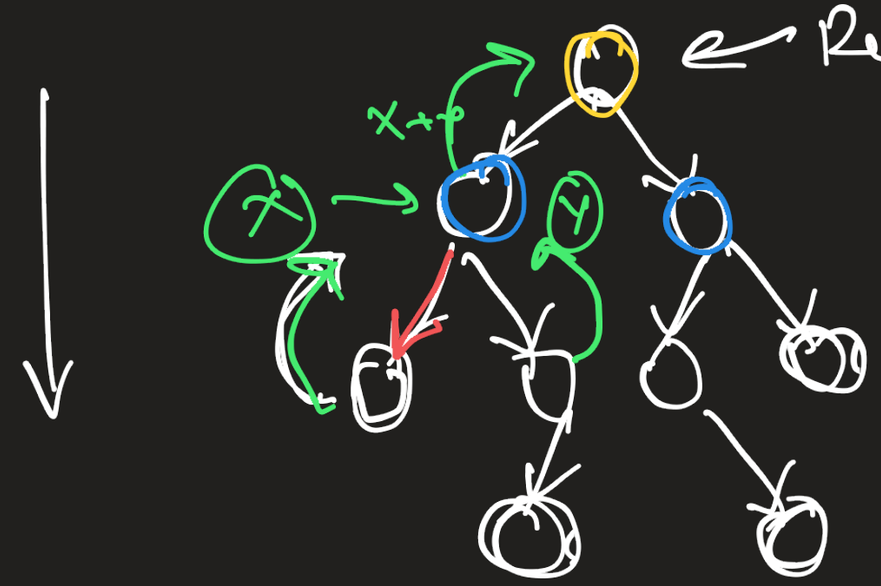
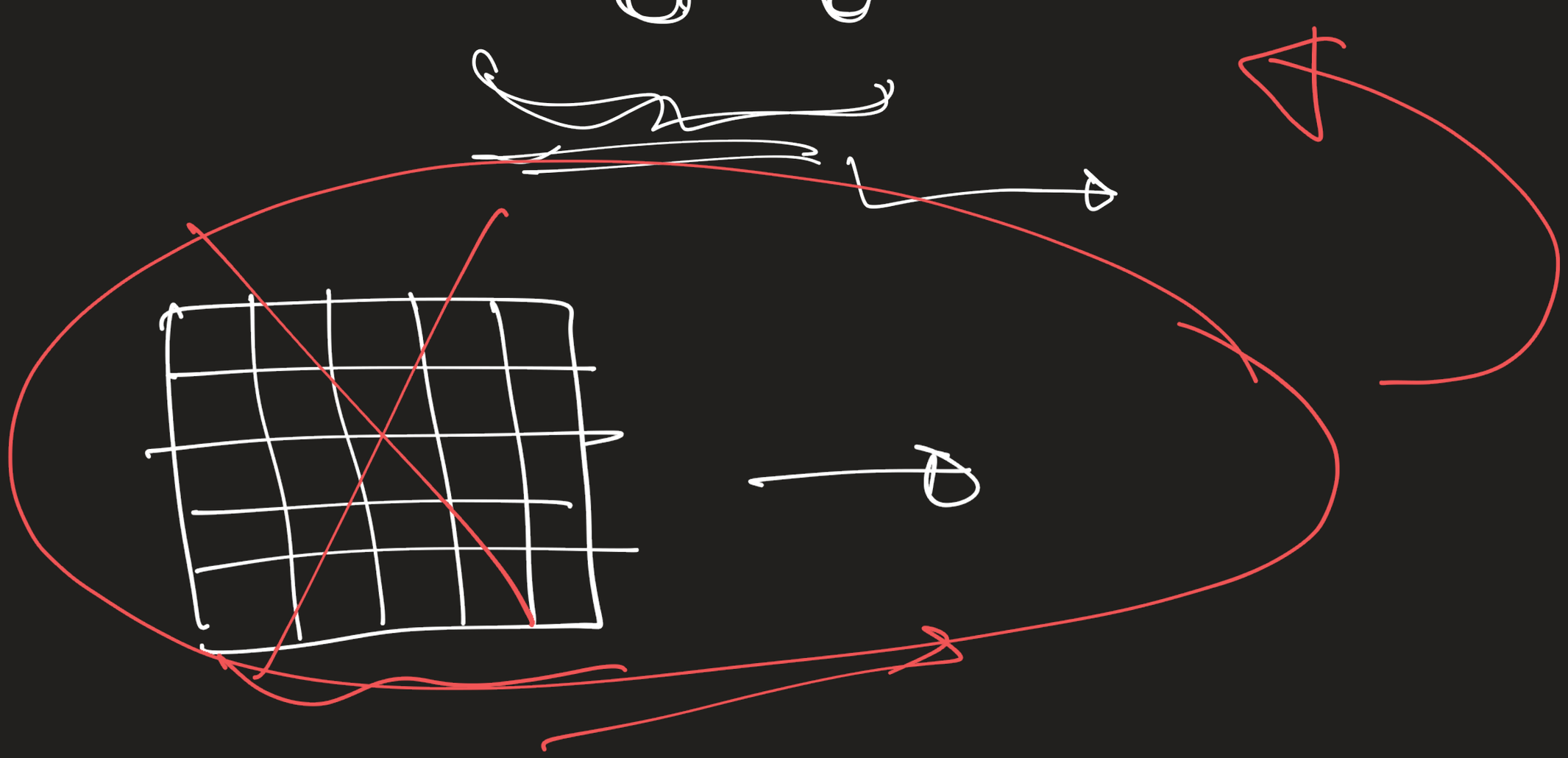
Recursion :-



