Divide and conquer algorithms

Silvio Peroni

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

Computational Thinking and Programming (A.Y. 2017/2018)

Second Cycle Degree in Digital Humanities and Digital Knowledge

Alma Mater Studiorum - Università di Bologna

Communication 1

Wednesday the 6th of December there will be the introduction to the project specifications and rules

Please do not miss this lecture!

Communication 2

The results of the first partial written examination has been sent to all the people who attended it (as a personal communication)

Have anyone missed it?

Any question about the previous lecture?

Historic hero: John von Neumann

He was a computer scientist, mathematicians, and physicists

Several contribution in quantum mechanics, game theory, and self-replicating machines

Von Neumann architecture: guidelines for building physical electronic computers, included in the document written by John von Neumann for defining the main design principles of the EDVAC, the binary-based successor of the ENIAC

Divide and conquer approach

Divide and conquer algorithm is based on four steps

1. **[base case]** address directly if it is an easy-to-solve problem, otherwise
2. **[divide]** split the input material into two or more balanced parts, each depicting a sub-problem of the original one
3. **[conquer]** run the same algorithm recursively for every balanced parts obtained in the previous step
4. **[combine]** reconstruct the final solution of the problem by means of the partial solutions

Advantages: usually quicker than brute force

Disadvantages: recursion must be defined carefully

Merge sort

Computational problem: sort all the items in a given list

Merge sort was proposed by John von Neumann in 1945

It implements a divide a conquer approach for sorting elements in a list

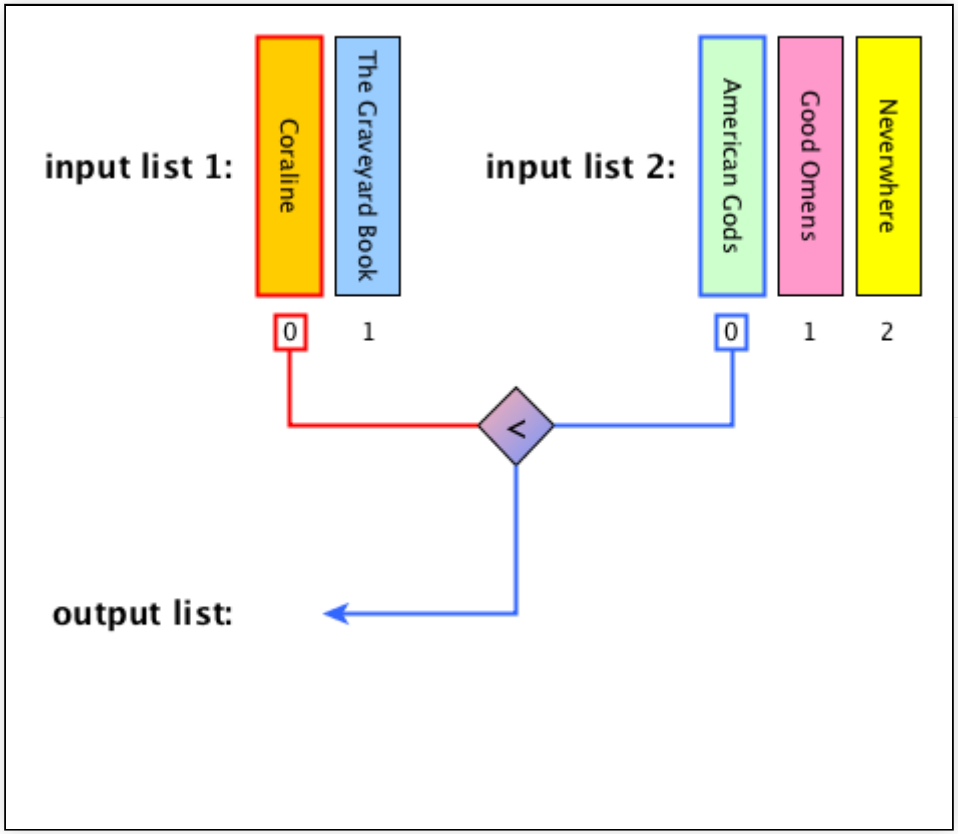
It is more efficient than the *insertion sort*

It needs an ancillary algorithm:

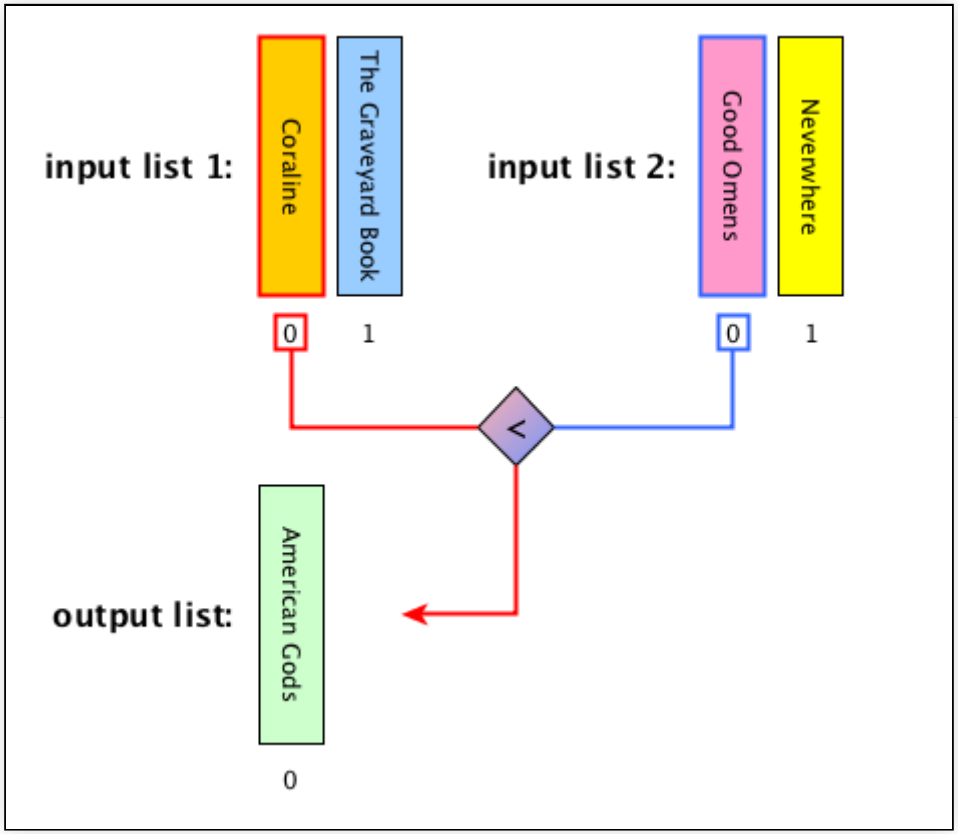
```
def merge(list_1, list_2)
```

It combines two ordered input lists together so as to return a new list which contains all the elements in the input lists ordered

