LAB 2 : RECURSIVITATE

Recursivitate : [pe stiva
pe coada
arborescenta

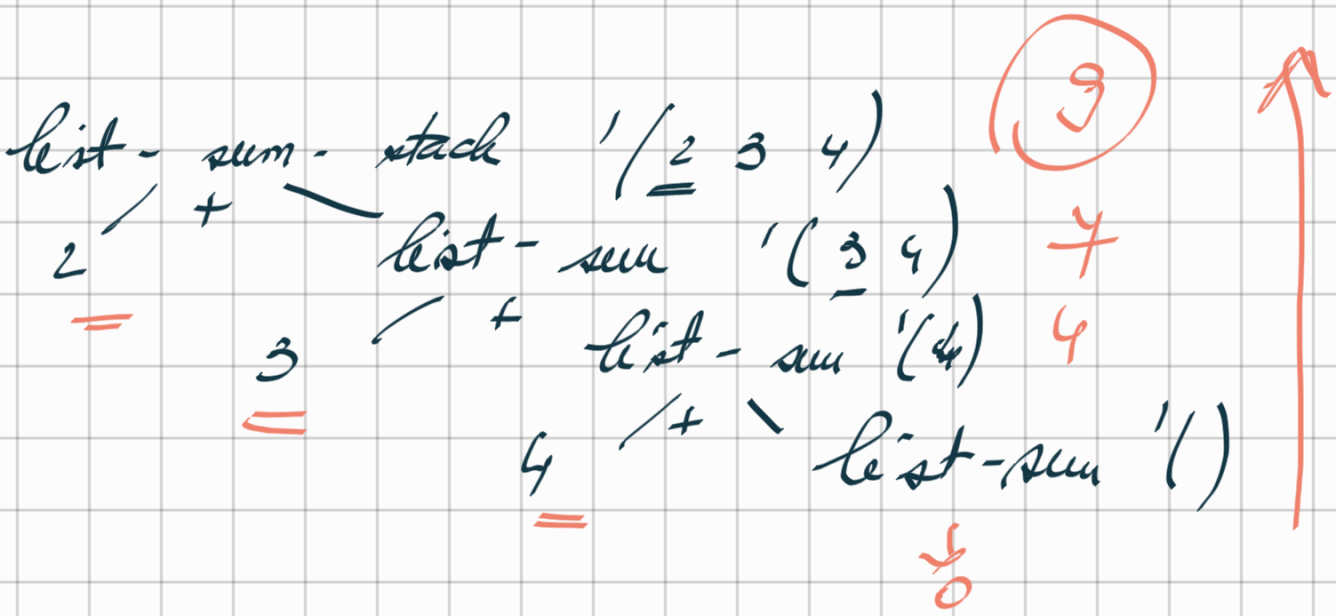
① Rec. pe stiva

→ păstrează informații pe stivă pe parcursul avansului în recursivitate

→ construiește rezultatul la revenirea din recursivitate

Ex :

```
(define (list-sum-stack L)
  (if (null? L)
      0
      (+ (car L) (list-sum-stack (cdr L)))))
```



! Complexitate [temporală: $O(k)$
$k = (\text{length } L)$ [spațială: $O(k)$

sp. pe stivă

=> ineficientă spațial

=> poate genera stack-overflow

② Rec. pe coadă

→ Rezultatul se constr. pe parcursul avansului în recursivitate

TAIL-CALL OPTIMIZATION = apelul recursiv apare ca ultimă operație înainte de return.

→ **acc** → param. în care calc. rezultatul (în general) [acc 0]

Ex:

```
(define (list-sum-tail-helper L acc)
  (if (null? L)
      acc
      (list-sum-tail-helper (cdr L) (+ acc (car L)))))

(define (list-sum-tail L)
  (list-sum-tail-helper L 0))
```

} wrapper

list-sum-tail '(2 3 4) 0

list-sum-tail '(3 4) 2

list-sum-tail '(4) 5

list-sum-tail '() 9

=> 9

! Complexitate [temporală: O(n)
n = length L [spațială: O(1)

=) d.p.d.v spațial, rec. pe coadă e mai „eficientă"

Ex:

my-member x L — # f
f.

