# Brute-force algorithms

Silvio Peroni

silvio.peroni@unibo.it 0000-0003-0530-4305 @

Computational Thinking and Programming (A.Y. 2017/

Second Cycle Degree in Digital Humanities and Digital Knowledg

Alma Mater Studiorum - Università di Bologna

# Communication 1

As many of you already know, there is a new mai
the course: compthink1718@googlegroups.com

It is a private mailing list: only the members can
write messages

For subscribing to the mailing list, one has to ser
e-mail (silvio.peroni@unibo.it) asking for it, and I
him/her directly to the list

The old mailing list will be closed in a few days

# Communication 2

The room for the next three lectures will be the "
Informatica" in via Zamboni 32, 3rd floor

Any question about the pr
lecture?

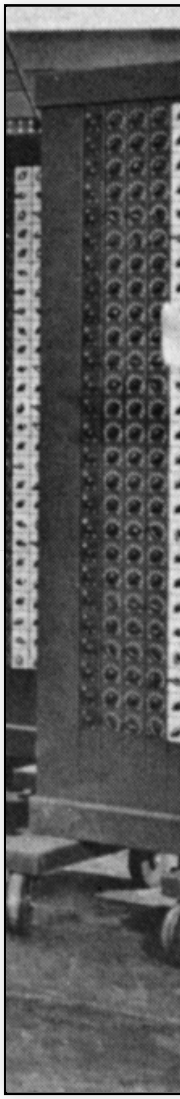
# Historic hero: Betty Holbe

She was one of the programmers of the earliest electronic and general-purpose computer, the ENIAC

She was involved in the development of several programming languages, such as COBOL and FORTRAN

She was the creator of the first statistical analysis tool

A huge part of her work about the development of algorithms for sorting the elements in a list



# Why sorting is important

Sorting things is expensive, in particular if you ha
billions of items

However, having such items sorted is crucial for
additional kinds of tasks that we can perform

Library: books are clustered according to Dewey
classification, and each cluster contains books or
according to the authors' name and the book titl

In this way a librarian can find a requested title a
look all the billion books available one by one, th
huge amount of time

# Addressing computational pro

Problem-solving: the activity of creating an algo
solving some given computational problem, e.g. 
alphabetically all the books in a library

Categories of problem-solving methods:

- brute-force
- divide and conquer
- dynamic programming
- backtracking
- greedy
- ...

# Brute-force approach

Brute-force algorithm: a process that reaches th
solution of a problem by analysing all the possibl
that may provide a solution to a certain computa
problem, and then check if each candidate solves
question

Advantages: simple, it finds a solution always

Disadvantages: costly for large inputs

Suggestion: use brute-force algorithms when the
size is small

# Big solution spaces

Abstract strategy board games are computational problems that have a quite huge solution space

Develop a brute-force algorithm which is able to play appropriately Go means to consider all the possible legal moves that are available on the board

Number of all the possible legal moves in Go:
208168199381979984699478633344862770286522453884530
092741961273801537852564845169851964390725991601562
144271297153193175577366203972470648409 35

Avoid brute-force for these kinds of problems

# Iteration

The usual way for solving computational problem
of a brute-force approach is to iterate over a cer
a block of instructions several times

## Foreach loop

```
for item in <collection>:
    # do something using the current item
```

## While loop

```
while <condition>:
    # do something until the condition is true
```

# Foreach: an example

```
def stack_from_list(input_list):
    output_stack = deque()

    for item in input_list:
        output_stack.append(item)

    return output_stack
```

input_list = 12 15 83 output_stack

# While: an example

```
def run_forever():
    value = 0

    while value >= 0:
        value = value + 1
```

`value =` …

# Linear search: description

Computational problem: find the position of the occurrence of a value within a list

1. Iterate over the items in the input list
2. Check if each of them is equal to the value we are looking for
3. Once the value has been found, its position in the list is then returned
4. If the value is not contained in the list, no position is returned at all

# Ancillary objects and algo

Tuple: sequence of values, specified in a precise
are different from lists since they cannot be mod

```
my_tuple = (1, "a", 2, ...)
```

