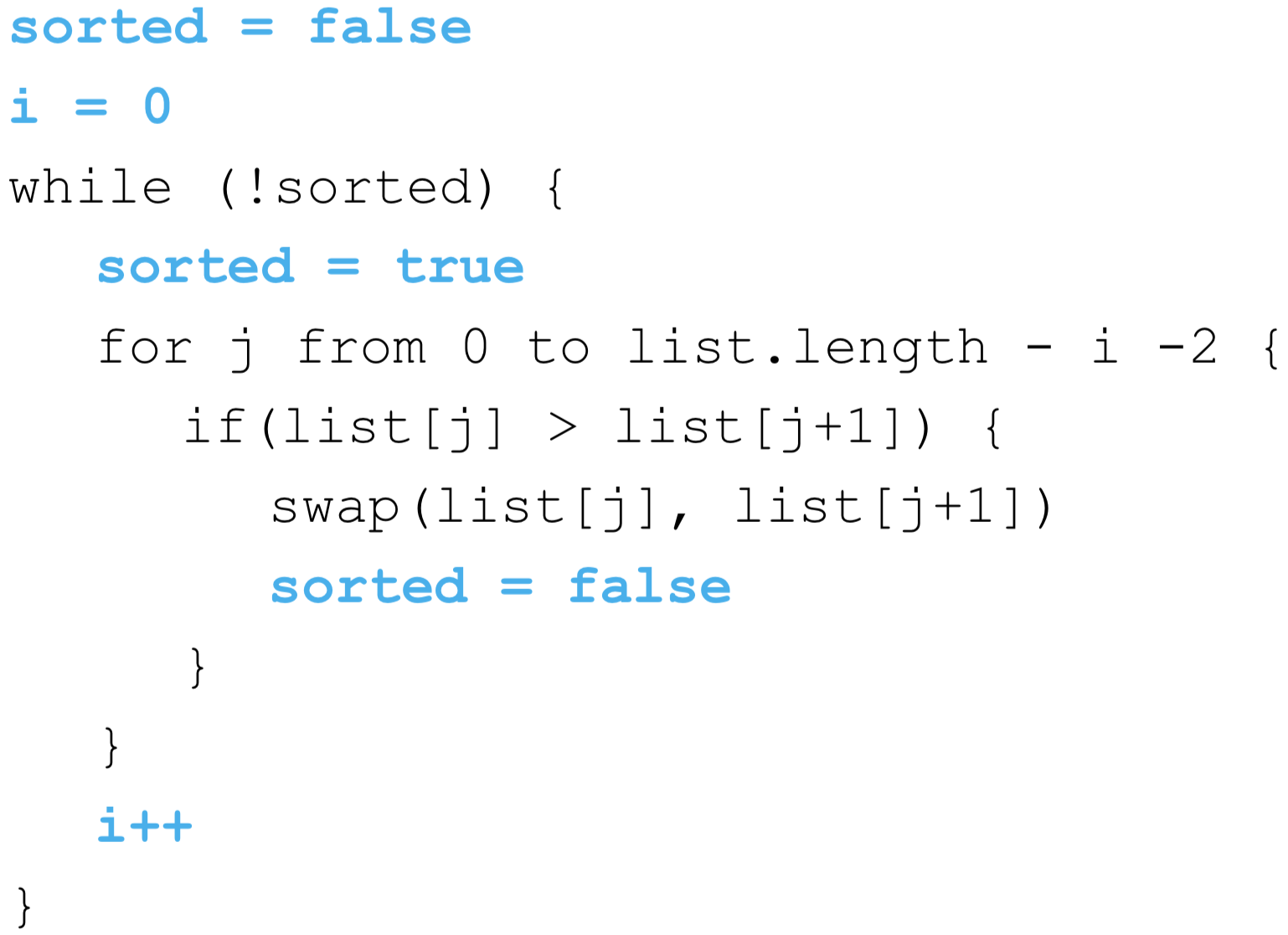
# Quadratic Sorting a List

Bubble Sort

*Goal: order a list of integers in ascending order*

Idea: repeatedly iterate through the list and swap adjacent elements if they are in the wrong order

* Since each time we iterate through the list we ensure that **the largest element is in the correct position.** At each iteration we can stop comparing adjacent elements one step earlier.
* The algorithm can infer that the list is sorted until no swap is needed in last iteration.

