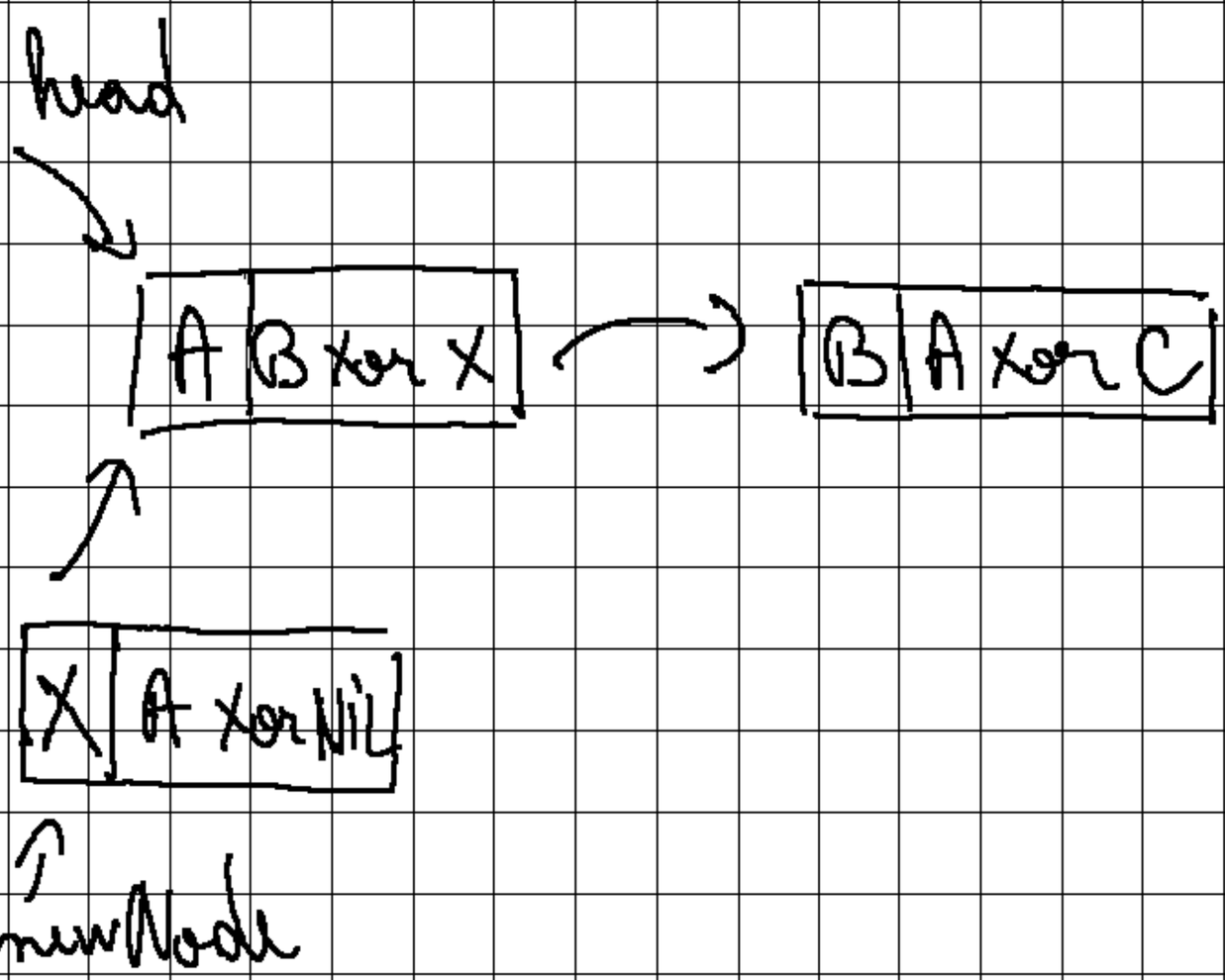
XOR linked list:

link in node B

A xor C xor A = C

link in node C

B xor D xor B = D



Sorted sequence

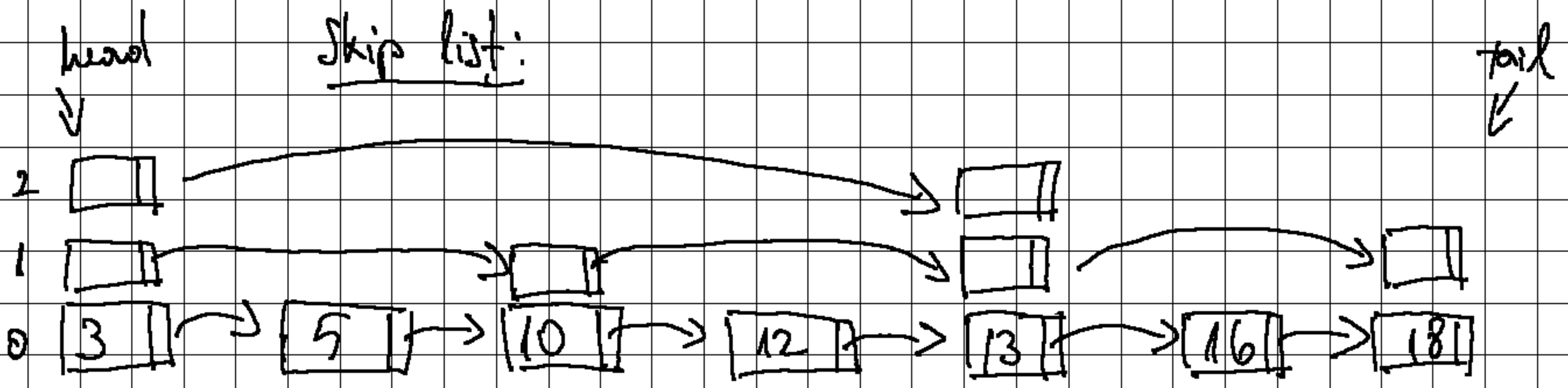
Dynamic array O(n)

(Doubly linked list O(n)

many insertions

	find position	insertion
D.A.	$O(\log_2 n)$ with binary search	$O(n)$
DLL.	$O(n)$	~~O~~ (1)

Skip list:



Level 0: n

level 1: $n/2$

level 2: $n/4$

$\vdots$

1

Linked Lists on arrays:

	1	2	3	4	5	6	7	8	9
	11	6	100	98	38	9	22		
nxt	7	4	5	6	1	-1	2		

head = 3

100 → 38 → 11 → 22 → 6 → 98 → 9

Representation:

elems: TElem []

next: integer []

head: integer

firstEmpty: integer

size: integer

capacity: integer

```
function search (slla, e):
    current ← slla.head
    while slla.elems[current] ≠ e
        and current ≠ -1 execute:
        current ← slla.next[current]
    if current = -1 then
        search ← false
    else
        search ← true
```

O(n)

```
Subalg init(slla):
    slla.capacity ← 10
    slla.elems ← O new array with slla.capacity Telems
    slla.next ← ——— 11 ——— integers
    slla.head ← -1
    slla.size ← 0
    for i←1, slla.capacity-1 ex
        slla.next[i] ← i+1
    end
    slla.next[slla.cap] ← -1
    slla.firstEmpty ← 1
end
```

```
Subalg insertPos (slla, pos, e):
    if p < 1 or p > slla.size + 1 then
        e throw exception

    insertionPos ← slla.firstEmpty
    if slla.firstEmpty = -1 then    // resize
        slla.cap ← slla.cap * 2
        newElems ← () new array with slla.cap TElems
        newNext ← () ——— " ——— integers
        for i ← 1, slla.size ex
            newElems[i] ← slla.elems[i]
            newNext[i] ← slla.next[i]
        D deallocate slla.elems and slla.next
        slla.elems ← newElems
        slla.next ← newNext
        slla.firstEmpty ← slla.size + 1
        for i ← slla.size+1, slla.cap - 1 ex
            slla.next[i] ← i+1
        slla.next[slla.cap] ← -1
```

insertionPos ← slla.firstEmpty
slla.firstEmpty ← slla.next[slla.firstEmpty]
slla.elems[insertionPos] ← e
slla.next[insertionPos] ← -1
if slla.head = -1
    slla.head ← insertionPos
else
    currentNode ← slla.head
    currentP ← 1
    while currentP < p-1 ex
        currentNode ← slla.next[currentNode]
        currentP ← currentP + 1
    slla.next[insertionPos] ← slla.next[currentNode]
    slla.next[currentNode] ← insertionPos