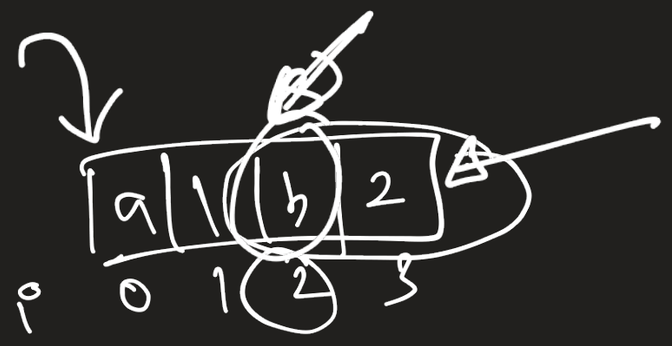
→ code w/ print

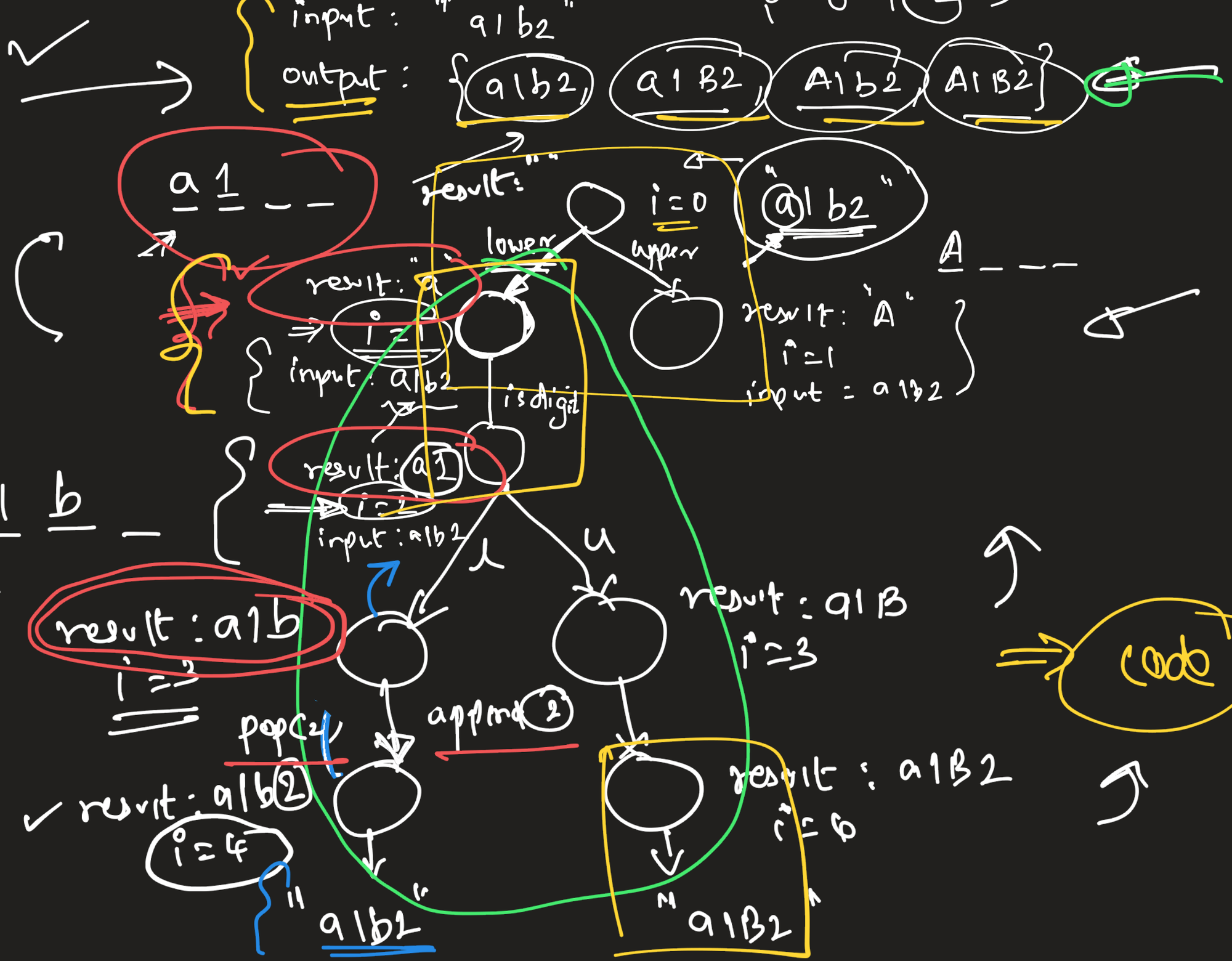


① Letter case permutation:

input: "a1b2"

output: {a1b2, a1B2, A1b2, A1B2}





S [N]

for (i : 0 .. N) ← O(1)

S = S + "$"; ← O(1)

O(N) ← join PT

→ O(N) ←

→ O(N²) ←

```
letpermute ( string input) {
    return helper ( input, 0, "");
}

void helper ( string input, int i, string result) {
    if ( i == input.len) {
        print (result);
        return;
    }
    if ( isdigit ( input [i]))
        helper ( input, i+1, result + input[i])
    else {
        left ← helper (input, i+1, result + toLower( input[i]))
        right ← helper ( input, i+1, result + toUpper ( input[i])
    }
}
```

result

results

vector<string> results){

results . append (result)

back

result: "", i = 0, input: "9b2"

i = 0

2 lower

1

3 upper 4

result: a

i = 1

result = A

tmp = result + toLower(input[i])

1

helper(input, i+1, result + toLower(input[i]))

