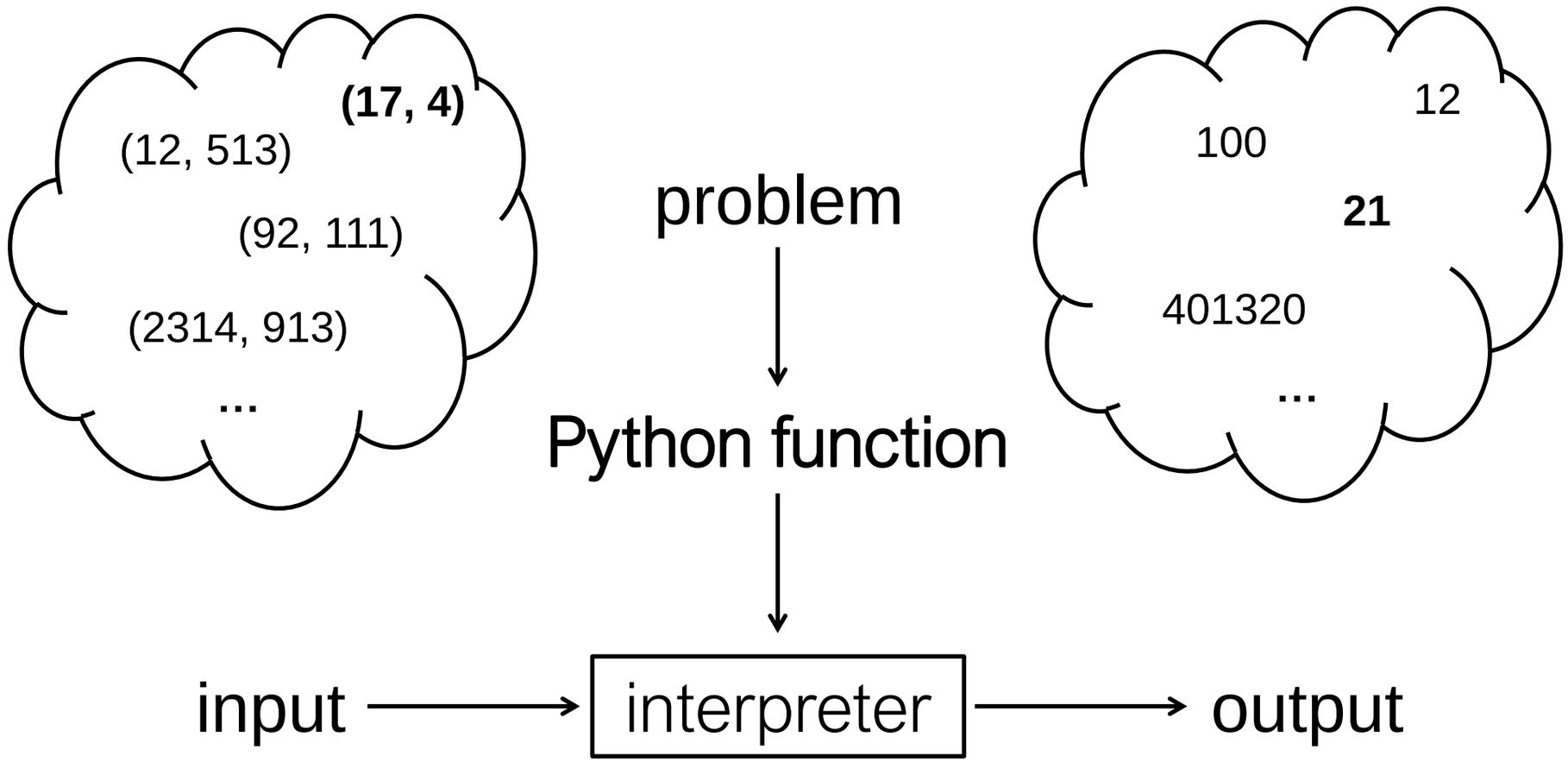


- Select smallest item in `lst` and swap it with `lst[0]`
- Select smallest item in `lst[1:n]` and swap it with `lst[1]`
- Select smallest item in `lst[2:n]` and swap it with `lst[2]`
- …
- Select smallest item in `lst[i:n]` and swap it with `lst[i]`