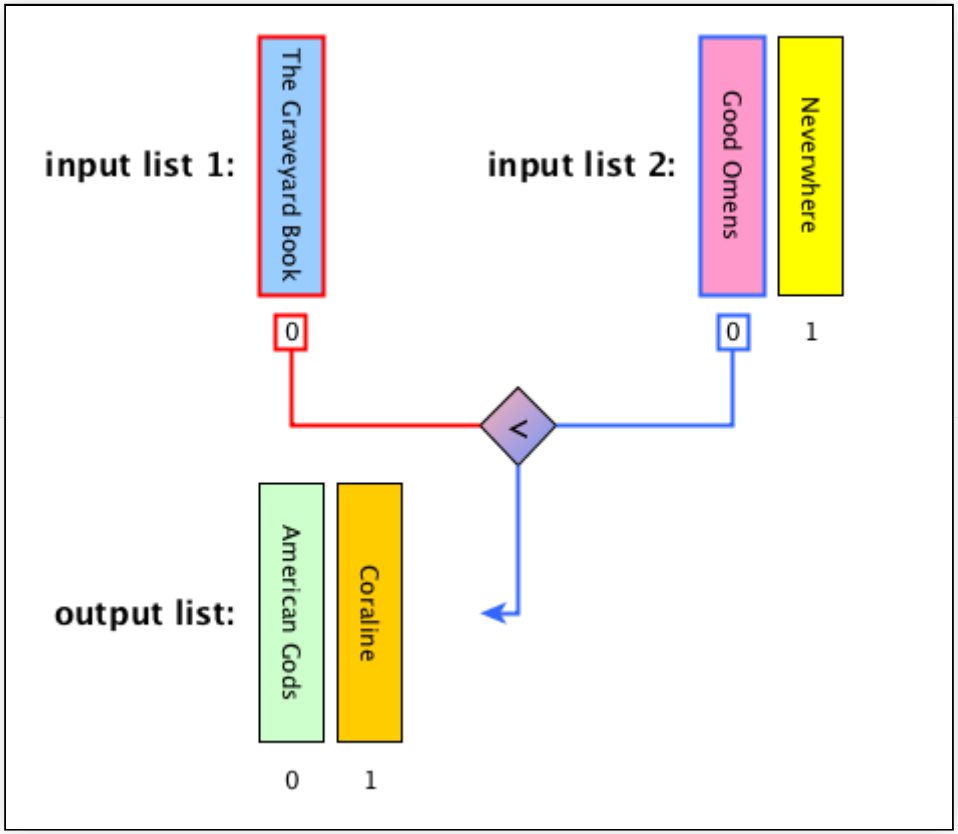
Merge: description



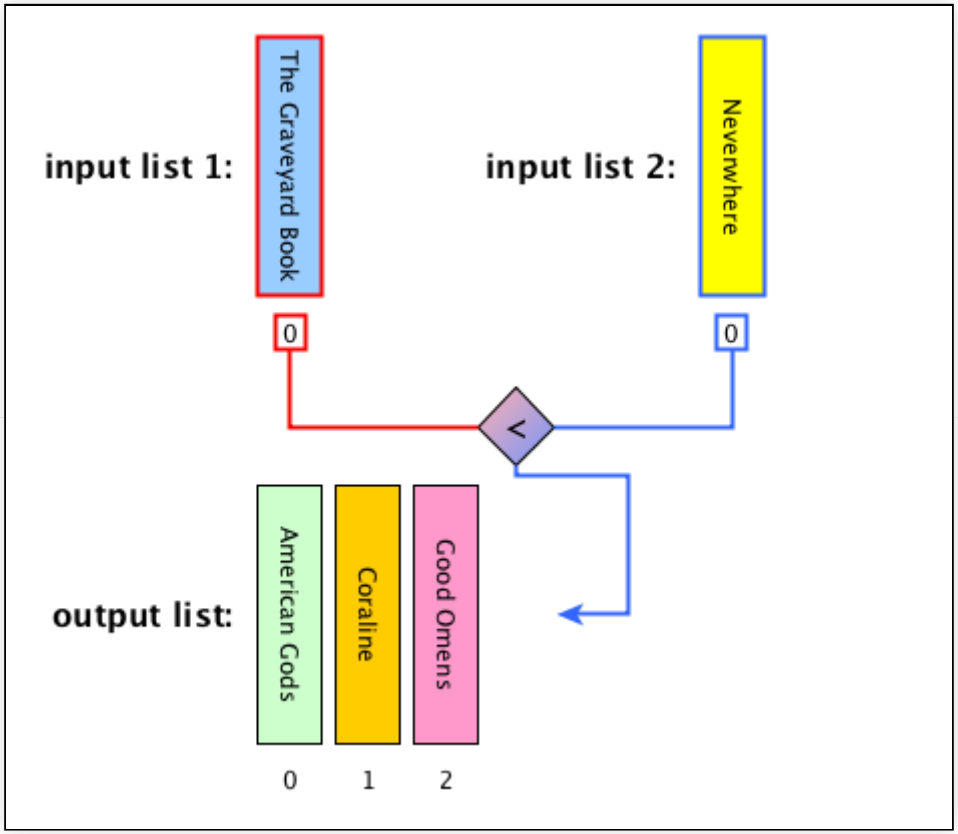
Merge: description



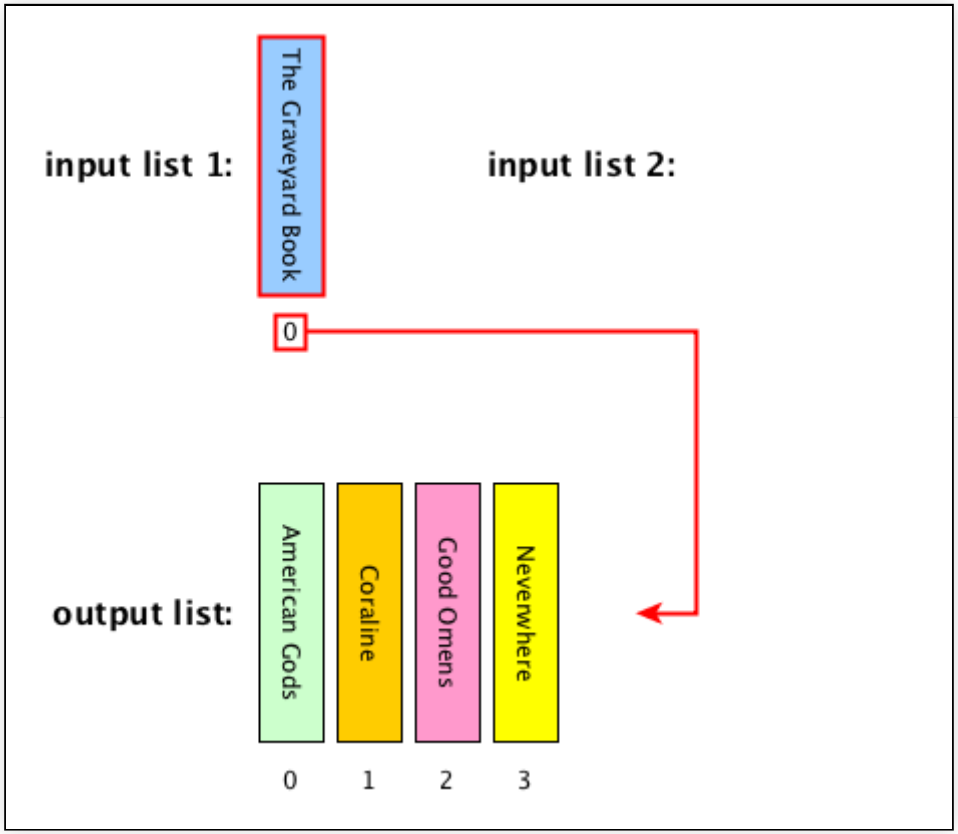
Merge: description



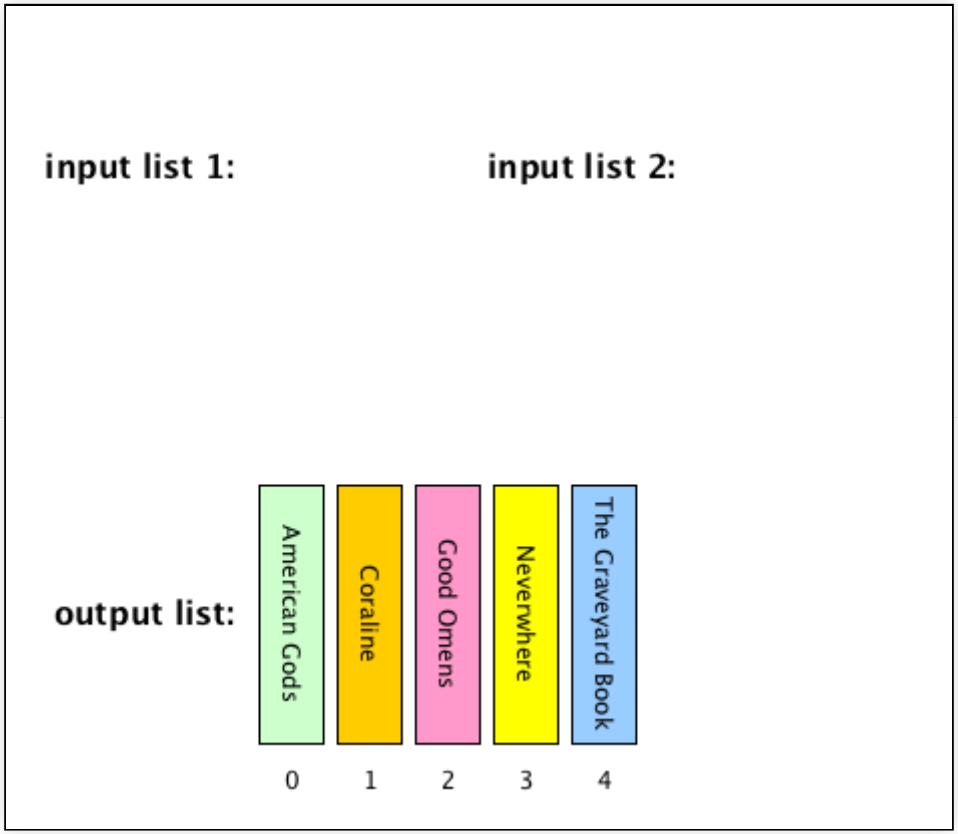
Merge: description



Merge: description



Merge: description



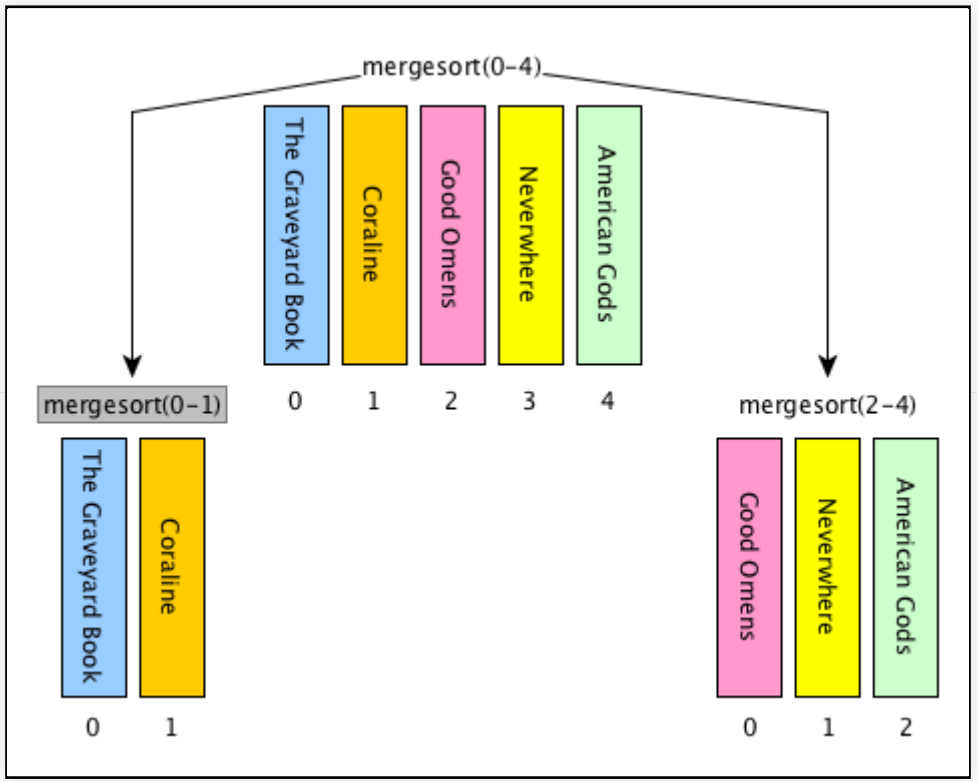
Merge: algorithm

```
def merge(list_1, list_2):
    result = list()

    while len(list_1) > 0 and len(list_2) > 0:
        list_1_item = list_1[0]
        list_2_item = list_2[0]

        if list_1_item <= list_2_item:
            result.append(list_1_item)
            list_1.remove(list_1_item)
        else:
            result.append(list_2_item)
            list_2.remove(list_2_item)
    result.extend(list_1)
    result.extend(list_2)

    return result
```

