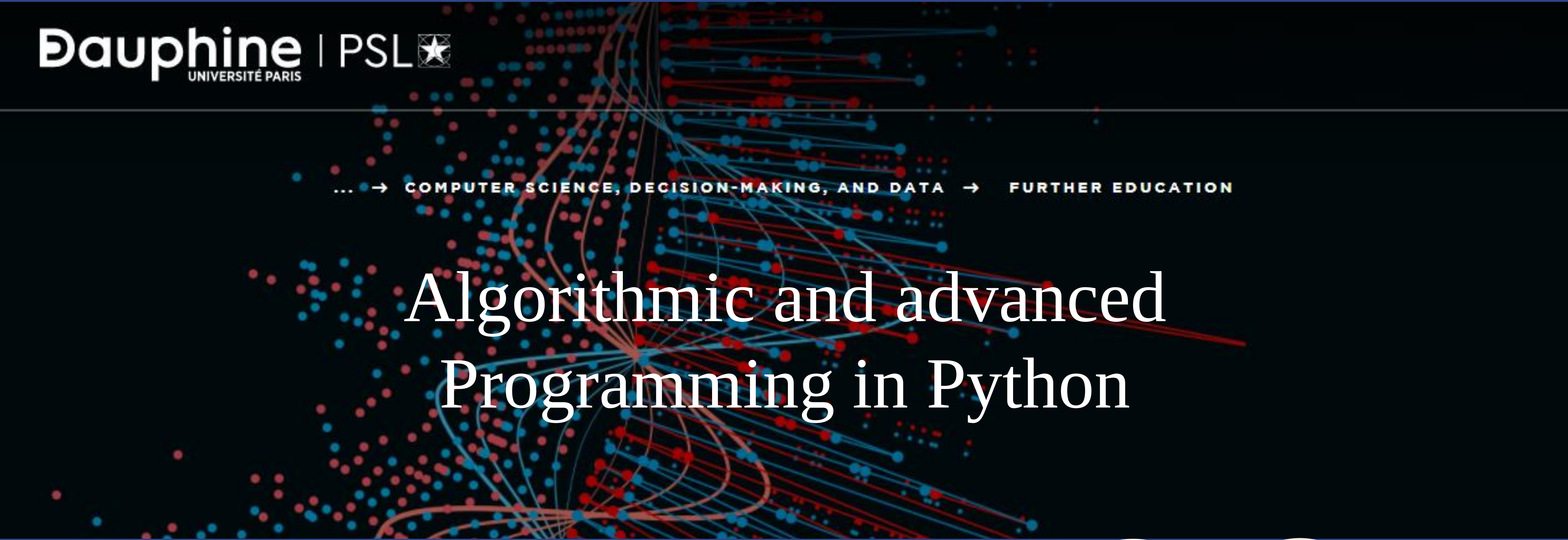


Dauphine UNIVERSITÉ PARIS | PSL

... → COMPUTER SCIENCE, DECISION-MAKING, AND DATA → FURTHER EDUCATION

Algorithmic and advanced Programming in Python

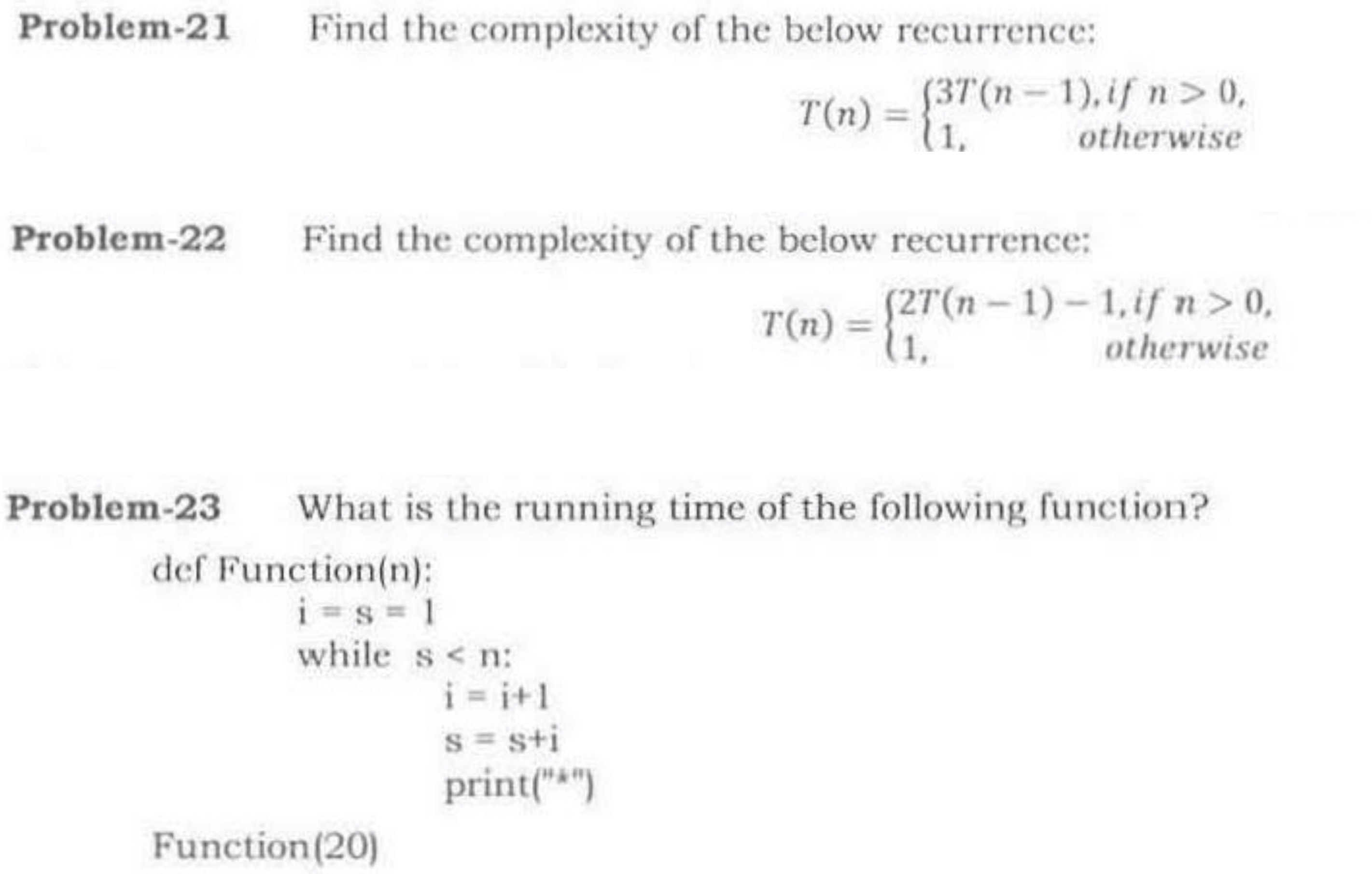
Eric Benhamou eric.benhamou@dauphine.eu
Chien-Chung.Huang chien-chung.huang@ens.fr
Sofía Vázquez sofia.Vazquez@dauphine.eu

Lab 5

Outline

1. Some algorithmic complexity question
2. Algorithm for double linked list
3. Stack and queues codes

Some complexity and running time questions 1/2



Problem-21 Find the complexity of the below recurrence:

$$T(n) = \begin{cases} 3T(n-1), if\ n > 0, \\ 1, \quad otherwise \end{cases}$$

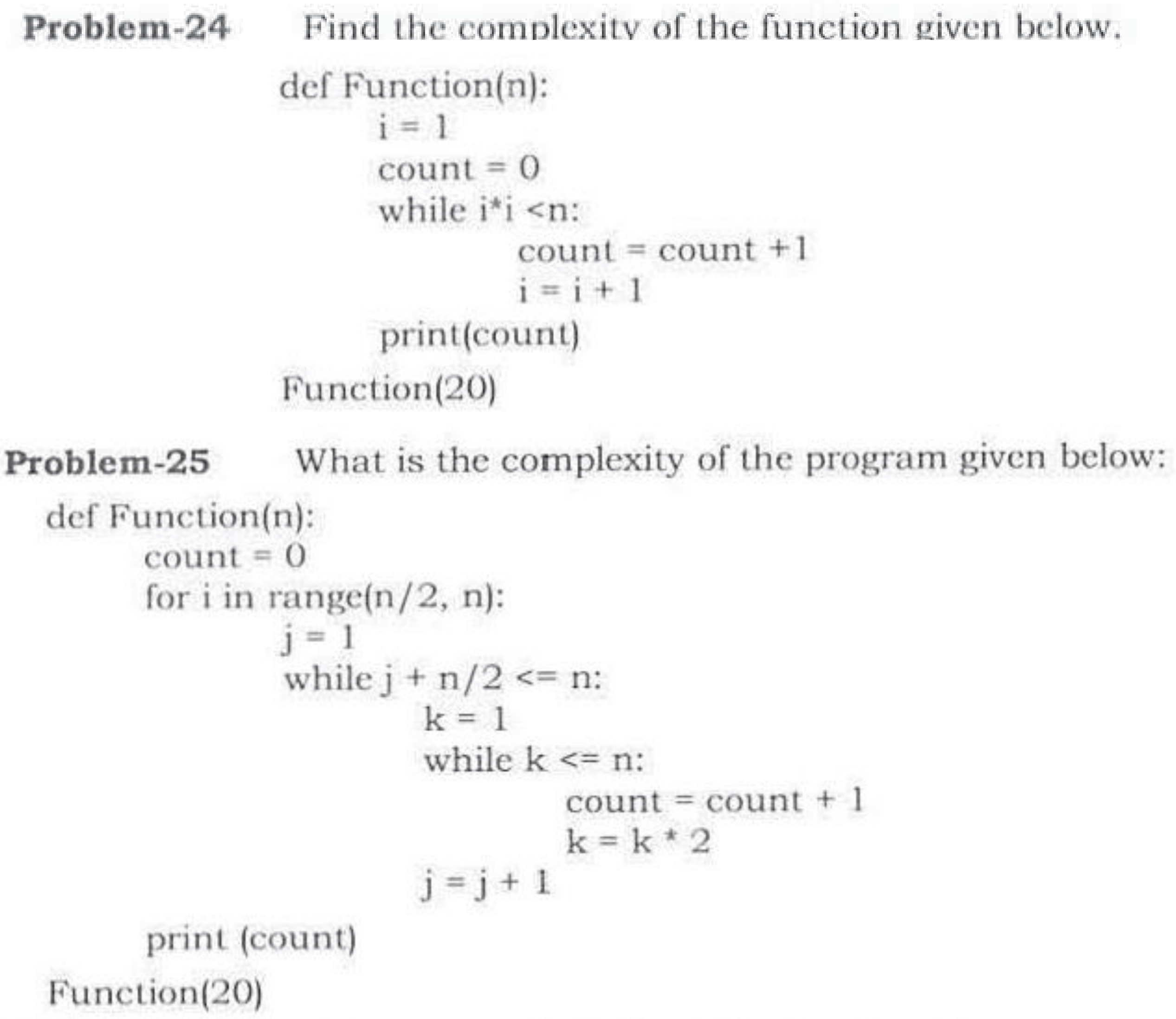
Problem-22 Find the complexity of the below recurrence:

$$T(n) = \begin{cases} 2T(n-1) - 1, if\ n > 0, \\ 1, \quad otherwise \end{cases}$$

Problem-23 What is the running time of the following function?

```
def Function(n):
        i = s = 1
        while  s < n:
                i = i+1
                s = s+i
                print("*")
Function(20)
```

Some complexity and running time questions 2/2



Problem-24 Find the complexity of the function given below.

```
def Function(n):
        i = 1
        count = 0
        while i*i <n:
                count = count +1
                i = i + 1
        print(count)
Function(20)
```

Problem-25 What is the complexity of the program given below:

```
def Function(n):
        count = 0
        for i in range(n/2, n):
                j = 1
                while j + n/2 <= n:
                        k = 1
                        while k <= n:
                                count = count + 1
                                k = k * 2
                        j = j + 1
        print (count)
Function(20)
```

Now play with linked list

- Download the file

Advanced Programming & Algo - 1 - Lab resource.py

Now play with linked list

- The file Advanced Programming & Algo - 1 - Lab resource.py

in moodle contains an incomplete implementation of a Python LinkedList class. Take a minute to look over this code. Open a Python interpreter and experiment with creating a LinkedList object and calling the methods that have already been implemented.

Exercise: question 1

1. Implement the count method, which should return a count of the number of times that the given item is found in the list.

