- Urutin nilai kartunya

Jumlah semua → kalikan

max pref sum * ...

Max

max(1 - 2 * ...)

Pref sum [l - .. r] * ...

(-)

(+)

(-)

(+)

(-)

(-)

1 5 g 7

Pref sum = (1+5+g) = 15

15 x 7 = 105

(1+5+7) * g = 13 * g

= =

5

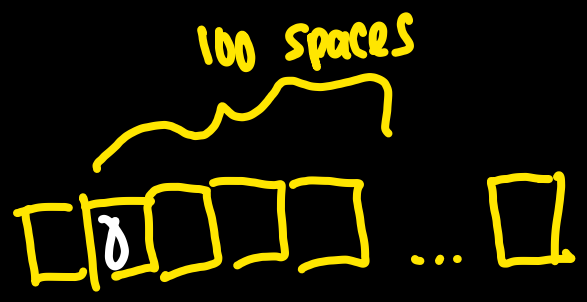
-3 -2 -4 -1 5

(-3 -2 -1) * -4 + 5


Tipe data
Primitif : int, string, ...
non primitif :
struktur data
Linier
non linier
Linier
static array
Dynamic array
stack
Queue
PQ
DQ
Linked list
set
multiset
Vector
stack
Queue - PQ - DQ
static array
map - hash table / Dictionary
Pair / Tuple
kapasitas → 100 GB
int arr[100 GB] =
arr[1] = 0
int x = 5 → MLE

* Array

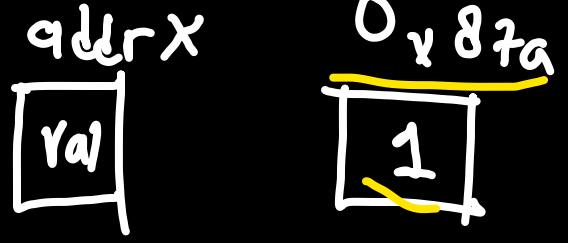
int arr[100] →



arr[1] = 0

Functional Programming → Pointer

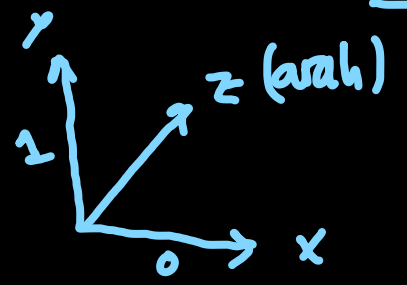
Pointer int x = 1 →



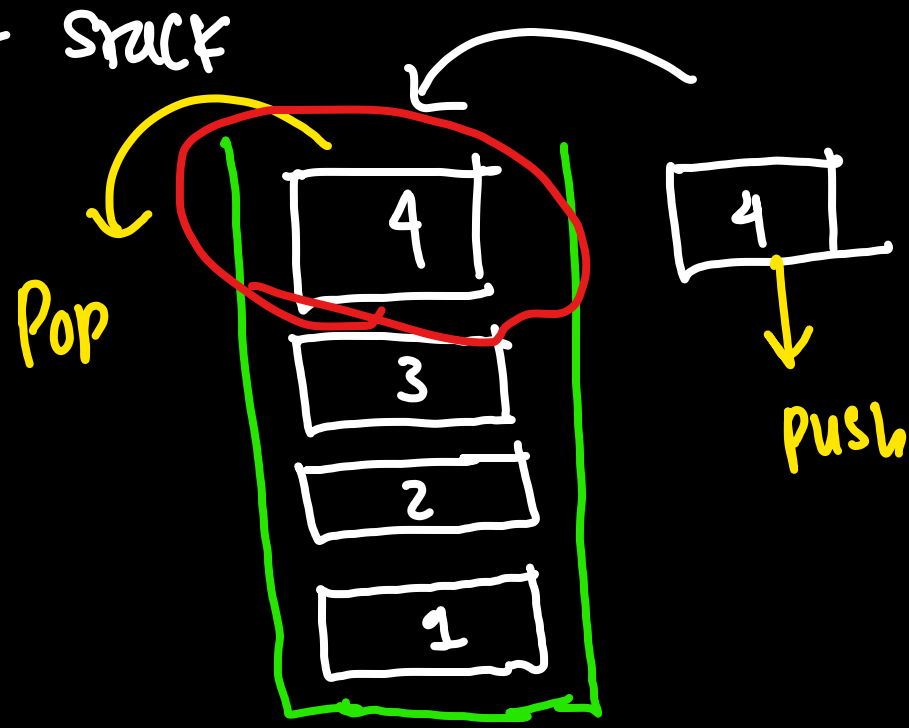
* static Vs Dynamic
  - Malloc
  - Dynamic < Operable action } push, pop, insert, ...
    → tahap / langkah komputasional sistematis