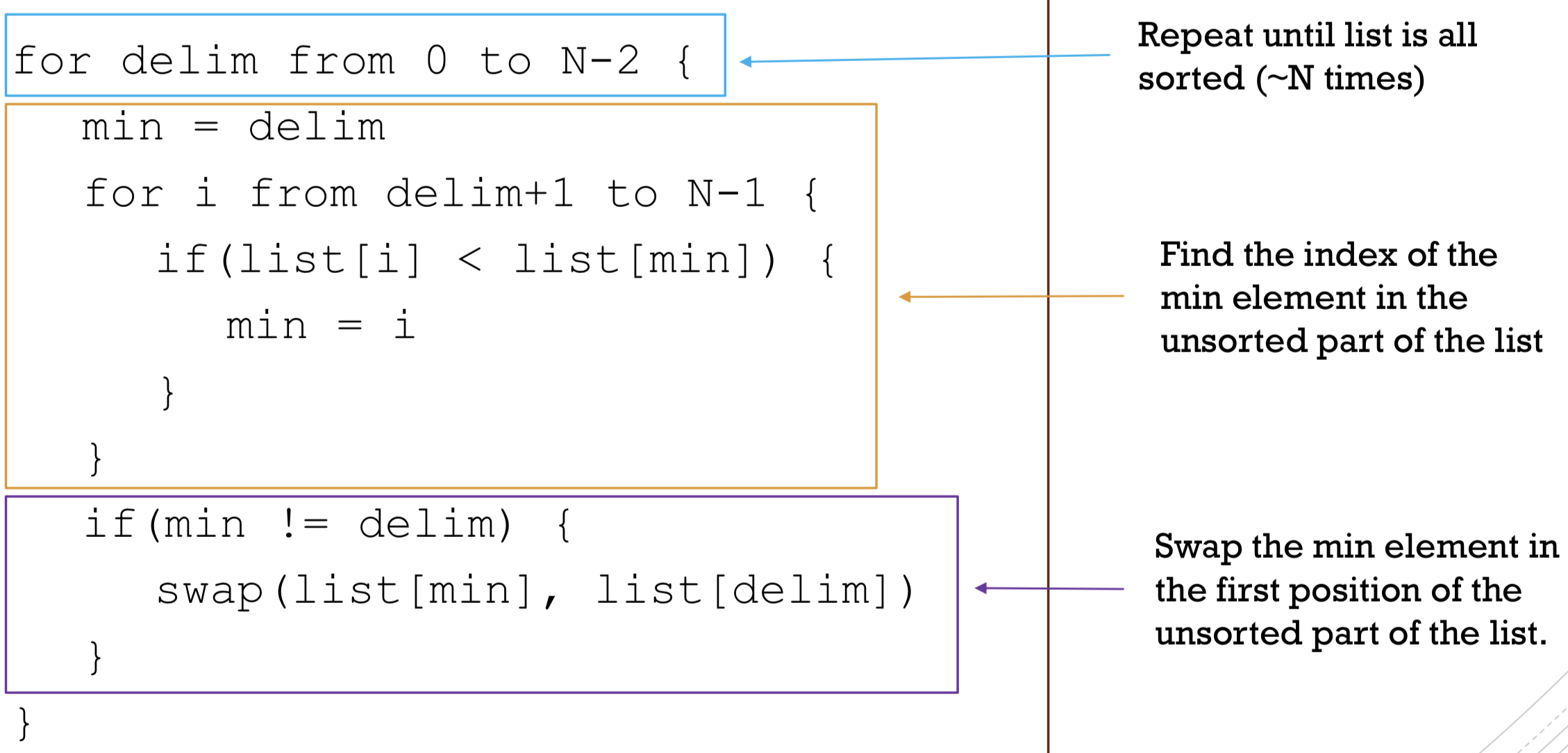


Selection Sort

*Goal: order a list of integers in ascending order*

Idea: consider the list as if it was divided into two parts, one sorted and the other unsorted. (Note that at the beginning the sorted part is empty)

* Select the smallest element in the unsorted part of the list
* Swap that element with the element in the initial position of the unsorted array
* Change where you divide the array from the sorted part to the unsorted part

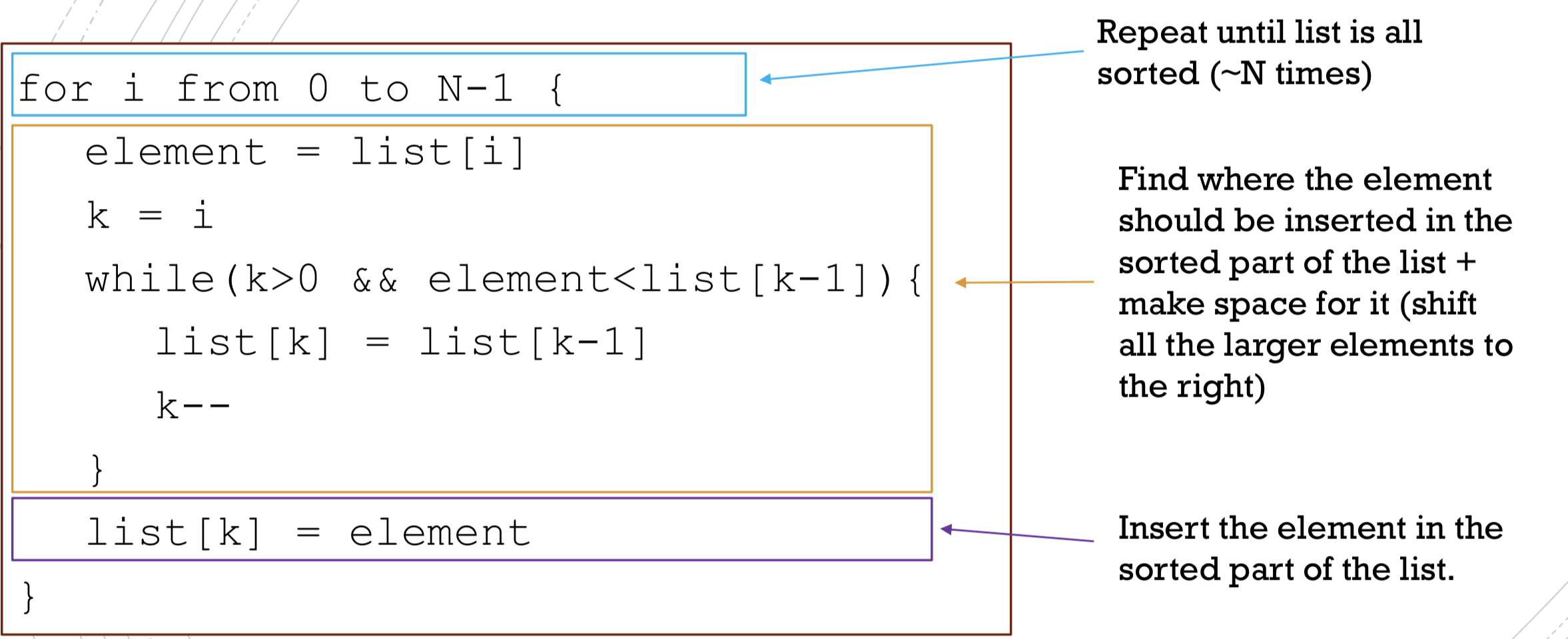


Insertion Sort

*Goal: order a list of integers in ascending order*

Idea: consider the list as if it was divided into two parts, one sorted and the other unsorted. (Note that at the beginning the sorted part is empty)

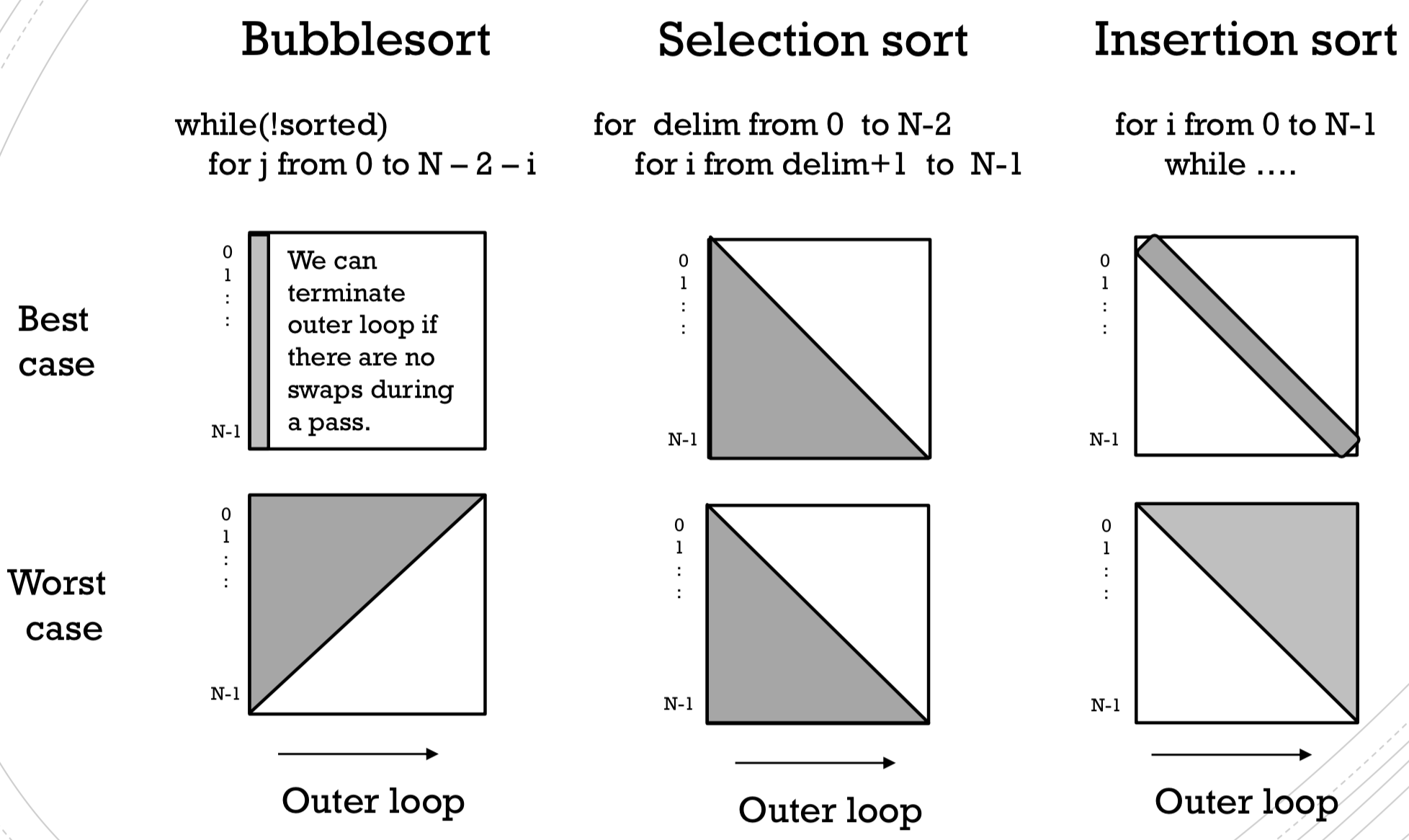
* Select the first element of the unsorted part of the list
* Insert such element into its **correct position** in the sorted part of the list.
* Change where you divide the array from the sorted part to the unsorted part.