Question 2: Index method

- Implement the index method. This will be very similar to the included __contains__ method, except that it needs to return the index of the element if it is found, rather than a simple boolean. Thus, you will need to track the current index as you traverse the linked list.

Question 3

- Implement the append method, which should add a new element onto the tail of the list. You must also remember to handle the special case when the list is empty. Given the current implementation, there is no O(1) way to add an element to the tail of the list. You have two options to implement this function:
- Iterate to the end of the list, finding the last node and adding the new node after that node. This will be O(n) but that is ok for the purposes of this lab.
- Add a _tail reference to the LinkedList class and use it to add a new item in O(1) time. This is a better solution, but will require you to change several other functions to properly maintain the tail pointer.

Question 4: equal and not equal

- Implement the __eq__ and __ne__ methods. For these functions, equality should be defined as follows: both lists have the same number of elements, and each pair of corresponding elements in the list are also equal (as defined by the == operator). You should implement only one of these operators from scratch; the other should delegate to the first.

Problem 1

- Implement stack with fixed size array

Problem 2

- Implement stack with dynamic array

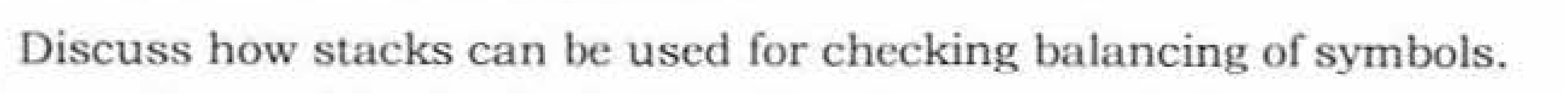
Problem 3

- Implement stack with linked list