Vector < static
         Q / PQ
         static array

vector



* Stack



LIFO (Last in - First out)

* DFS (Depth First Search)
* certain algorithms

array : [1, 1, 2, 3]

First in

Last in

stack : [3, 2, 1, 1]

First out

Last out

* Queue (General)

FIFO (First in - First out)

belakang → depan

depan → ~~belakang~~

~~Sembarang Tempat~~

orang Pertama masuk = Paling depan

orang terakhir masuk = Paling belakang

* BFs (Breadth - First - Search)

array : [1, 1, 2, 3] ⟶ Q : [1, 1, 2, 3]

$1 \le n \le 10^{23}$ $n = \ldots$

$1 \le n \le 10^{23}$, $n \neq 15$ → edge case

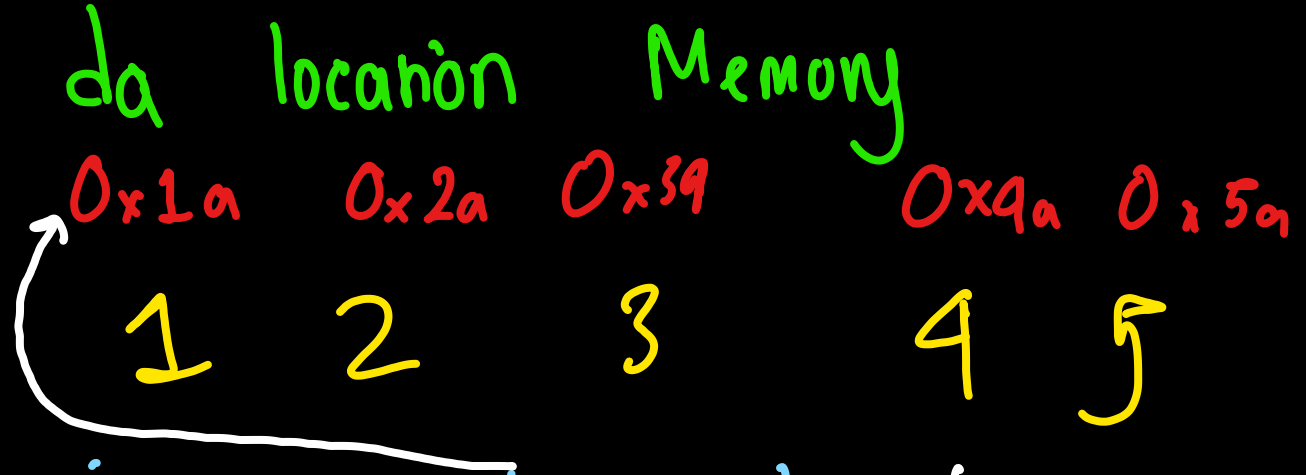
* Priority Queue → top : max

arr : [1, 5, 3, 2, 0] → PQ → PQ : [0, 1, 2, 3, 5] (min-heap)

PQ : [5, 3, 2, 1, 0] (max-heap)

* Binary-heap (Tree)
  * Graph → • Shortest Path
    • Min cost
  * Tree

* Dynamic Queue



iterator it = dq.begin()

Tempatkan di idx ke-2

it++ = Sebelah kanan 0x1a

it+2 = 2 lokasi sebelah kanan 0x1a

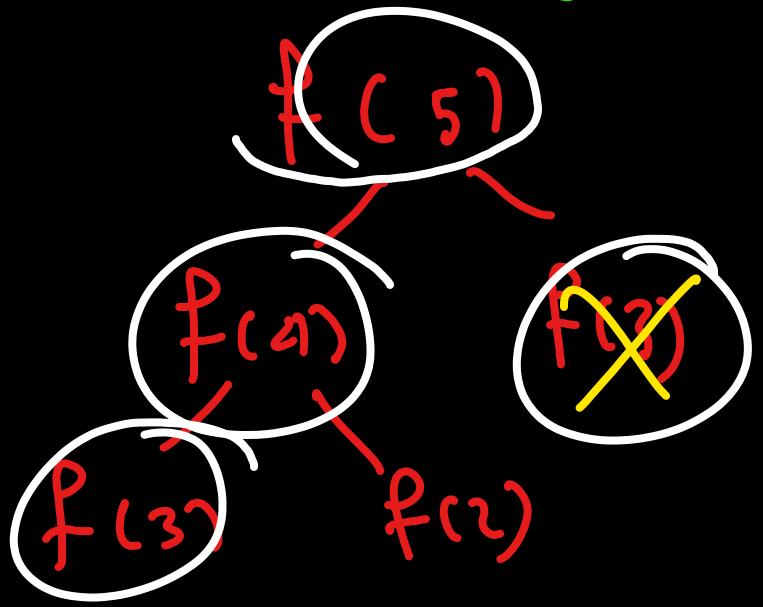
* Set (himpunan)