[Comparison of the three algorithms]

**Time complexity: O(N2)**

Note that performance depends highly on initial data. Also, it depends on implementation (array vs. linked list), e.g. what is cost of swap and ‘shift’.



# Stack

Abstract data type (ADT)

*“ADT” defines a data type by the* ***values of the data and operations on the data****. It is defined from the point of view of the user. It ignores the details of the implementation. An ADT is more abstract than a data structure.*



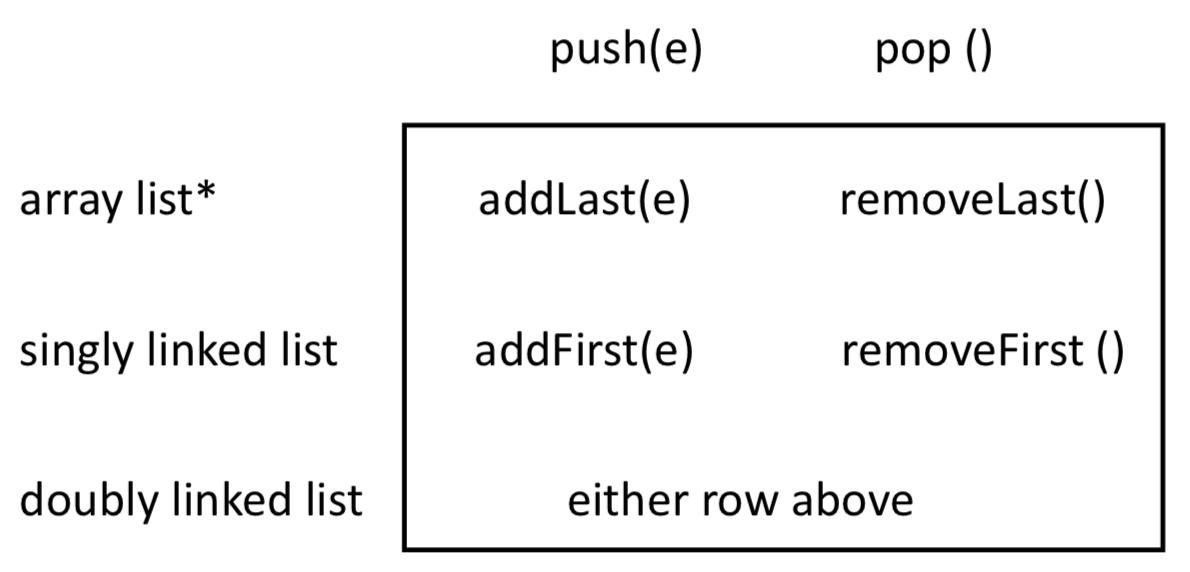
**Stack ADT**

A stack is a list.

However, it typically does NOT have operations to access the list element i directly.

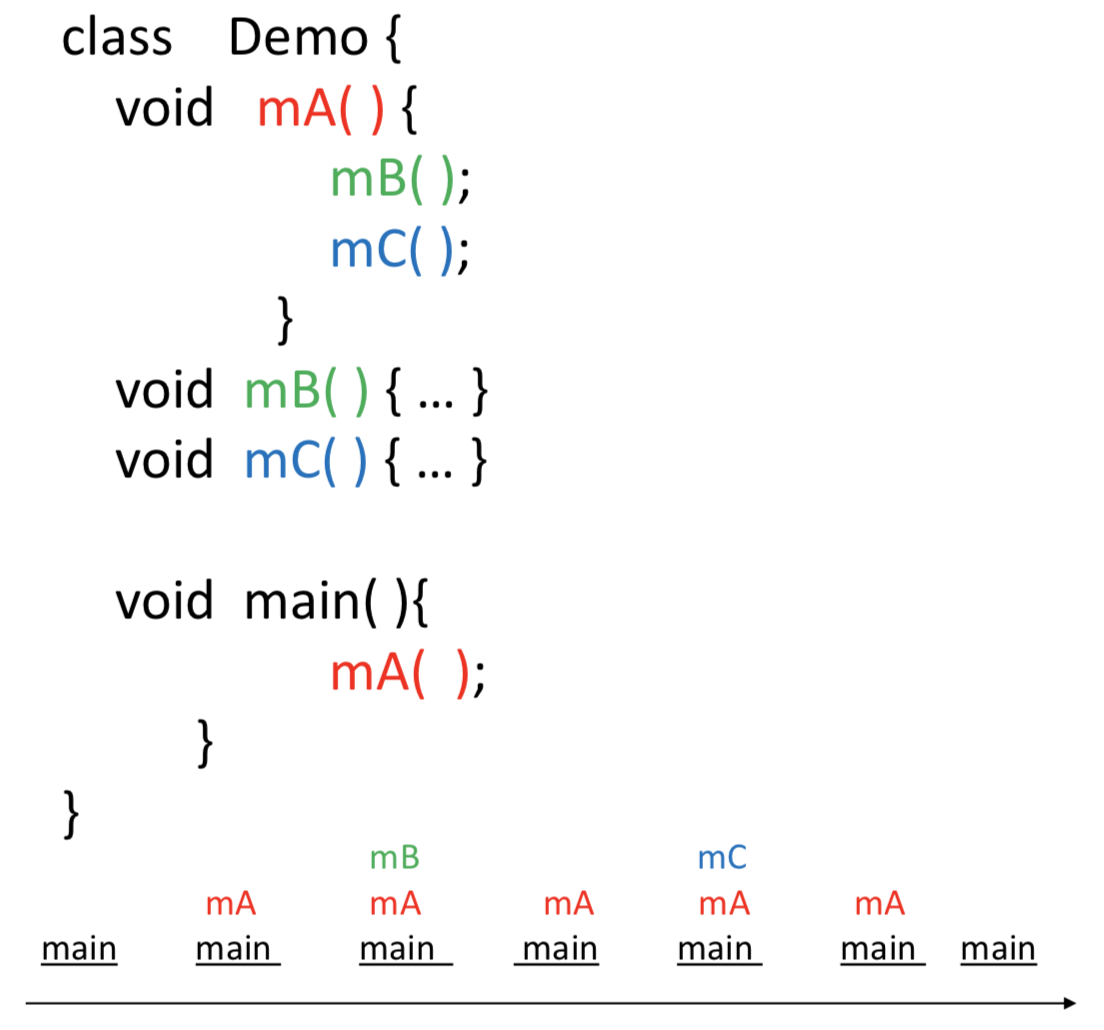
Instead one accesses only one end of the list.