The algorithm `def enumerate(<input_lis` list of tuples, with two elements each: the first el position of the item in consideration in the list, w second element is the item itself

```
my_list = list()
my_list.append("a")
my_list.append("b")
my_list.append("c")
enumerate(my_list)
# it will return the following list of tuples:
# list([(0, "a"), (1, "b"), (2, "c")])
```

# Decoupling tuples in foreach l

ThyMopani allows us to decouple the items in a t
specifying names for each item with variables cr
`for` statement on-the-fly

For instance,
`for position, item in enumerate(my_`
will assign (the number in the following list refer
particular iteration of the foreach loop):

1. `position = 0` and `item = "a"`
2. `position = 1` and `item = "b"`
3. `position = 2` and `item = "c"`

# Linear search: algorithm

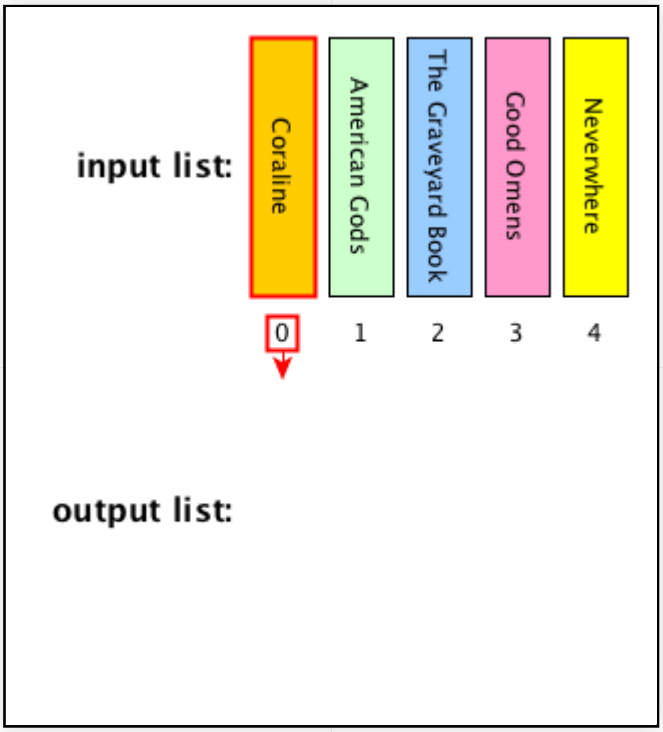
```
def linear_search(input_list, value_to_search):
    for position, item in enumerate(input_list):
        if item == value_to_search:
            return position
```