↳ gak boleh ada duplikat

arr = { 1, 1, 2, 3 } —set→ set = { 1, 2, 3 }

↳ subset → gaboleh berulang

* pruning tree



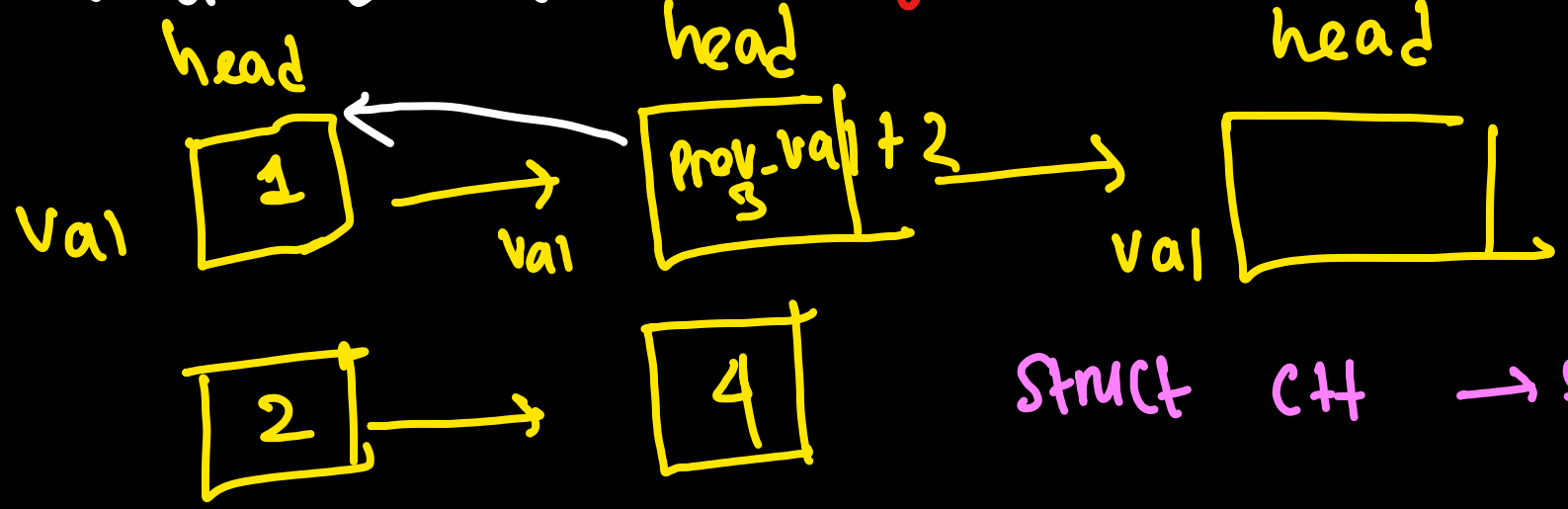
= { 5, 4, 3 }

* Multi Set

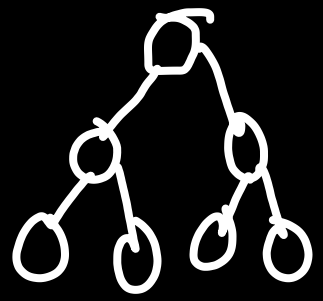
Set (set<>) = { {1,2}, {1,2} }

multiset = { {1,2} }

* Linked list → binary tree, block-chain



struct ctt → semi class (oop)



Memo [1000001] array.