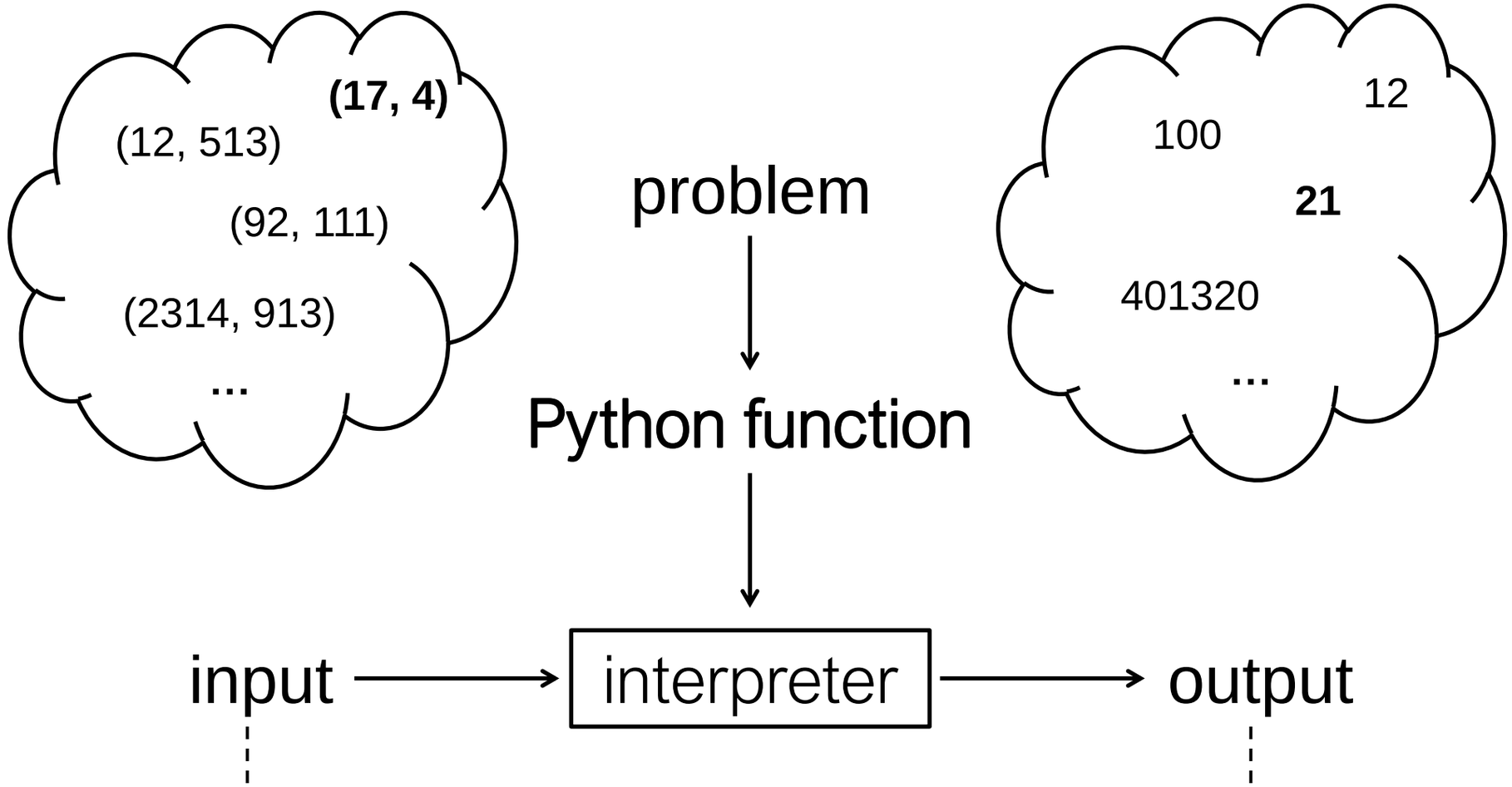
Python functions *can* represent algorithms