Implement a stack

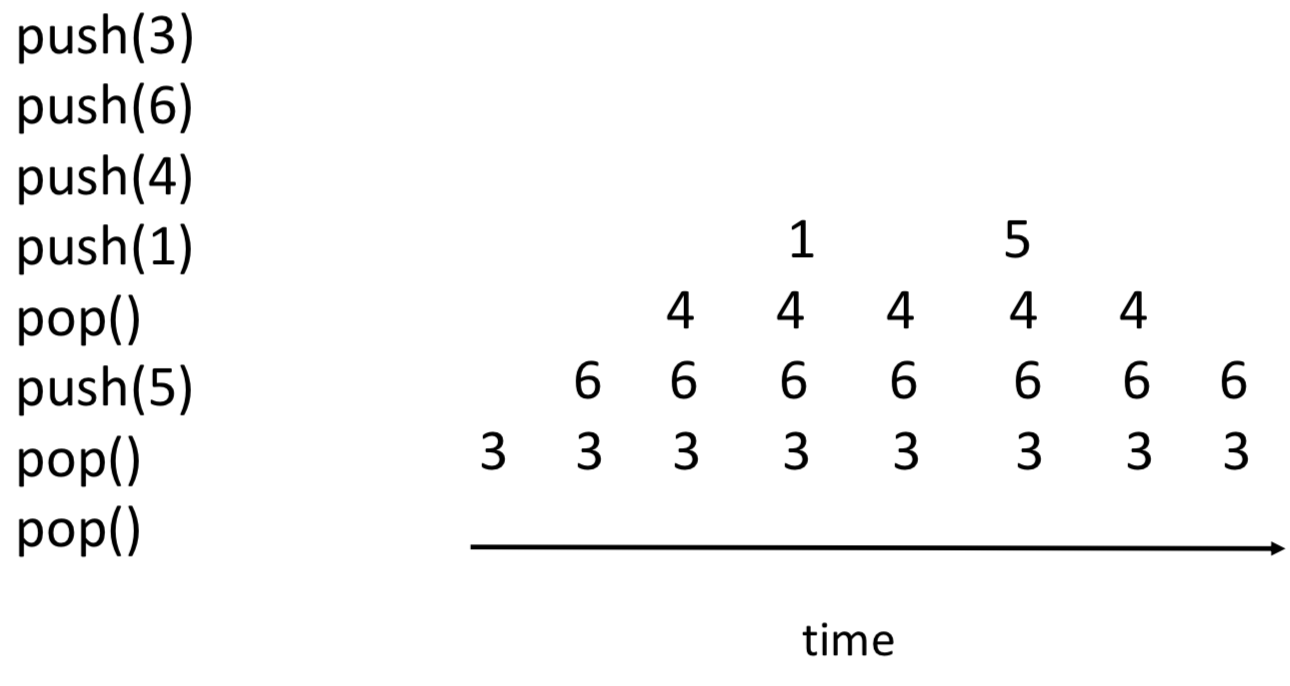


Java ArrayList class doesn’t have addLast and removeLast methods.

* Example 5: “Call Stack”

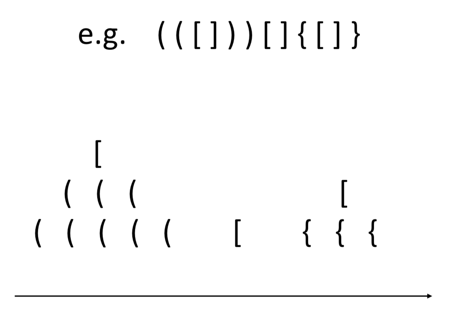
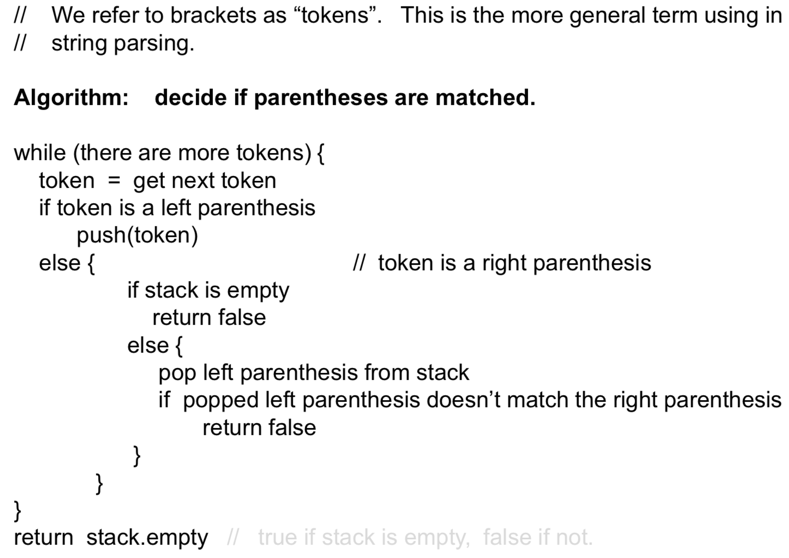


* Example 1: Stack of int



* Example 2: Balancing parentheses

To ensure proper nesting, we traverse the list and use a stack. When we reach a left parenthesis, we push it onto the stack. When we reach a right parenthesis, we compare it to top of the stack. It it matches, then we pop.