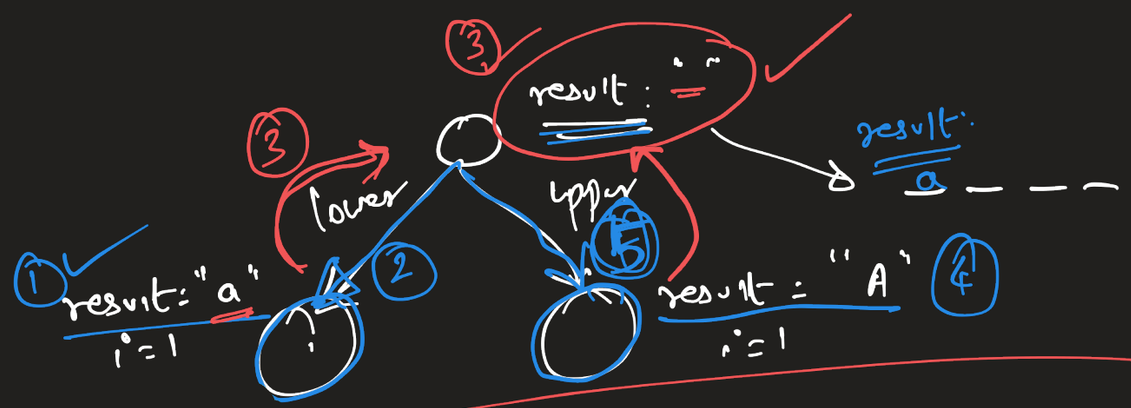
2

helper(input, i+1, result + toUpper(input[i]))

3

4 =>

immutable




```
result.append(toLower(input[i]))
helper(input, i+1, result)
result.pop()
result.append(toUpper(input[i]))
helper(input, i+1, result)
result.pop()
```

left

right

mutable

result[i].toLower()

helper

result[i].toUpper()