Python functions *can* represent algorithms

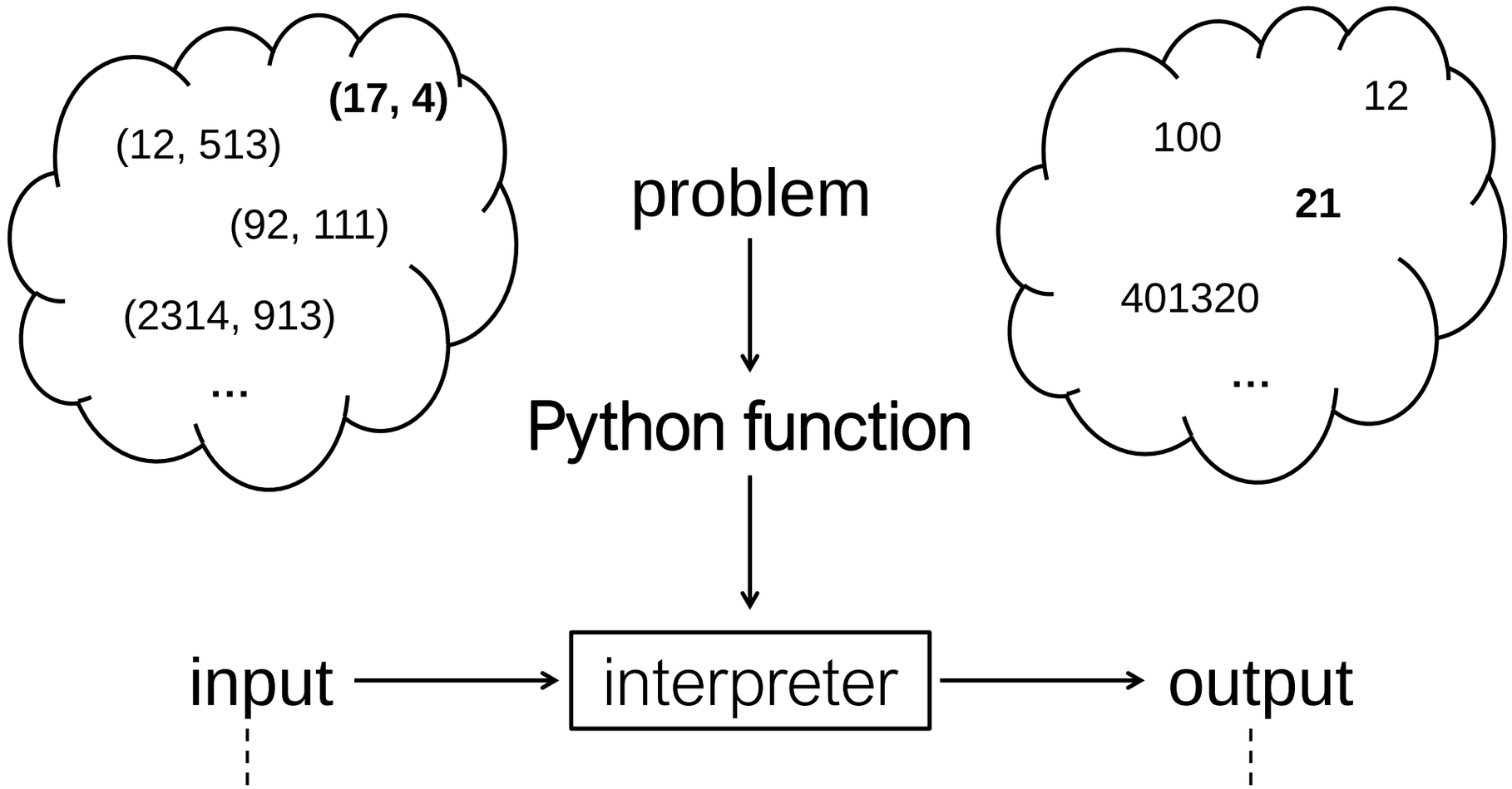


Python functions *can* represent algorithms… in many ways



arguments global variables read from input source	return value mutated arguments mutated/re-assigned global vars written to file or console

For Selection Sort: output represented by mutated input object



arguments global variables read from input source	return value mutated arguments mutated/re-assigned global vars written to file or console

Selection Sort in Python

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
```

Selection Sort in Python

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    #for all indices i of lst:
    #   find index j of min element in sublist from i
    #   swap elements at i and j
```

Selection Sort in Python

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
    #   find index j of min element in sublist from i
    #   swap elements at i and j
```

Selection Sort in Python

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
    #   find index j of min element in sublist from i
        lst[i], lst[j] = lst[j], lst[i]
```

Selection Sort in Python

```
def min_index(lst):
```

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
        j = min_index(lst[i:])
        lst[i], lst[j] = lst[j], lst[i]
```

Selection Sort in Python

```
def min_index(lst):
    """
    accepts: list of length n>0 of comparable elements


    """
```

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
        j = min_index(lst[i:])
        lst[i], lst[j] = lst[j], lst[i]
```

Selection Sort in Python

```
def min_index(lst):
    """
    accepts: list of length n>0 of comparable elements
    returns: index k in range(n) such that
             for all j in range(n), lst[k]<=lst[j]
    """
```

```
def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
        j = min_index(lst[i:])
        lst[i], lst[j] = lst[j], lst[i]
```