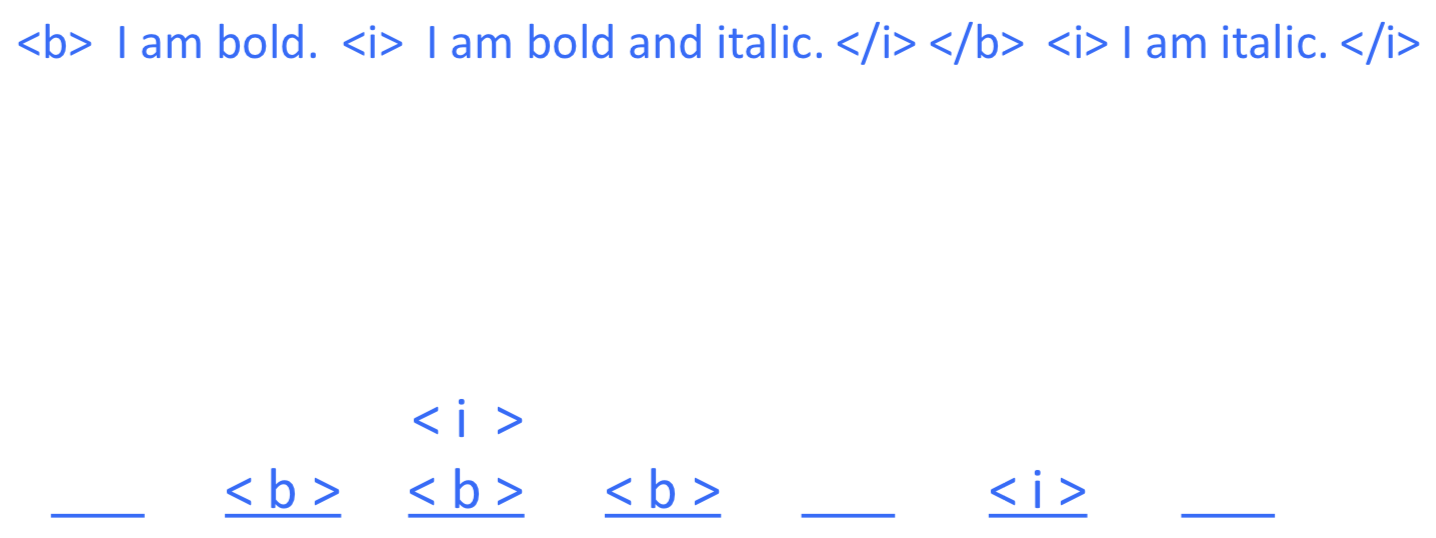


* Example 3: HTML tags

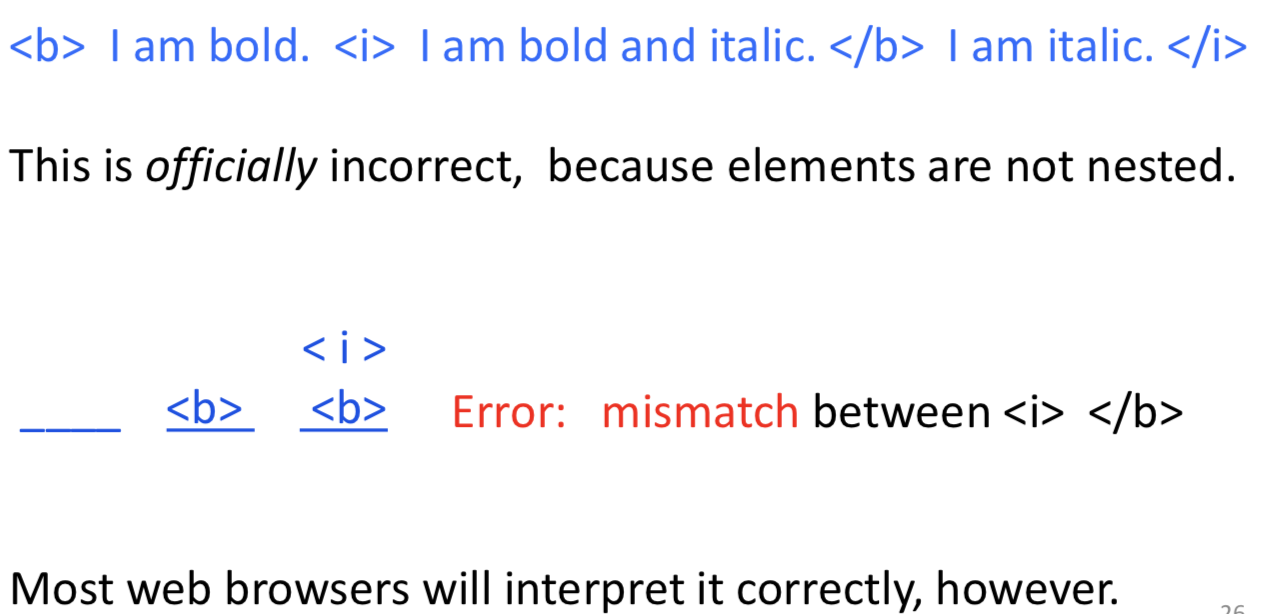
[HTML elements] An HTML element starts with a start tag and ends with an end tag. These tags can be thought of as brackets. HTML documents consist of nested HTML elements.

**I am bold. *I am bold and italic.*** *I am italic.*

Correct way: <b> I am bold. <i> I am bold and italic. </i> </b> <i> I am italic. </i>

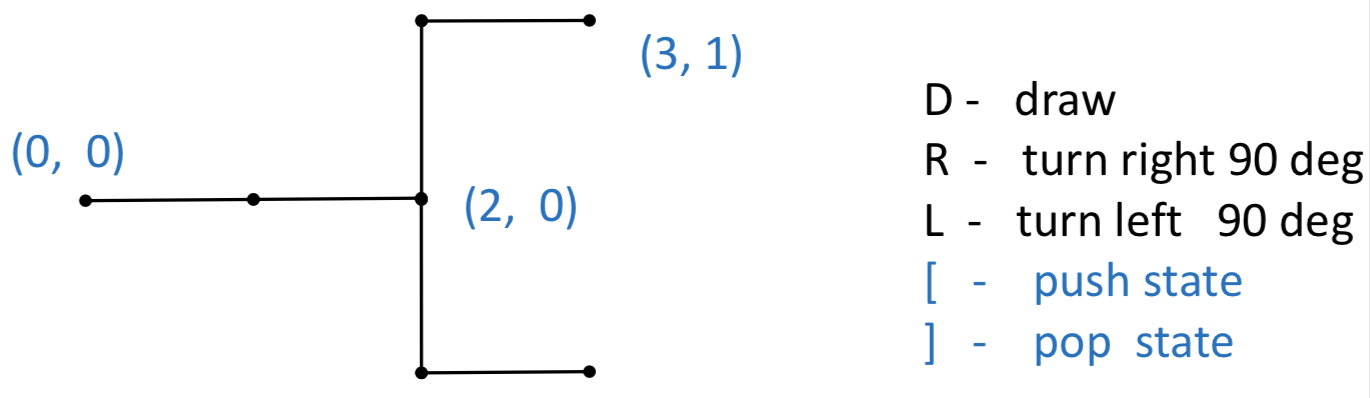


Officially incorrect, because elements are not nested (but most browser interprets correctly): <b> I am bold. <i> I am bold and italic. </b> I am italic. </i>