N = 10 → 10

memo [1] = 1
memo [2] = 1
memo [3] = 2
} 3 space located

* map → hash table

arr[2] = "C"

hash table → map (dictionary)

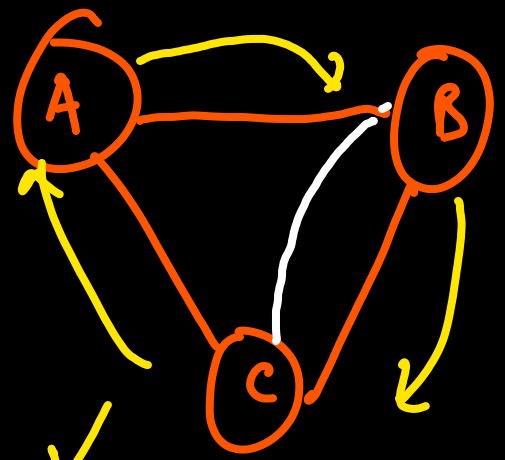
Map <string, int> umur;

Key	Value
Bintang	16
Abdan	18
Revan	15

umur["Bintang"] = 16

umur["Abdan"] = 18

+ enumerate Key, Value



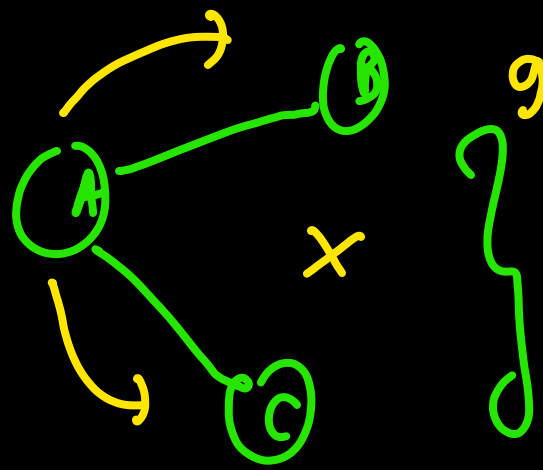
graf

cycle

adj[a] = {B}

adj[B] = {a}

adj[C] = {a, B}



x

gaada Cycle

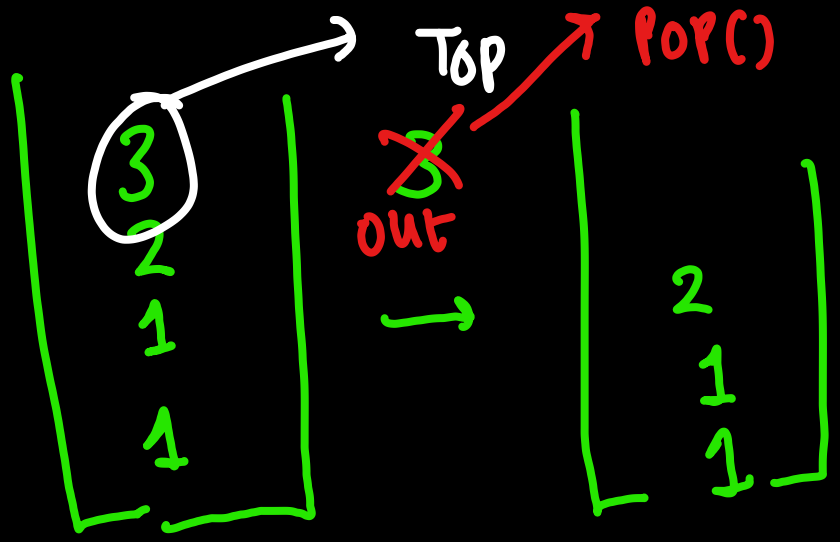
Tree

Can = x → search

Can ≤ x → lower_bound

Can ≥ x → upper_bound

1, 1, 2, 3 →

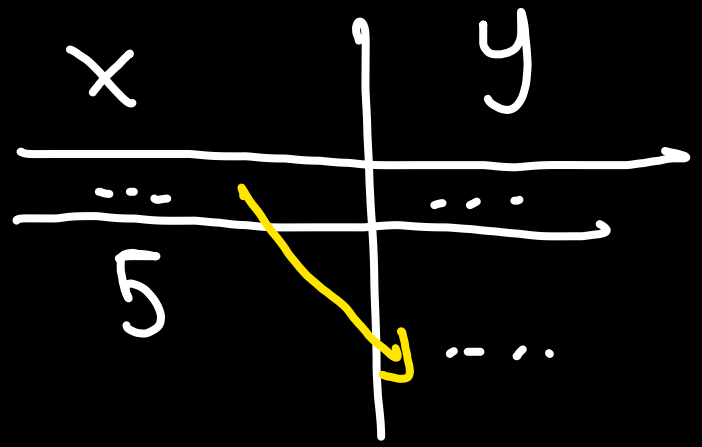



```
int x = 5
int y = x
```

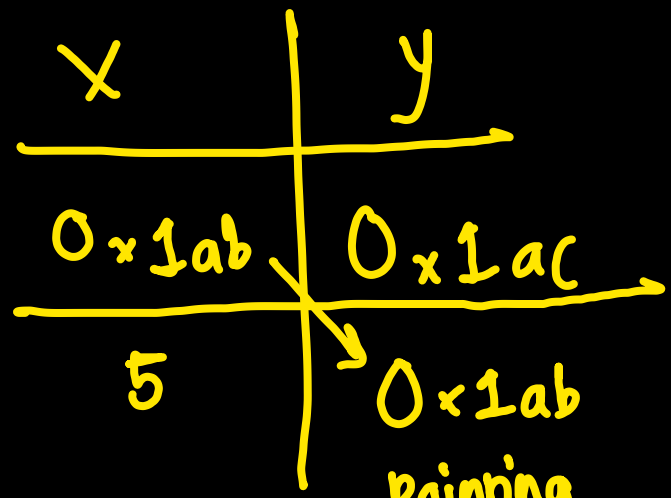
	x	y
addr	0x1ab	0x1ac
val	5	5

→ Pass by value

Pass by Reference → y bembah, x bembah → y navigator x, y refering to x



Val:

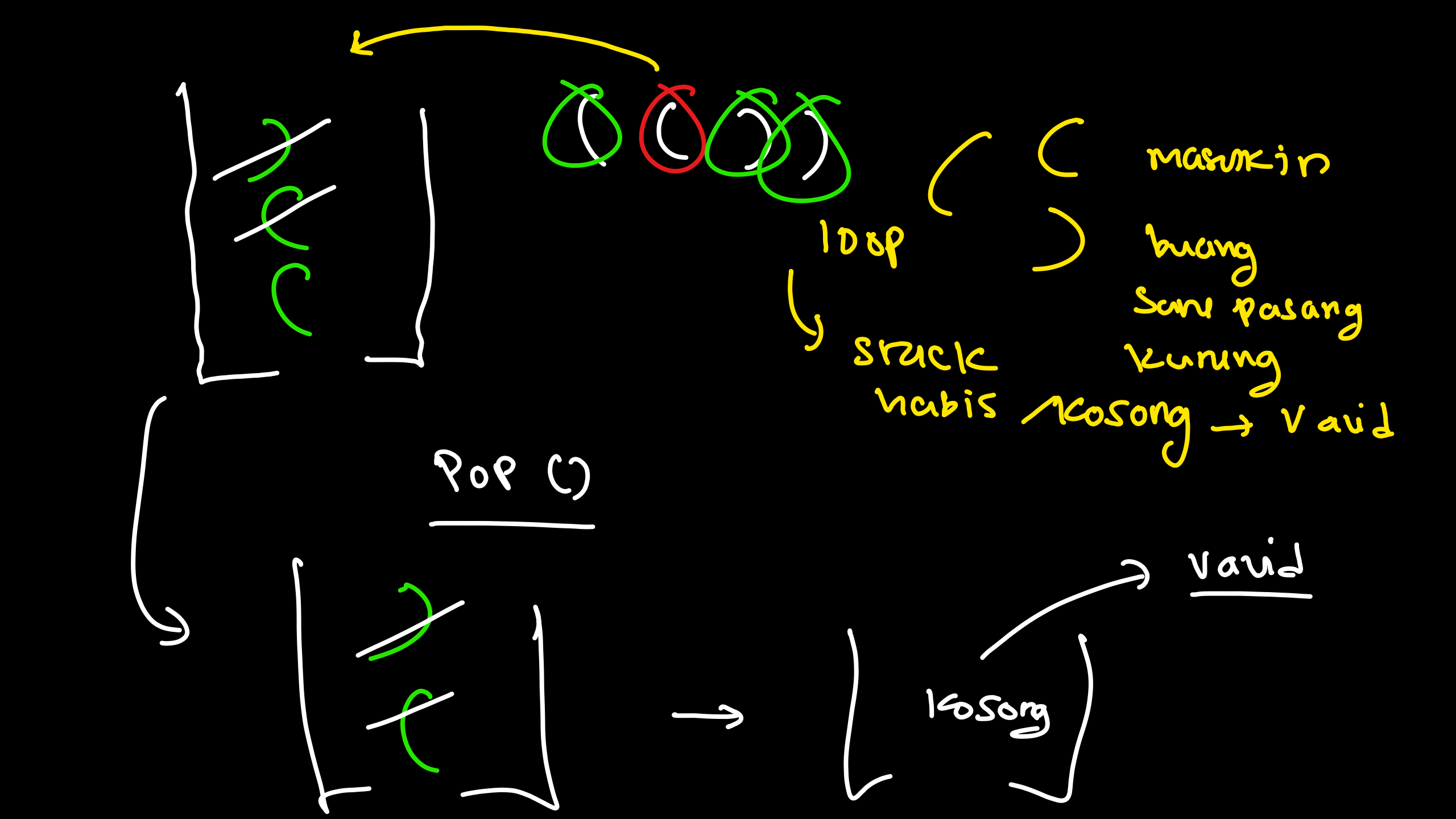


Poinning

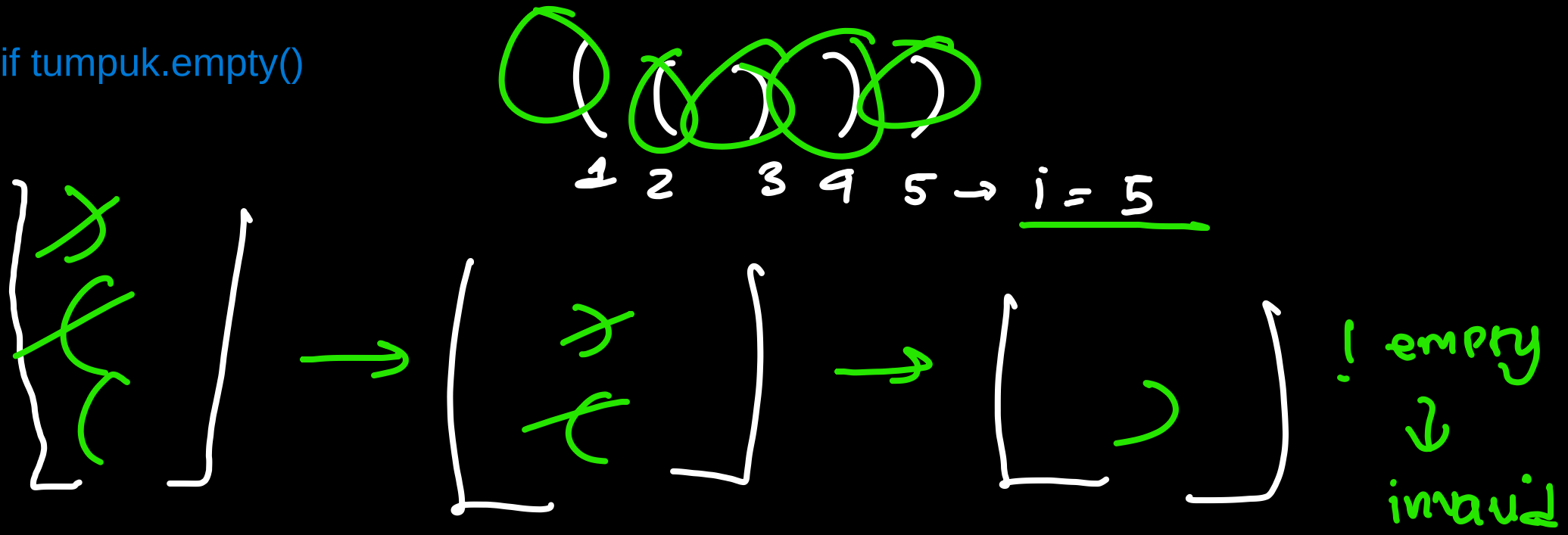
Parenthesis → Tanda kurung

(( )) → Valid ?

(( ))) → Tidak Valid

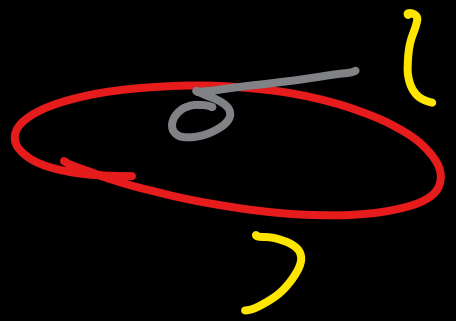

masukin
loop
kurung
buang
sama pasang
stack
habis
kosong → valid
Pop ()
valid
kosong

if tumpuk.empty()