helper

Rec: / letter case permutation

T: $O(b^h)$ → no. of nodes : $O(2^n)$

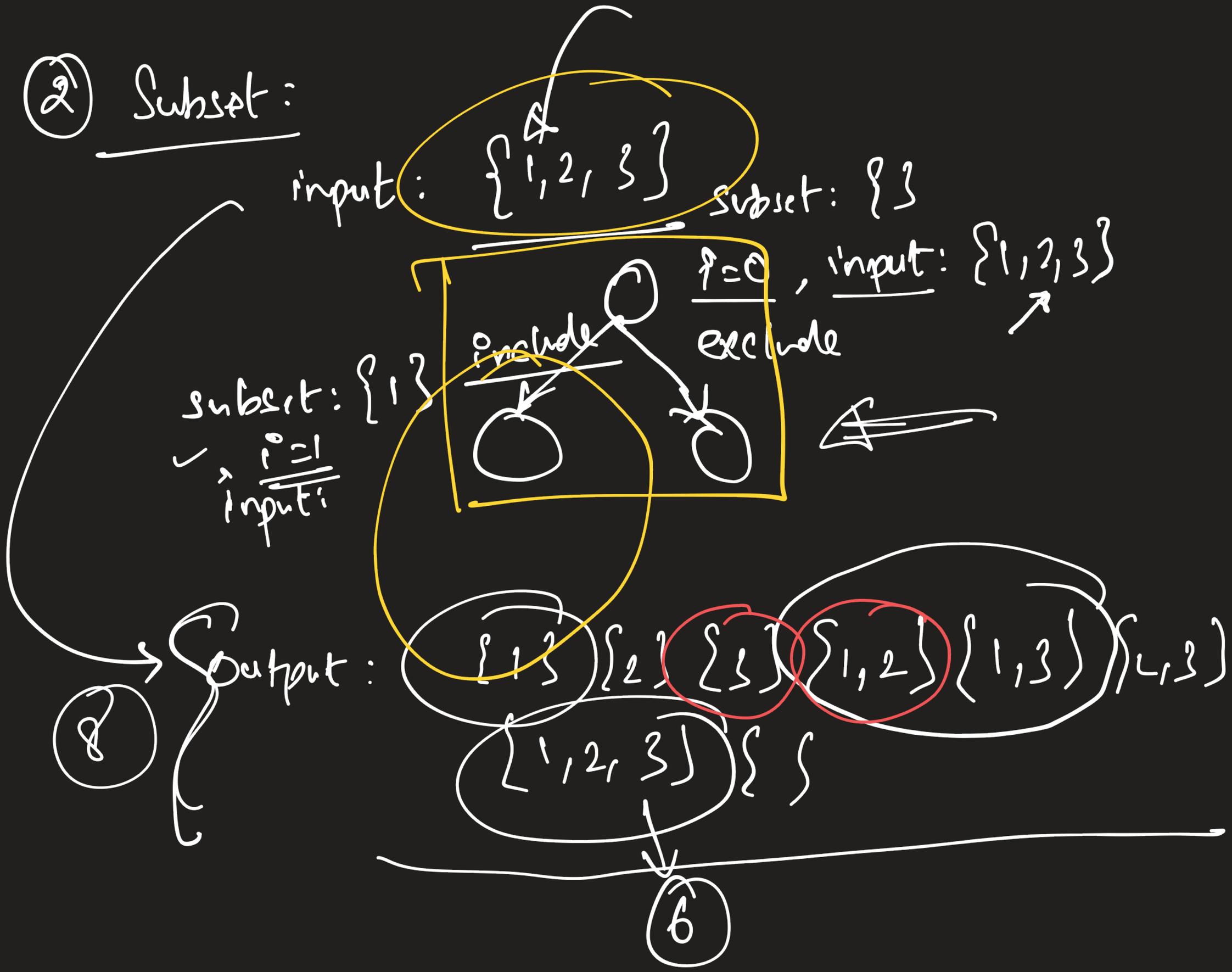
S: $O(h)$ = $O(n) + O(n)$

↳ stack space

↓ stack space

↓ result

② Subset :


input : {1, 2, 3}
subset : {}
i=0 , input : {1, 2, 3}
include
exclude
subset : {1}
i=1
input : {1, 2, 3}
output :
{1}
{2}
{3}
{1, 2}
{1, 3}
{2, 3}
{1, 2, 3}
{}
8
6