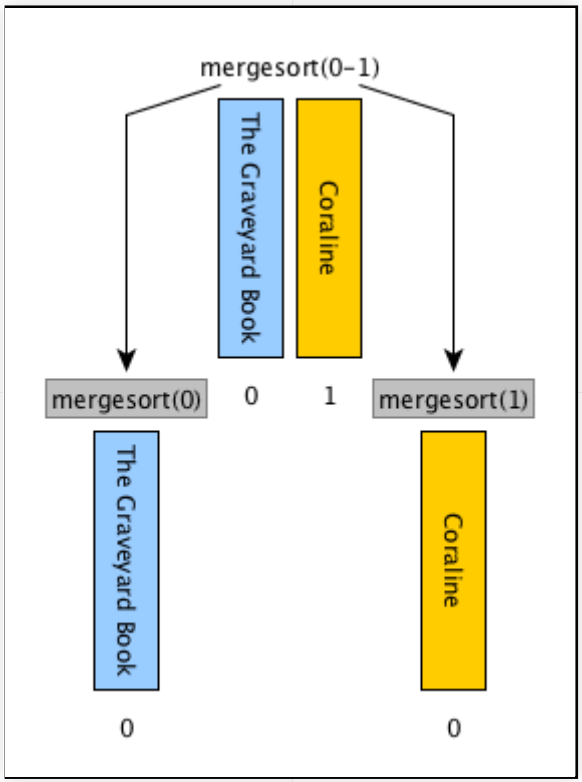
Merge sort: steps

1. **[base case]** if the input list has only one element, return the list as it is, otherwise
2. **[divide]** split the input list into two balanced halves, i.e. containing almost the same number of elements each
3. **[conquer]** run recursively the merge sort algorithm on each of the halves obtained in the previous step
4. **[combine]** merge the two ordered lists returned by the previous step by using `def merge(list_1, list_2)` and return the result