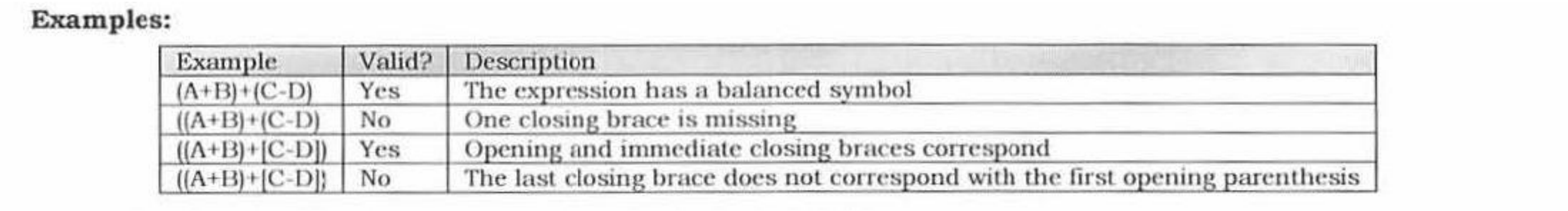
Problem 4

- Implement stack with queues

Problem 5 : discuss and implement



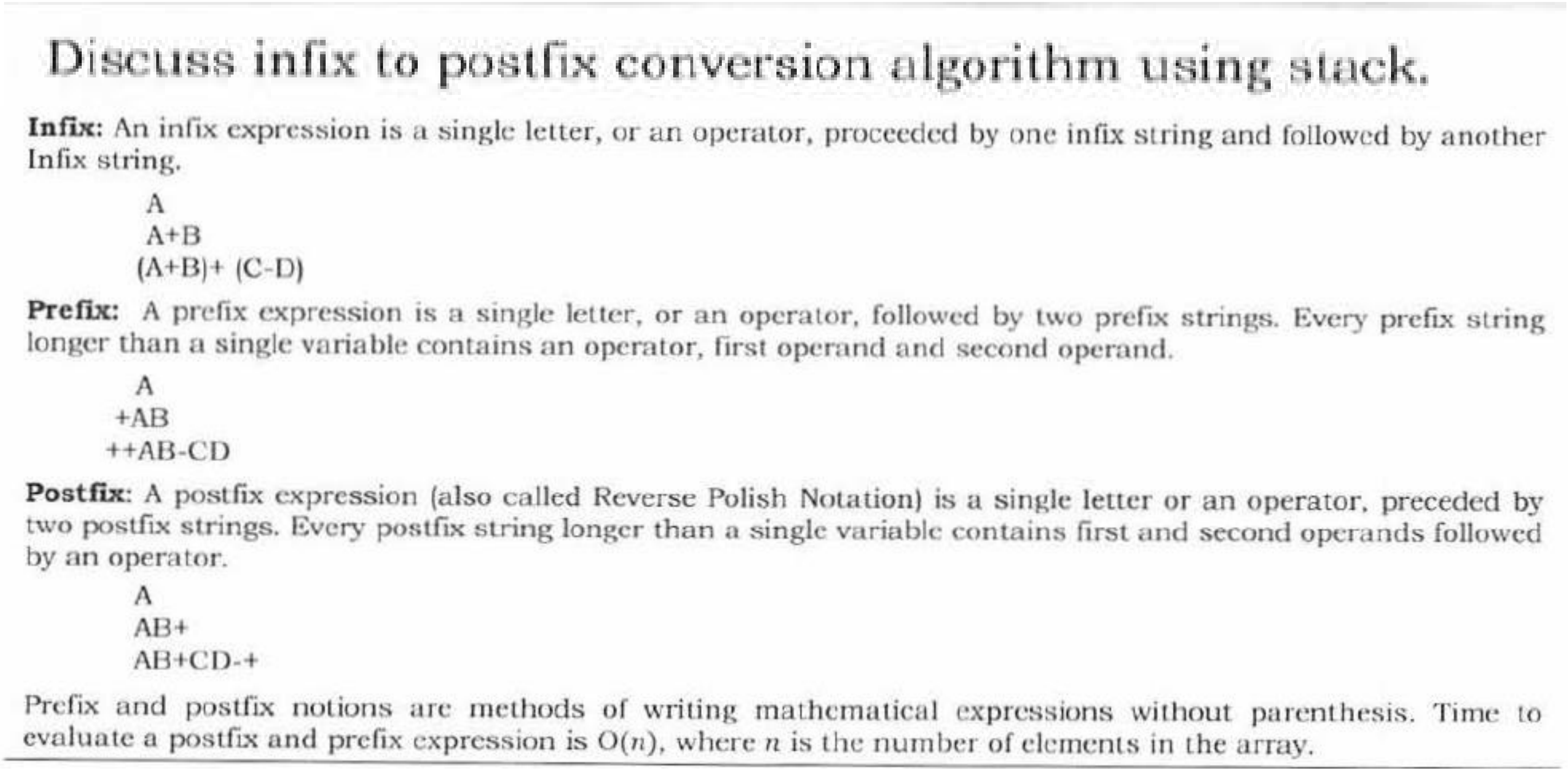
Discuss how stacks can be used for checking balancing of symbols.



Examples:

Example	Valid?	Description
(A+B)+(C-D)	Yes	The expression has a balanced symbol
((A+B)+(C-D)	No	One closing brace is missing
((A+B)+[C-D])	Yes	Opening and immediate closing braces correspond
((A+B)+[C-D]}	No	The last closing brace does not correspond with the first opening parenthesis

Problem 6: discuss and implement



Discuss infix to postfix conversion algorithm using stack.

Infix: An infix expression is a single letter, or an operator, proceeded by one infix string and followed by another Infix string.

A
A+B
(A+B)+ (C-D)

Prefix: A prefix expression is a single letter, or an operator, followed by two prefix strings. Every prefix string longer than a single variable contains an operator, first operand and second operand.

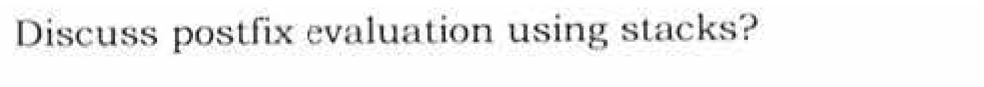
A
+AB
++AB-CD

Postfix: A postfix expression (also called Reverse Polish Notation) is a single letter or an operator, preceded by two postfix strings. Every postfix string longer than a single variable contains first and second operands followed by an operator.

A
AB+
AB+CD-+

Prefix and postfix notions are methods of writing mathematical expressions without parenthesis. Time to evaluate a postfix and prefix expression is $O(n)$, where n is the number of elements in the array.

Problem 7: discuss and implement



Discuss postfix evaluation using stacks?