my-member x '() = #f

my-member x [x : L] = # t

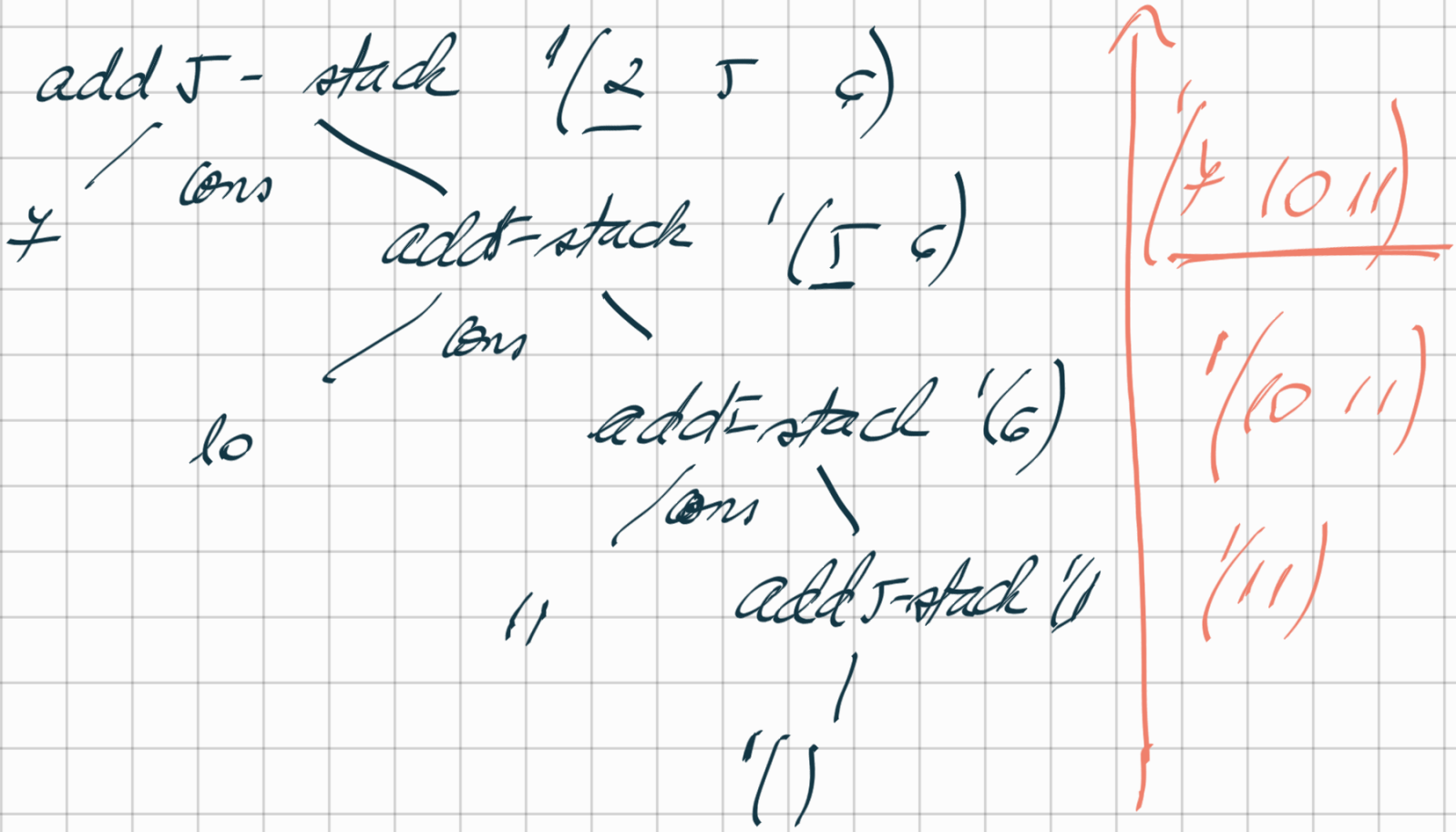
my-member x [y : L] =
my-member x L

! acc. poate lipsi

STIVĂ (vs) COADĂ

1) STIVĂ

```
(define (add5-stack L)
  (if (null? L)
      '()
      (cons (+ 5 (car L)) (add5-stack (cdr L)))))
```



② COADĂ

```
(define (add5-tail L [acc '()])
  (if (null? L)
      acc
      (add5-tail (cdr L) (cons (+ 5 (car L)) acc))))
```

(reverse acc) ①

(append acc (list (+ 5 (car L))))

add5-tail '(2 7 6) '()

|

add-tail '(7 6) '(4)

|

add-tail '(6) '(10 4)

|

add-tail '() '(11 10 4)

↓

'(11 10 4)

① reverse acc

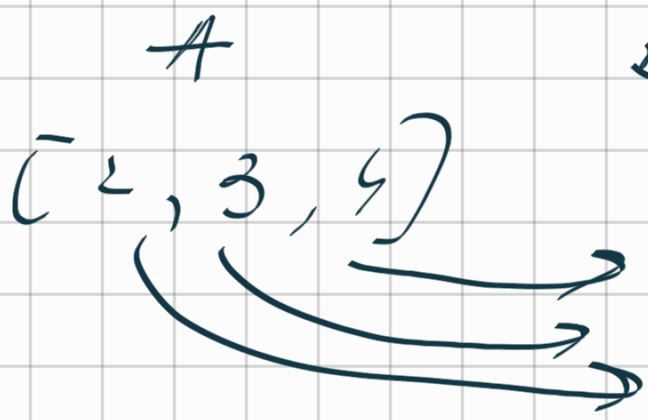
② append

---

① reverse acc

$$O(n) + \underbrace{O(n)}_{\text{reverse}} \Rightarrow O(n)$$

② append



$O(\text{length}(A))$

$O(n) + O + 1 + 2 + \ldots + (n-1)$

$\Rightarrow O(n^2)$

③ Rec. arborescentă

→ cel puțin 2 apeluri recursive independente

fibonacci 0 = 0

fibonacci 1 = 1

fibonacci n = fibonacci(n-1) + fibonacci(n-2)