input : {1,2,3}

Subset : { }
i := 0

Subset : {1}
i := 1

{ }
i := 1

Subset : {1,2}
i := 2

{3}
i := 2

{1,2,3}

i := 3

halting condition

{1,2,3}

{1,2}

{1,3}

{1}

i, e, -

1, 3

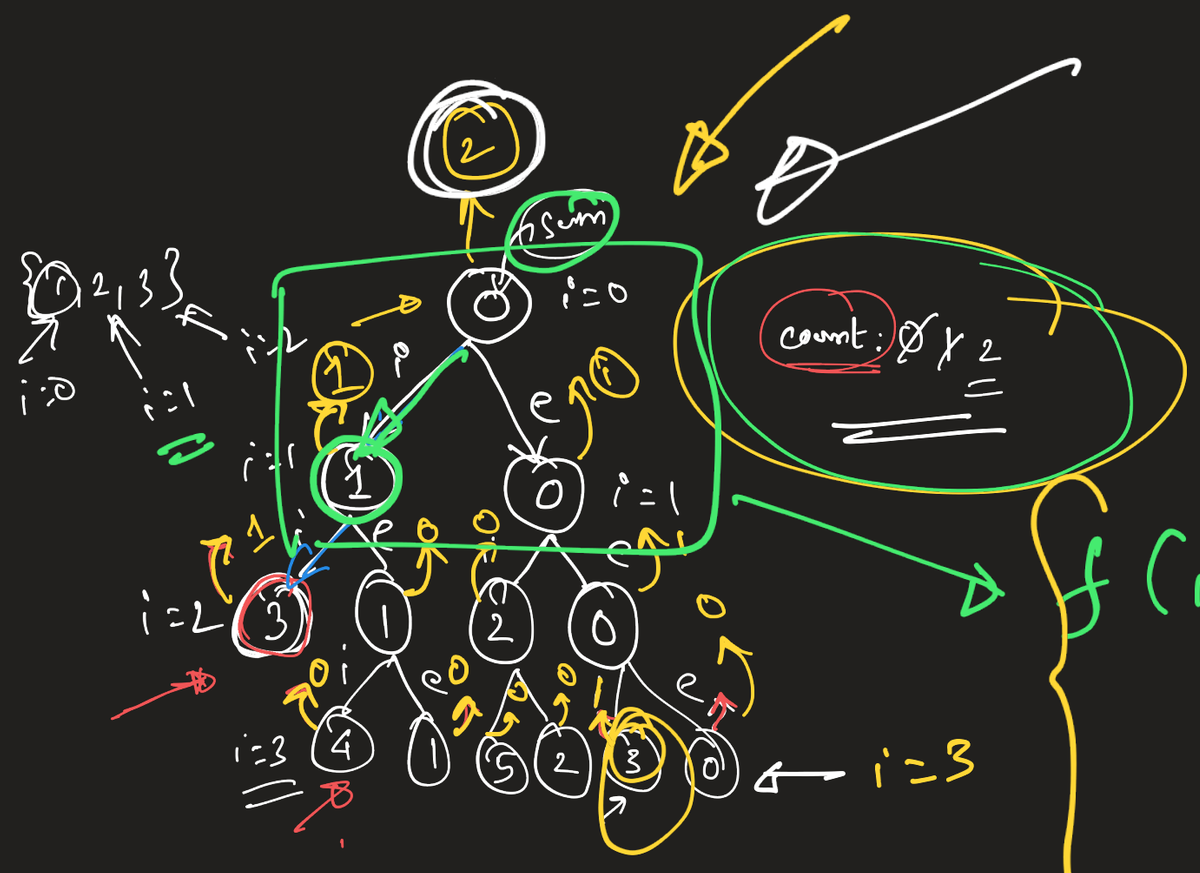
2

(3) Subset Sum :- ((Counting Subsets))

input : {1, 2, 3}

target_Sum : 3

result : 2 → {1, 2} {3}



f(i, sum) =

f(i+1, sum + input[i])

+

f(i+1, sum)

base:

```
if i == input.len : no-op
if sum == targetSum : count++
if sum > targetSum : no-op