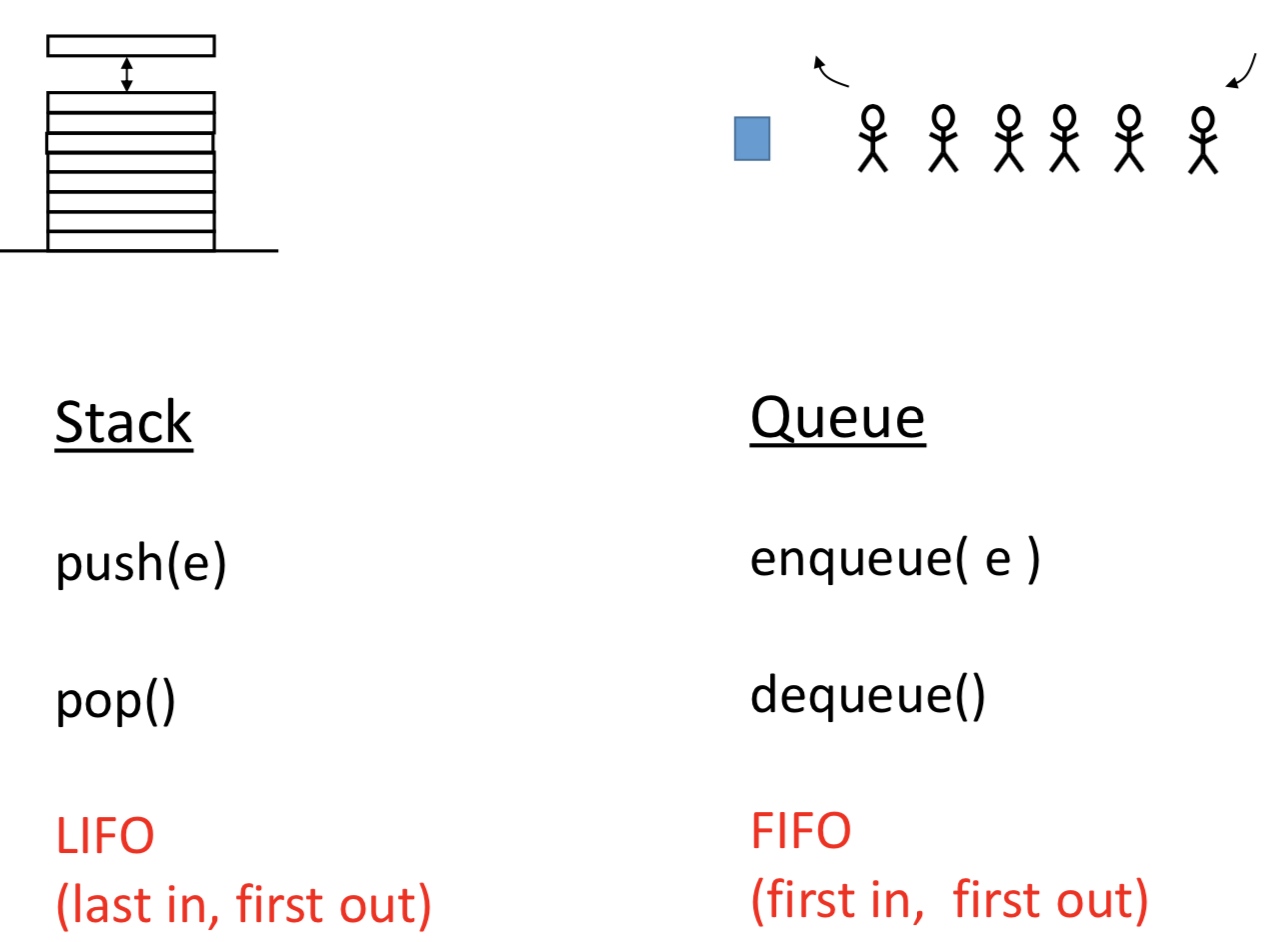
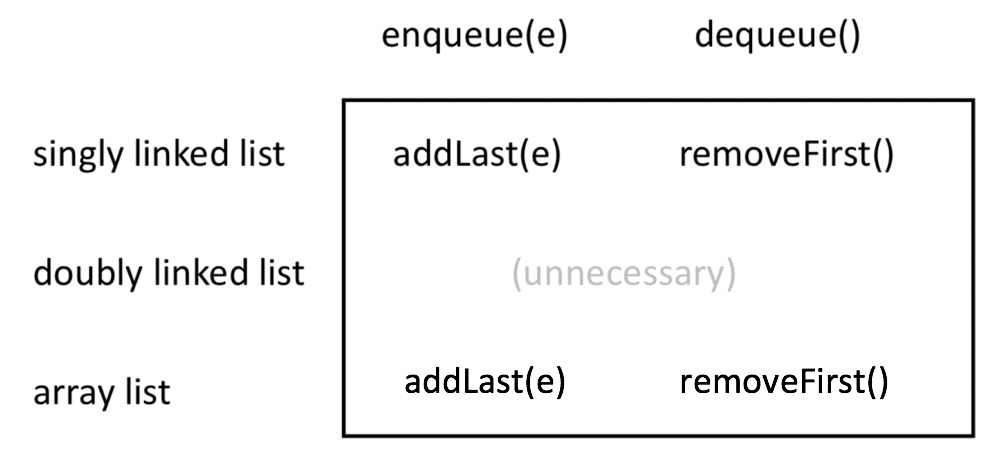
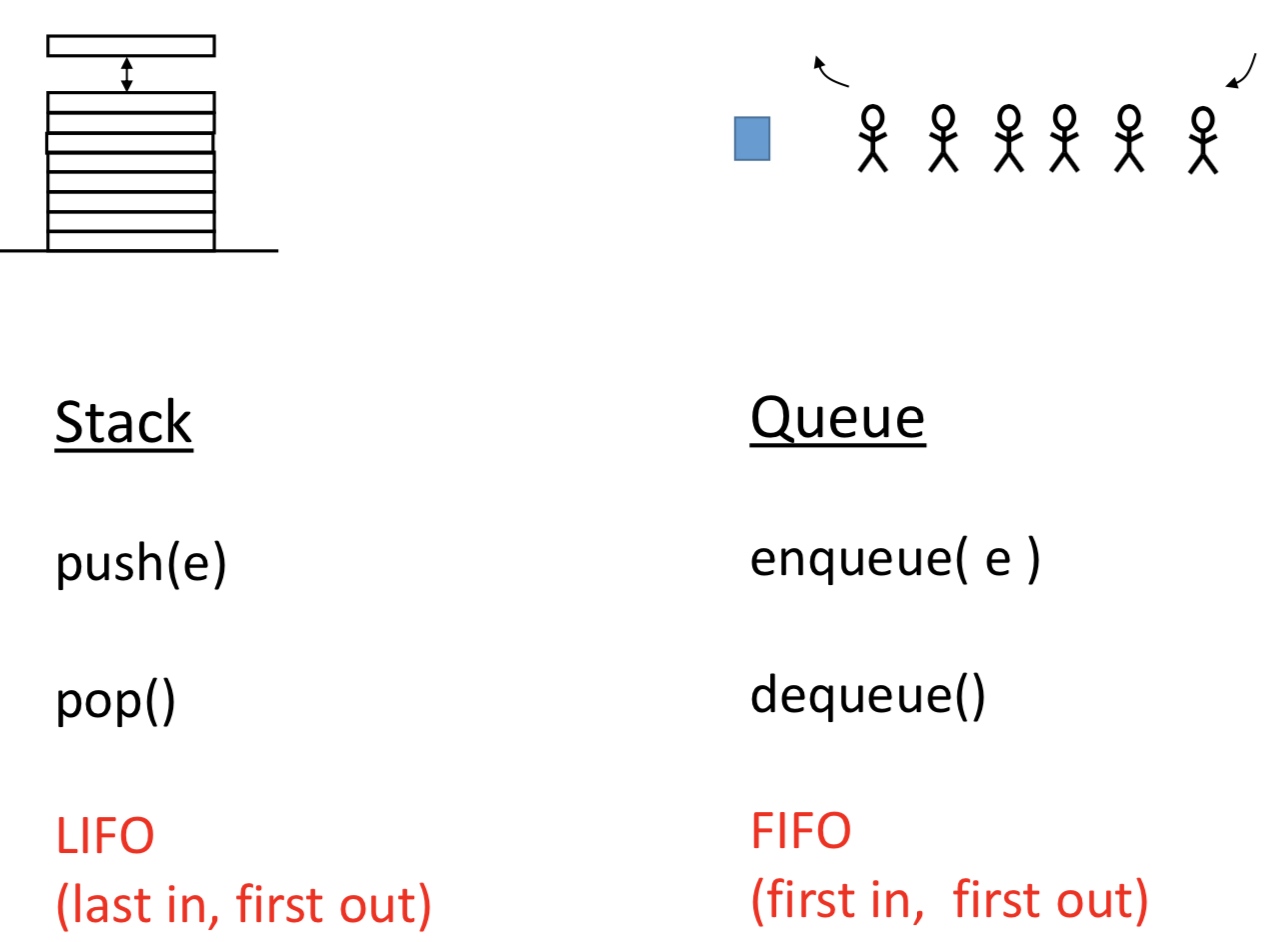