If stack empty and iterasi = size str

Valid



( )

awal

-) → invalid
- jumlah buka != jumlah tutup
- ( )) (→ invalid

akhirnya

- ( )) ((( ) cek pake stack

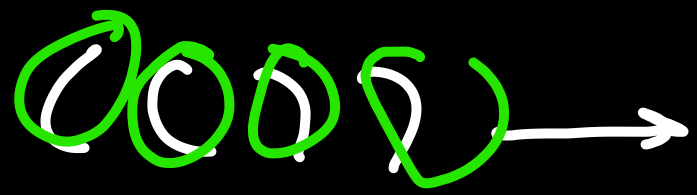
SPPPSSSP

int x

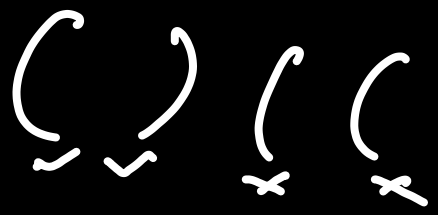
const int x

Var , Const

Global , Local Scope



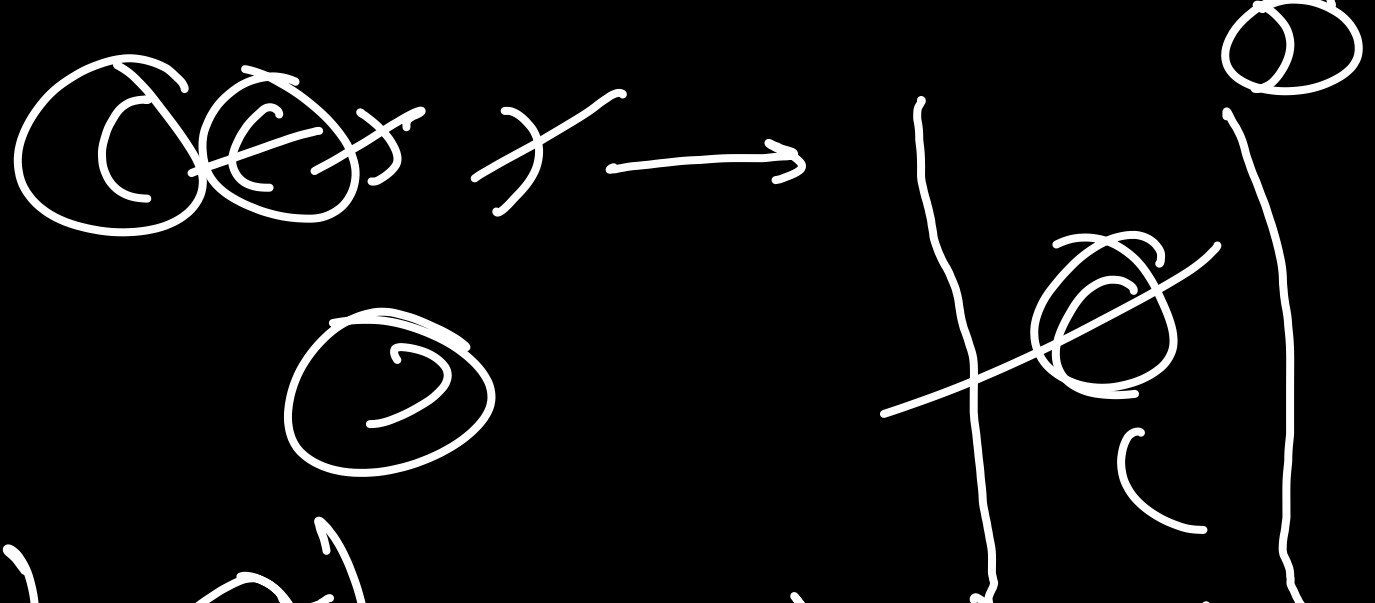
→ Valid Parenthesis



→ invalid Parenthesis

S = ...)( )( )(...

S Valid Parenthesis ?