Selection Sort in Python

```
def min_index(lst):
    """
    accepts: list of length n>0 of comparable elements
    returns: index k in range(n) such that
             for all j in range(n), lst[k]<=lst[j]
    """
    #
    # exercise
    #

def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
        j = min_index(lst[i:])
        lst[i], lst[j] = lst[j], lst[i]
```

careful: index refers to slice

Selection Sort in Python

```
def min_index(lst):
    """
    accepts: list of length n>0 of comparable elements
    returns: index k in range(n) such that
             for all j in range(n), lst[k]<=lst[j]
    """
    #
    # exercise
    #

def selection_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(len(lst)):
        j = min_index(lst[i:]) + i
        lst[i], lst[j] = lst[j], lst[i]
```

https://flux.qa
Clayton: AXXULH
Malaysia: LWERDE

Where am I?

- The Problem of Sorting
- Selection Sort
- Insertion Sort

Alternative approach: how do you sort a hand of cards?

Assume all cards are lying face down on the table

- Pick 1st card
- Pick 2nd card and **insert** it such that the 2 picked cards are sorted
- Pick 3rd card and **insert** it such that the 3 picked cards are sorted
- Pick 4th card and **insert** it such that the 4 picked cards are sorted
- …
- Pick n-th card and **insert** it such that the n picked cards are sorted

Insertion Sort algorithm

6 5 3 1 8 7 2 4

Source: wikimedia.org

We only do this all the time!

- declare `lst[0]` to be sorted
- compare `lst[1]` and `lst[0]` and swap if required
- insert item `lst[2]` to right place in `lst[0:3]`
- …
- insert item `lst[i]` into right place in `lst[0:i+1]`
- …
- insert item `lst[n-1]` to right place in `lst[0:n]` (==`lst`)

Insertion Sort in Python

```
def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted

    """
```

```
def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted
    postcon: lst[:k+1] is sorted
    """
```

```
def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted
    postcon: lst[:k+1] is sorted
    """
    # let j be the index of insertion element
    # while j is not correct position for insertion el.
        # swap insertion element with element to left
        # update j


def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted
    postcon: lst[:k+1] is sorted
    """
    j = k
    # while j is not correct position for insertion el.
        # swap insertion element with element to left
        j = j - 1


def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted
    postcon: lst[:k+1] is sorted
    """
    j = k
    # while j is not correct position for insertion el.
        lst[j - 1], lst[j] = lst[j], lst[j - 1]
        j = j - 1


def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted
    postcon: lst[:k+1] is sorted
    """
    j = k
    while lst[j - 1] > lst[j]:
        lst[j - 1], lst[j] = lst[j], lst[j - 1]
        j = j - 1


def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

what about case that Insertion element is minimum?

Insertion Sort in Python

```
def insert(k, lst):
    """
    accepts: list lst of length n>k>=0 of comp. elements
             such that lst[:k] is sorted
    postcon: lst[:k+1] is sorted
    """
    j = k
    while j > 0 and lst[j - 1] > lst[j]:
        lst[j - 1], lst[j] = lst[j], lst[j - 1]
        j = j - 1


def insertion_sort(lst):
    """
    accepts  : list lst of length n of comp. elements
    post-cond: lst has same elements as on call but
               for all i in range(1,n), lst[i-1]<=lst[i]
    """
    for i in range(1, len(lst)):
        insert(i, lst)
```

https://flux.qa
Clayton: AXXULH
Malaysia: LWERDE

What have we learnt?

- Selection Sort and Insertion Sort
- How to carefully specify a problem
  - input assumptions (what does it mean to be sortable)
  - output conditions (what does it mean to be sorted)
- Reduce complex problem to simpler problem (*decomposition*)
  - sorting to *selection* of minimum
  - sorting to *insertion* into already sorted list

Checkpoint for this week

After today you should be able to the following:

1. Implement Selection Sort
2. Implement Insertion Sort
3. Write Python function to retrieve the smallest 5 items in a list

Until next week

Re-implement sorting algorithms

Think about the following question…

Cutting the Chocolate Block

A chocolate block is divided into squares by horizontal and vertical grooves. The objective is to cut the chocolate block into individual squares.

Assume each cut is made on a single piece along a groove. How many cuts are needed?

Coming Up

- Relational data (Graphs)
- Reasoning about algorithms (Invariants)