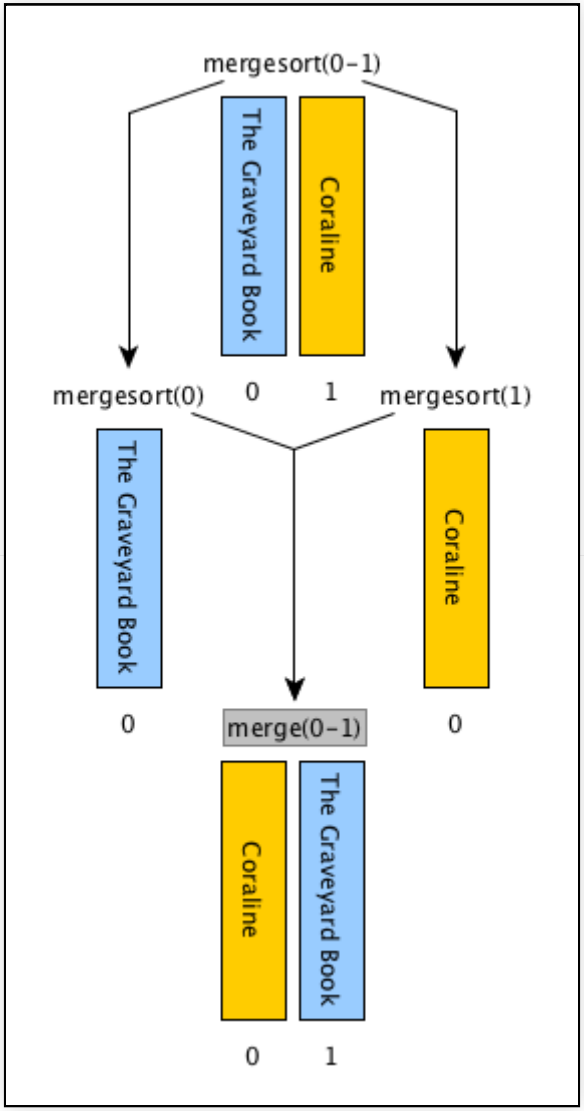
Merge sort: description



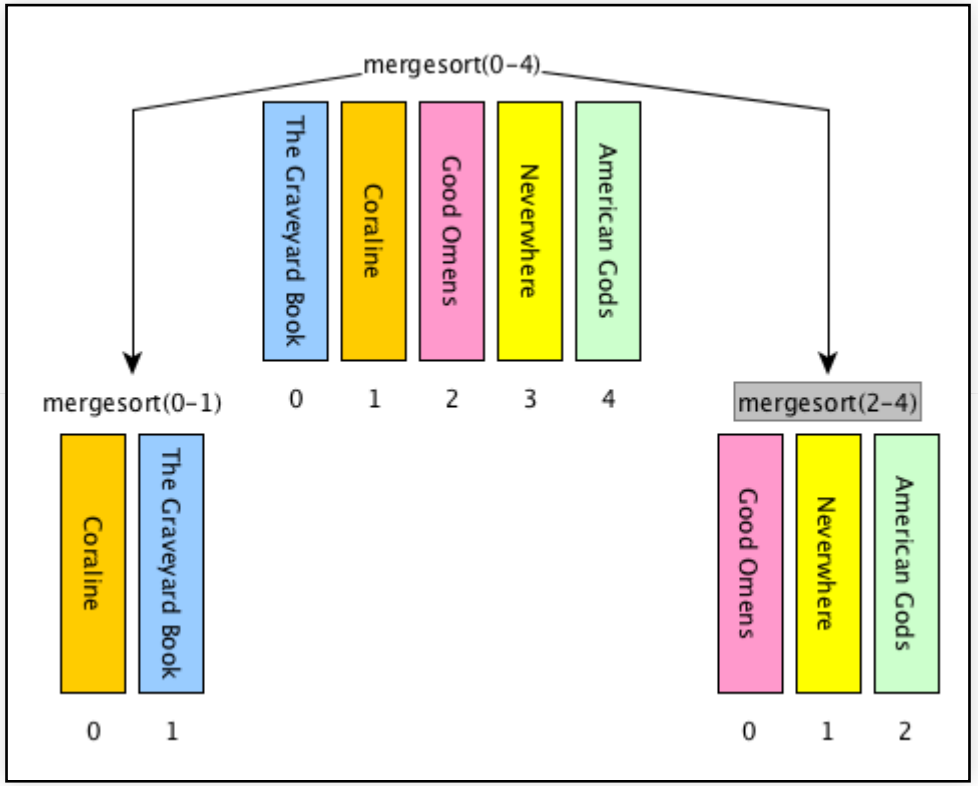
Merge sort: description



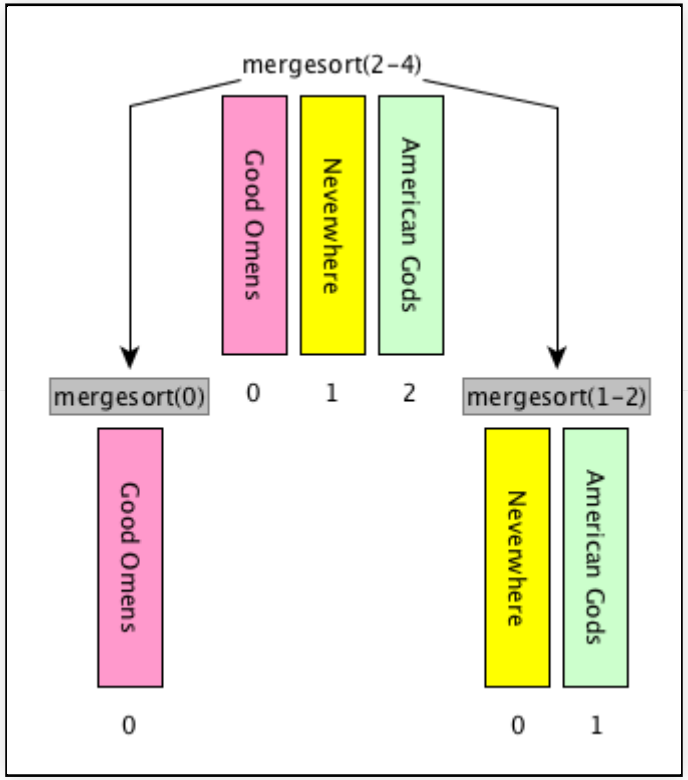
Merge sort: description



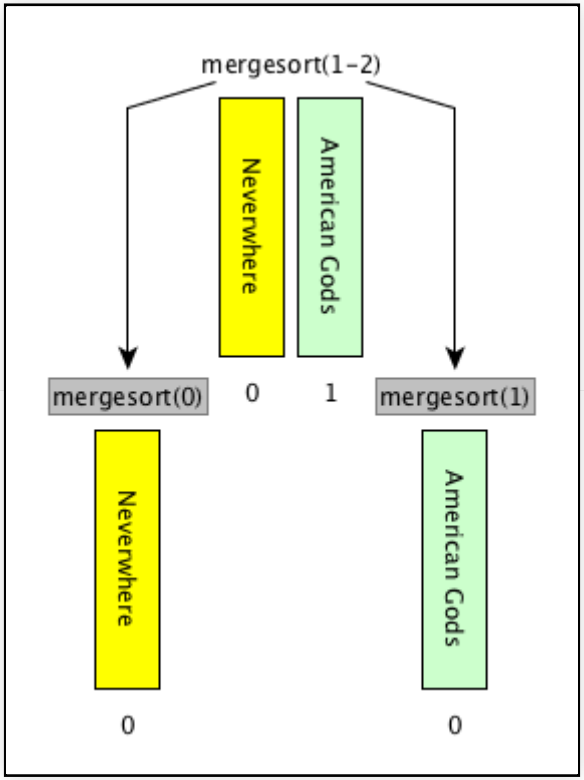
Merge sort: description



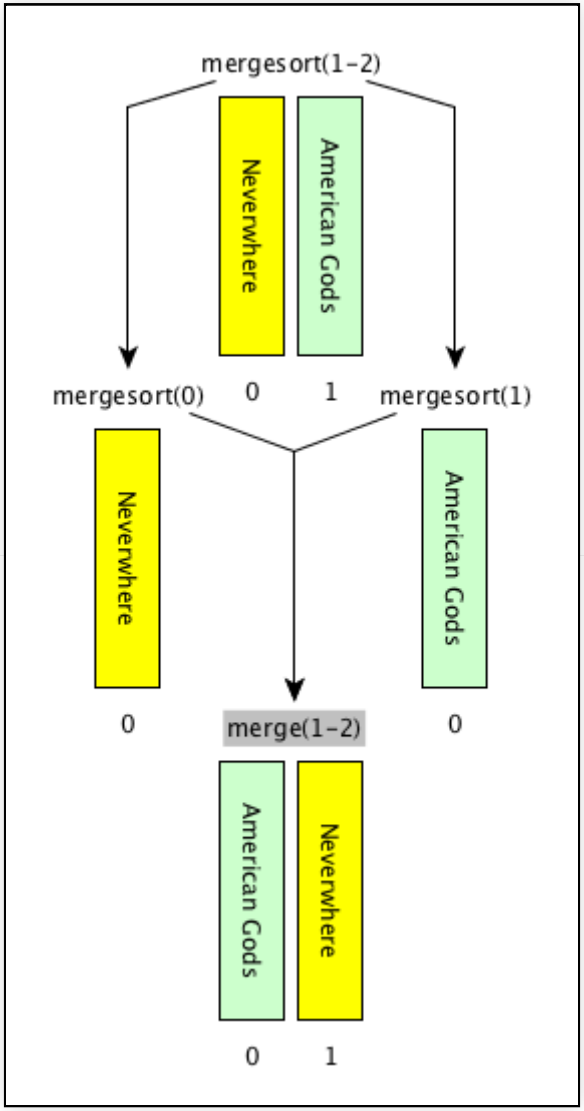
Merge sort: description



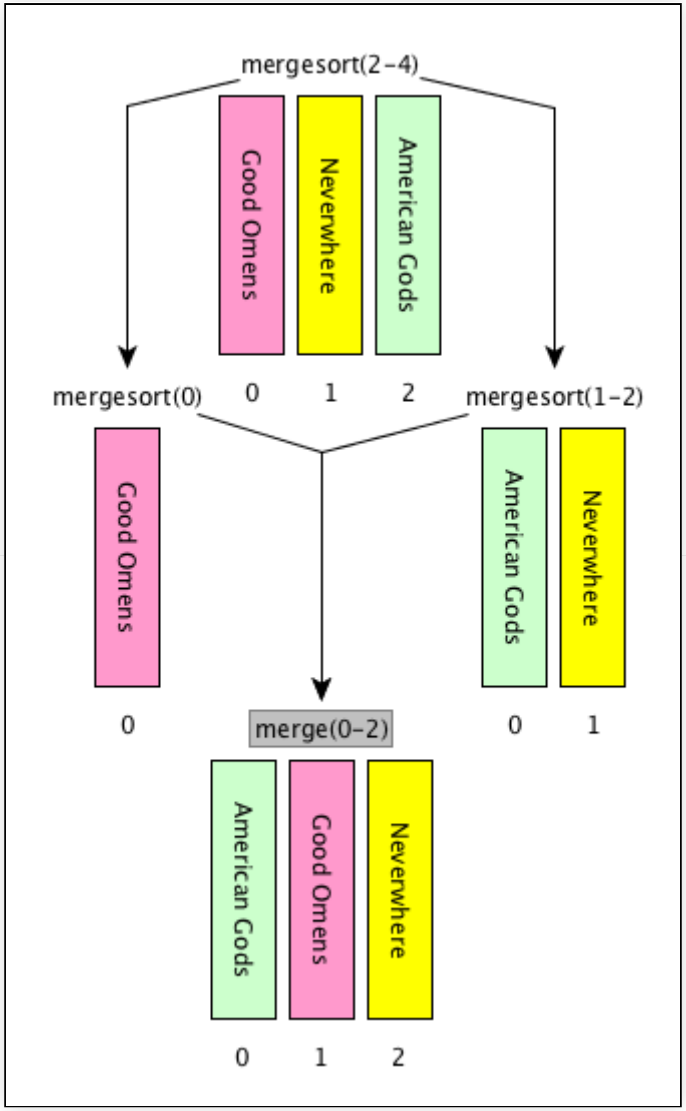
Merge sort: description



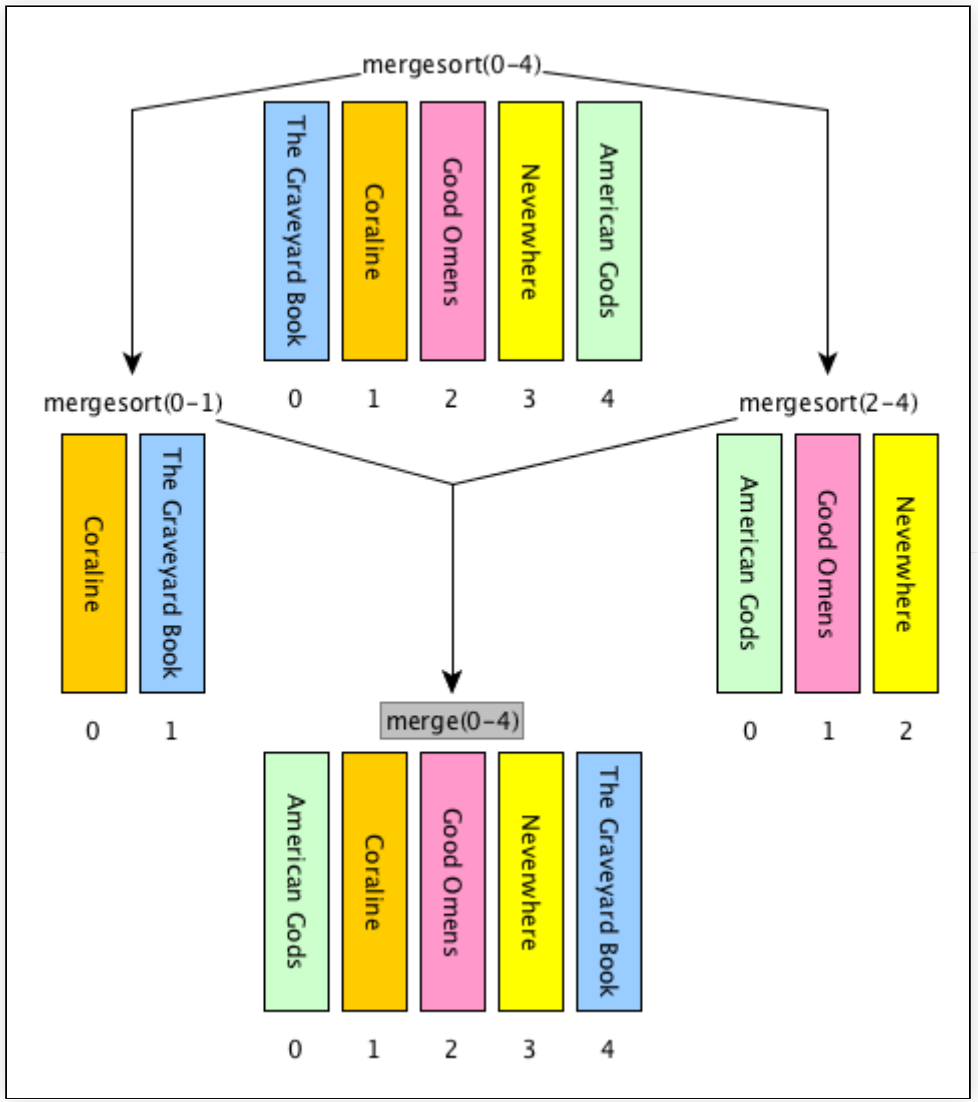
Merge sort: description



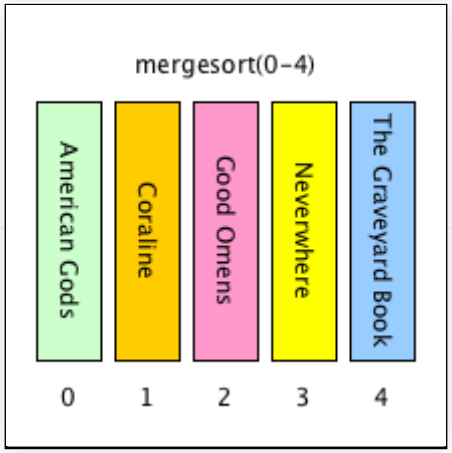
Merge sort: description



Merge sort: description



Merge sort: description



Merge sort: ancillary operations

Floor division: `<number_1> // </number_2>`

It returns only the integer part of the result number discarding the fractional part

E.g.: 3 // 2 = 1, 6 // 2 = 3, 1 // 4 = 0

Create sublist:

`<list>[<start_position>:<end_position>]`

Creates a new list containing all the elements in `<list>` that range from `<start_position>` to `<end_position>-1`

E.g., considering my_list = list(["a", "b", "c"]), my_list[0:1] returns list(["a"]), my_list[1:3] returns list(["b", "c"])

Merge sort: algorithm

```
def mergesort(input_list):
    if len(input_list) <= 1:
        return input_list
    else:
        input_list_len = len(input_list)
        mid = input_list_len // 2

        left = mergesort(input_list[0:mid])
        right = mergesort(input_list[mid:input_list_len])

        return merge(left, right)
```

END

Divide and conquer algorithms

Silvio Peroni

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

Computational Thinking and Programming (A.Y. 2017/2018)

Second Cycle Degree in Digital Humanities and Digital Knowledge

Alma Mater Studiorum - Università di Bologna