* Example 4: Stacks in Graphics

[ D D [ R D L D ] L D R D ] The pen state will return to (0,0,0)



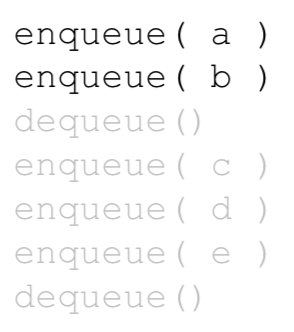
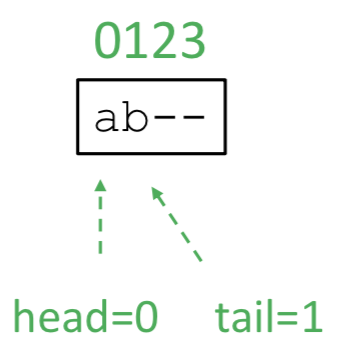
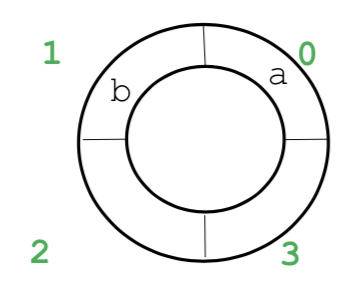
# Queue

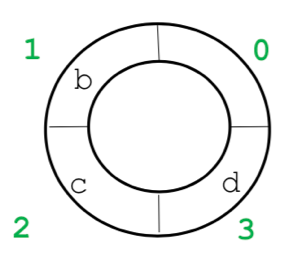
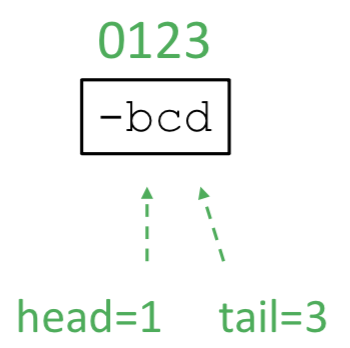
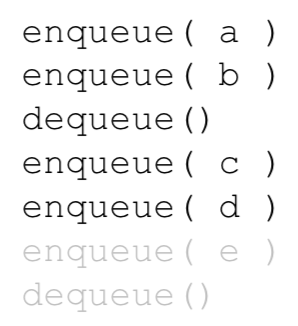
 

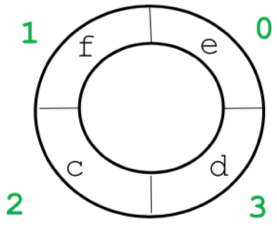
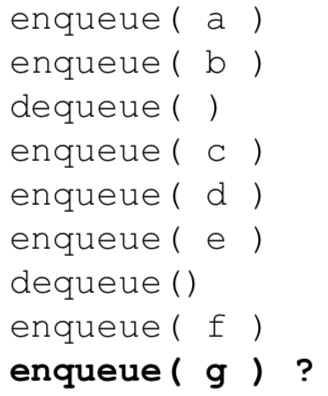
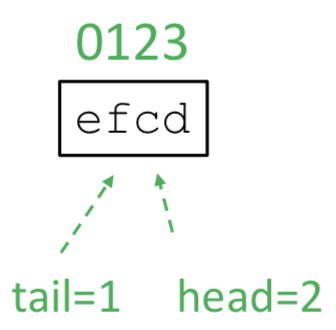
may require expansion

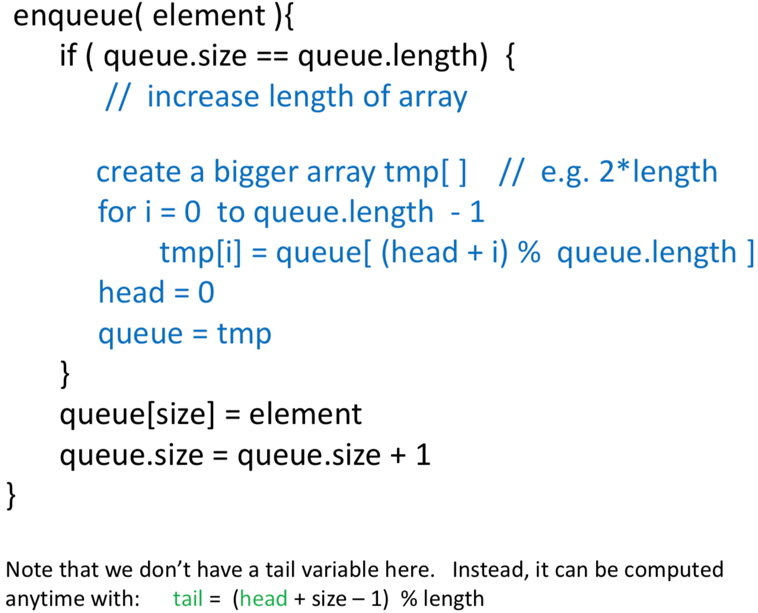
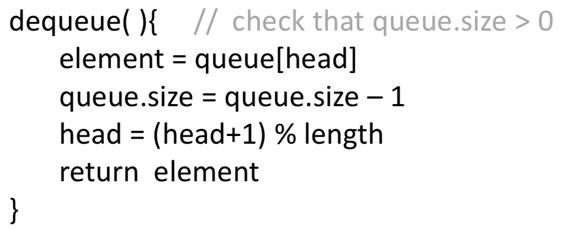
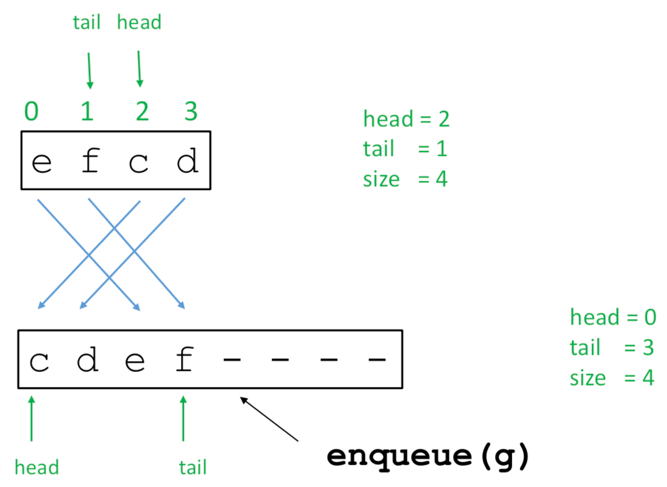
need shift