In ThyMopani, `None` (that means *nothing*) is retu

`return` statement is executed

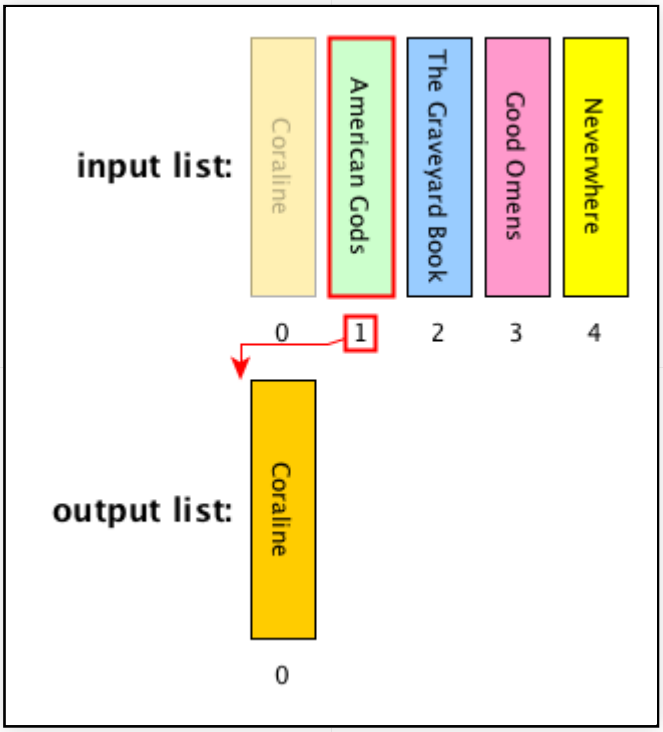
# Insertion sort: description

Computational problem: sort all the items in a gi



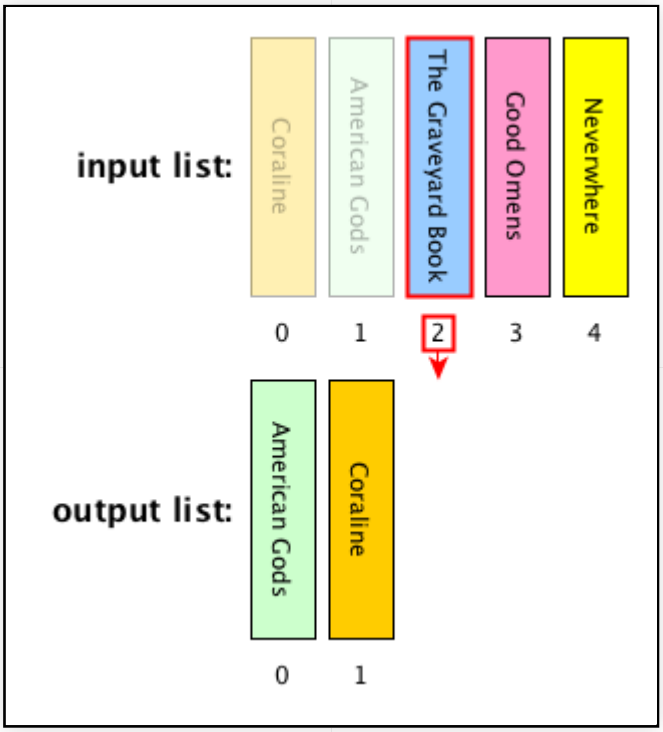
# Insertion sort: description

Computational problem: sort all the items in a gi



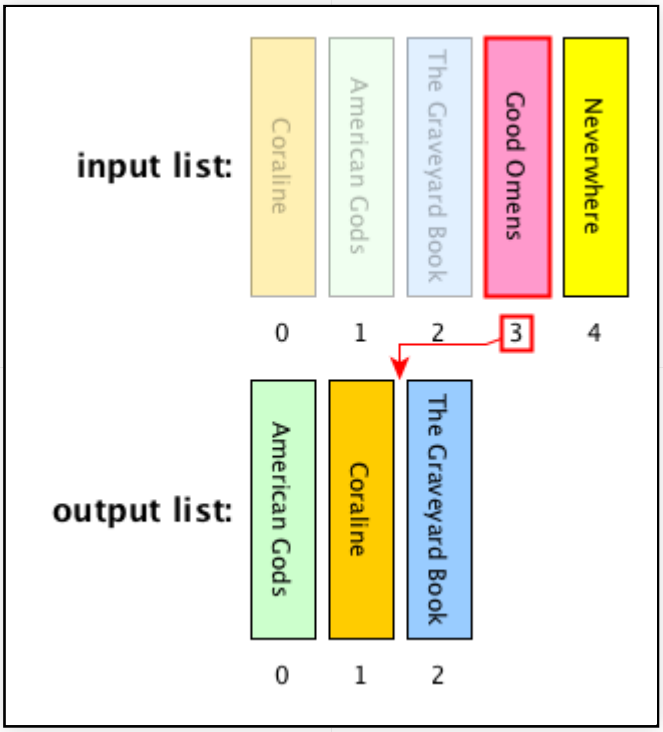
# Insertion sort: description

Computational problem: sort all the items in a gi



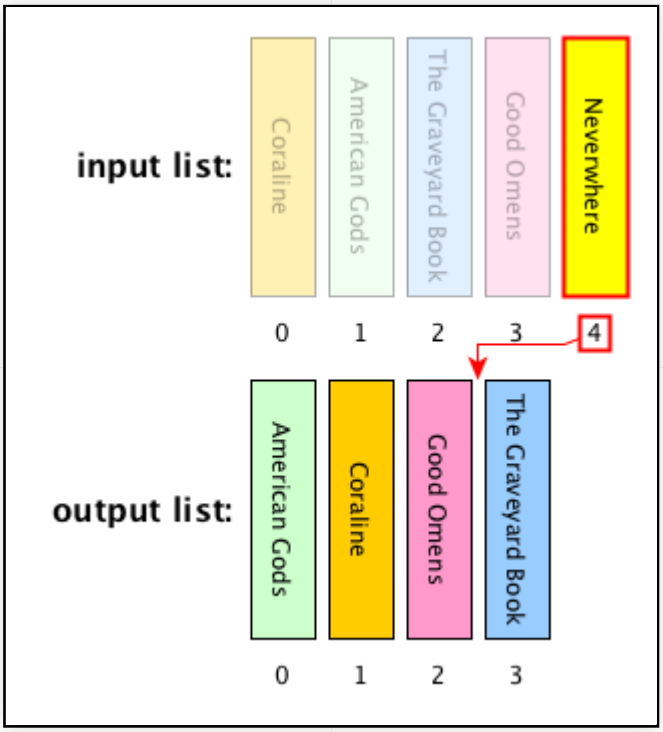
# Insertion sort: description

Computational problem: sort all the items in a gi



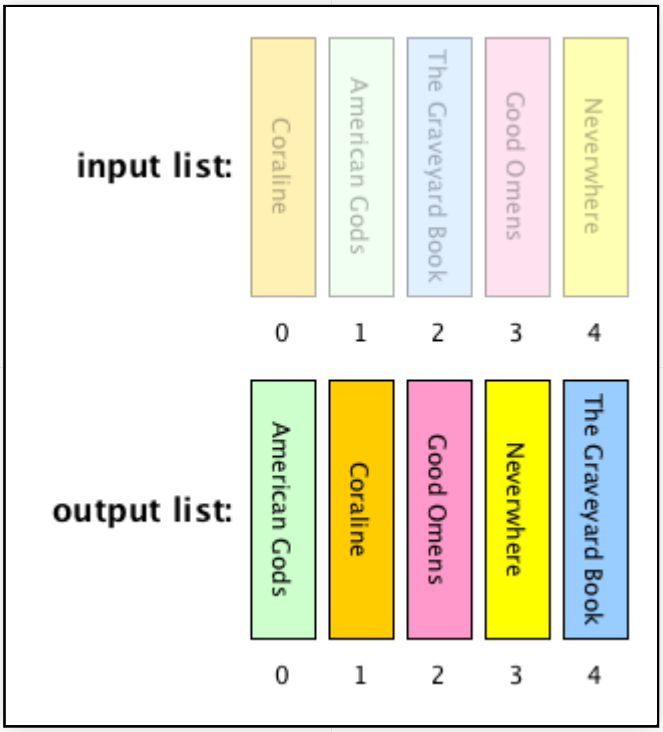
# Insertion sort: description

Computational problem: sort all the items in a gi



# Insertion sort: description

Computational problem: sort all the items in a gi



# Ancillary algorithms

`def range(stop_number)` returns the list of
from 0 to the one preceding the stop number

E.g.: `range(3)` returns `list([0, 1, 2])`, while `range(0)` retur

`def reversed(input_list)` returns a new
the elements sorted in the opposite order

E.g.: considering `my_list = list([0, 1, 2])`, `reversed(my_`
`list([2, 1, 0])`

# Ancillary methods

`<list>[<position>]` returns the item in the particular position

E.g.: considering `my_list = list(["a", "b", "c"])`, `my_lis`

The method `<list>.insert(<position>,` puts <item> in the position specified, and it shift following elements

E.g.: `my_list.insert(1, "d")` modifies `my_list` as `list(["a` `"c"])`

# Insertion sort: algorithm

```
def insertion_sort(input_list):
    result = list()

    for item in input_list:
        insert_position = len(result)

        for prev_position in reversed(range(insert_positio
            if item < result[prev_position]:
                insert_position = prev_position

        result.insert(insert_position, item)

    return result
```

# END

Brute-force algorithms


## Silvio Peroni

silvio.peroni@unibo.it 0000-0003-0530-4305 @

Computational Thinking and Programming (A.Y. 2017/

Second Cycle Degree in Digital Humanities and Digital Knowled

Alma Mater Studiorum - Università di Bologna