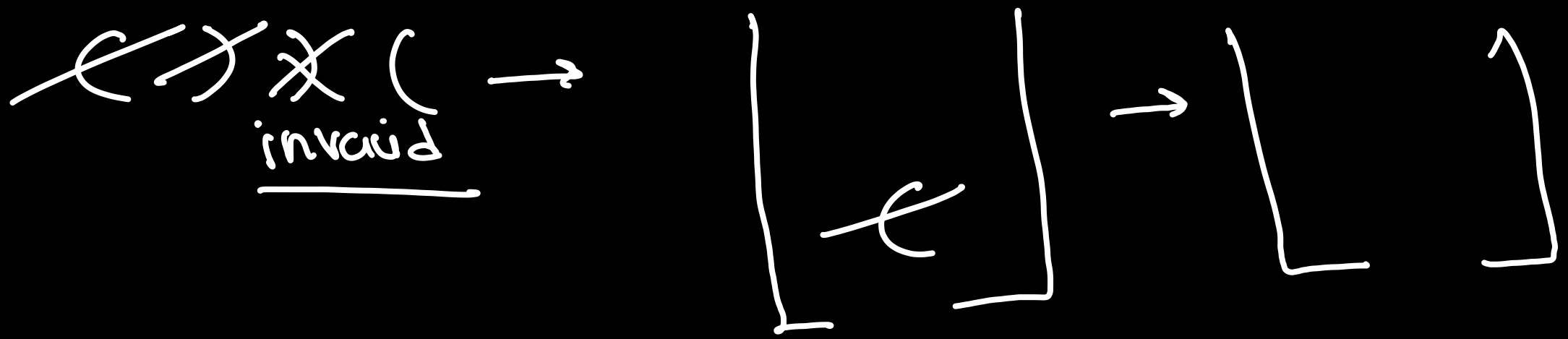


"(" masukkan ke dalam stack

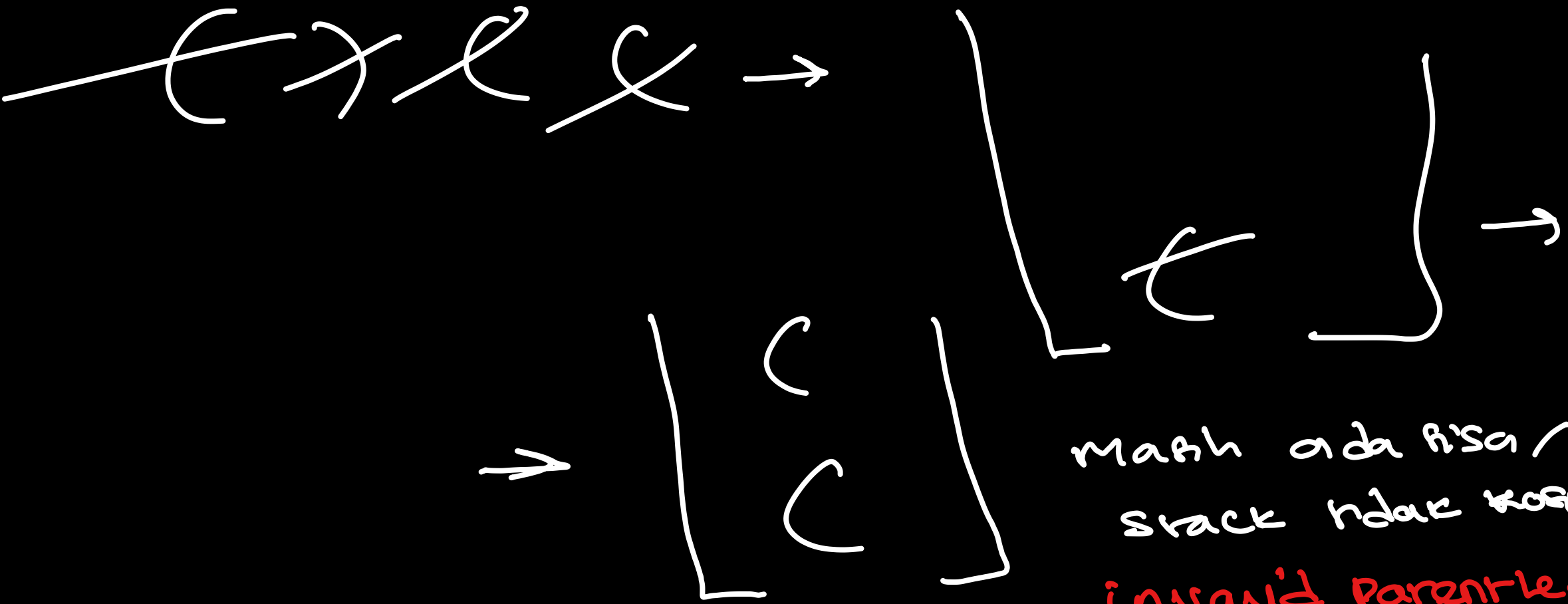
")" ambil semua stack dan hitung

kalau ada

stacknya kosong



invalid



masih ada isi / stack tidak kosong

invalid parentheses