Circular array **tail= (head+ size –1) % length**

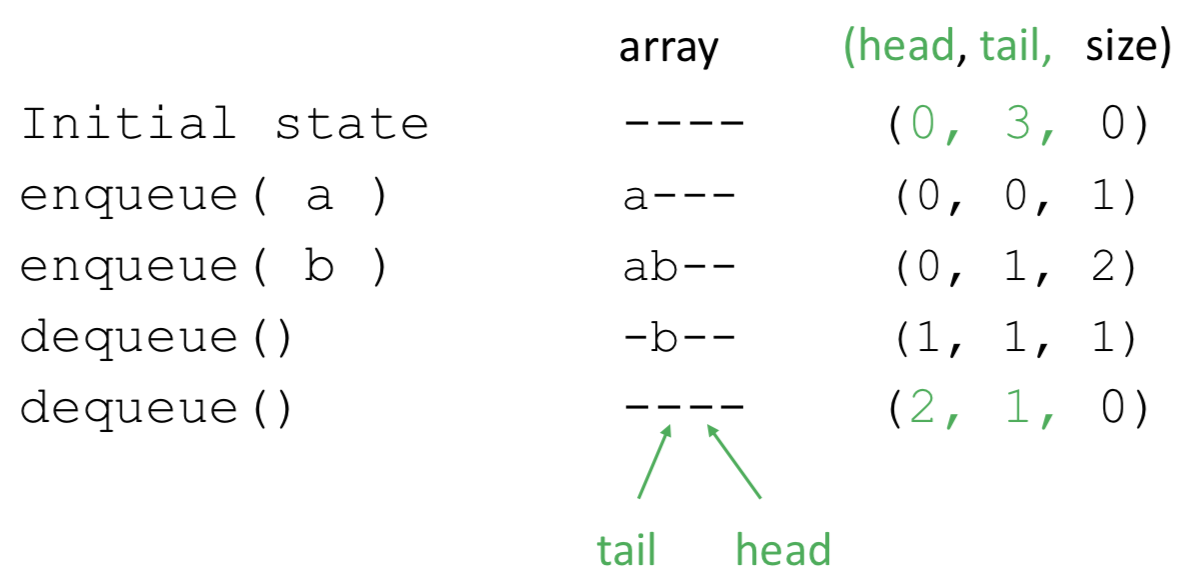
**** **** 

****

**** ****

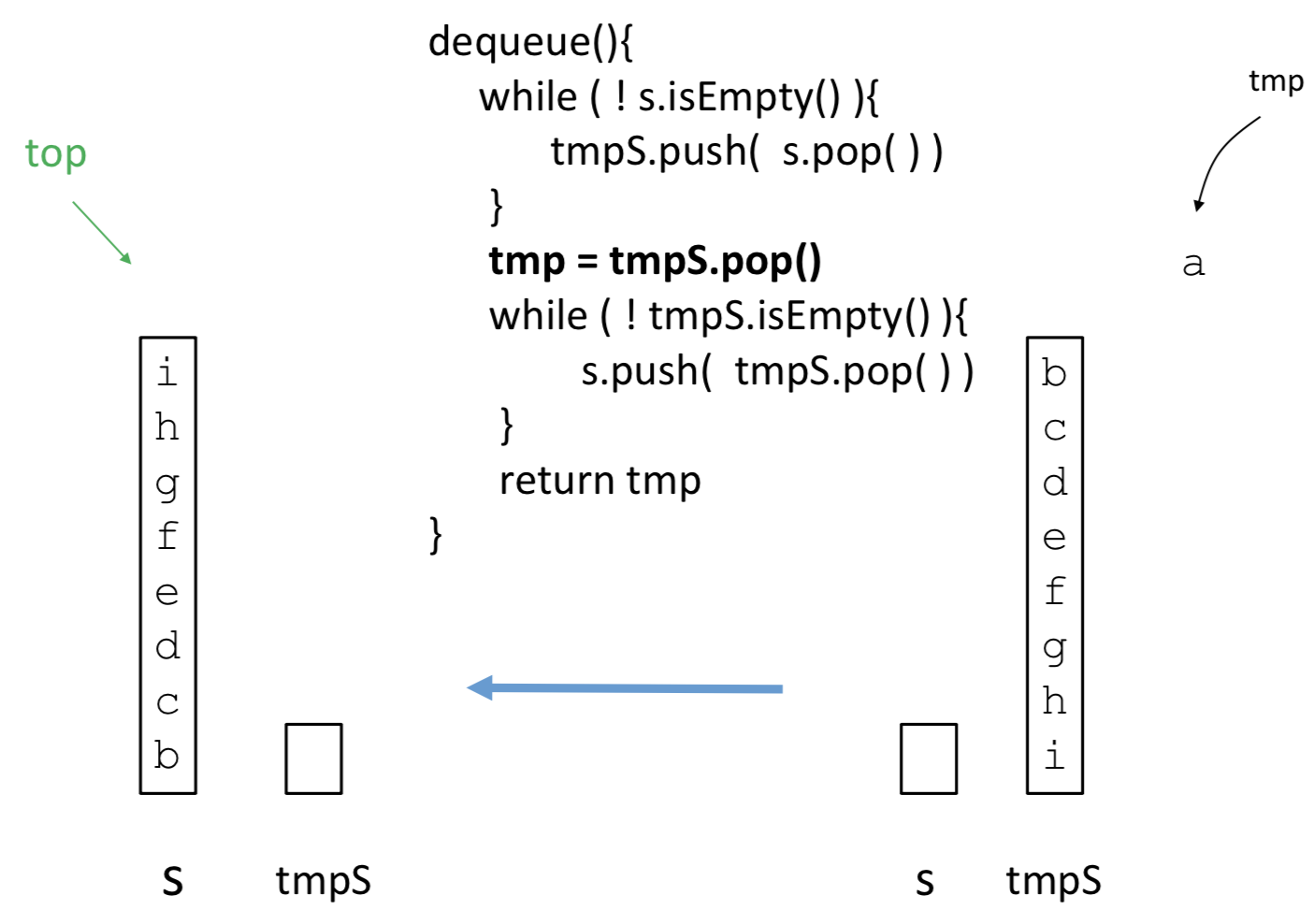
**Increase array length. Copy so that head go to 0.**

What’s the relation between head and tail when size==0?

****

tail= (head+ ~~size~~ –1) % length

Stack (Exercise)

****