```

2

1

2

return 0

return 1

return 0

```
int countSubset(vector<int> input, int targetSum) {
    return helper(input, targetSum, 0, 0);
}

int helper(vector<int> input, int targetSum, int i, int sum) {
    if (sum == targetSum) return 1;
    if (sum > targetSum || i == input.len) return 0;
    return helper(input, targetSum, i+1, sum + input[i])
         + helper(input, targetSum, i+1, sum);
}
```

base

base

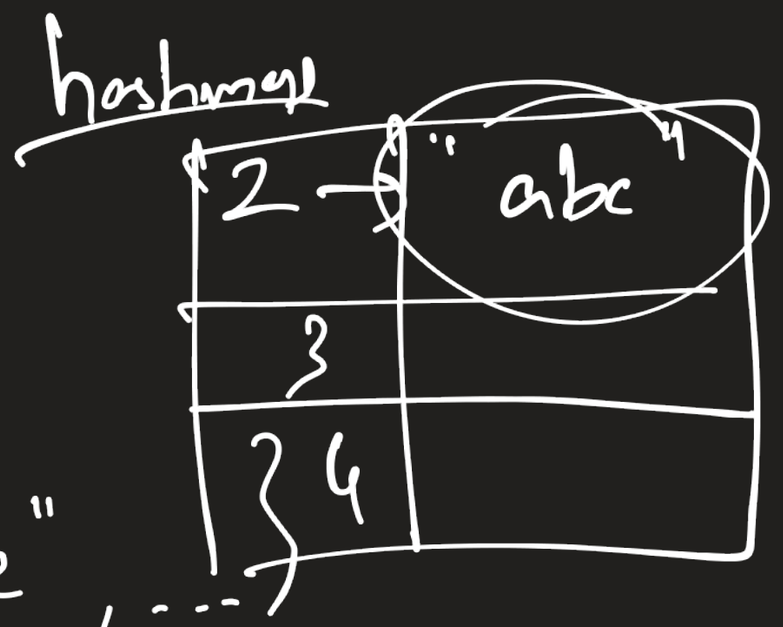
backtrack

sum

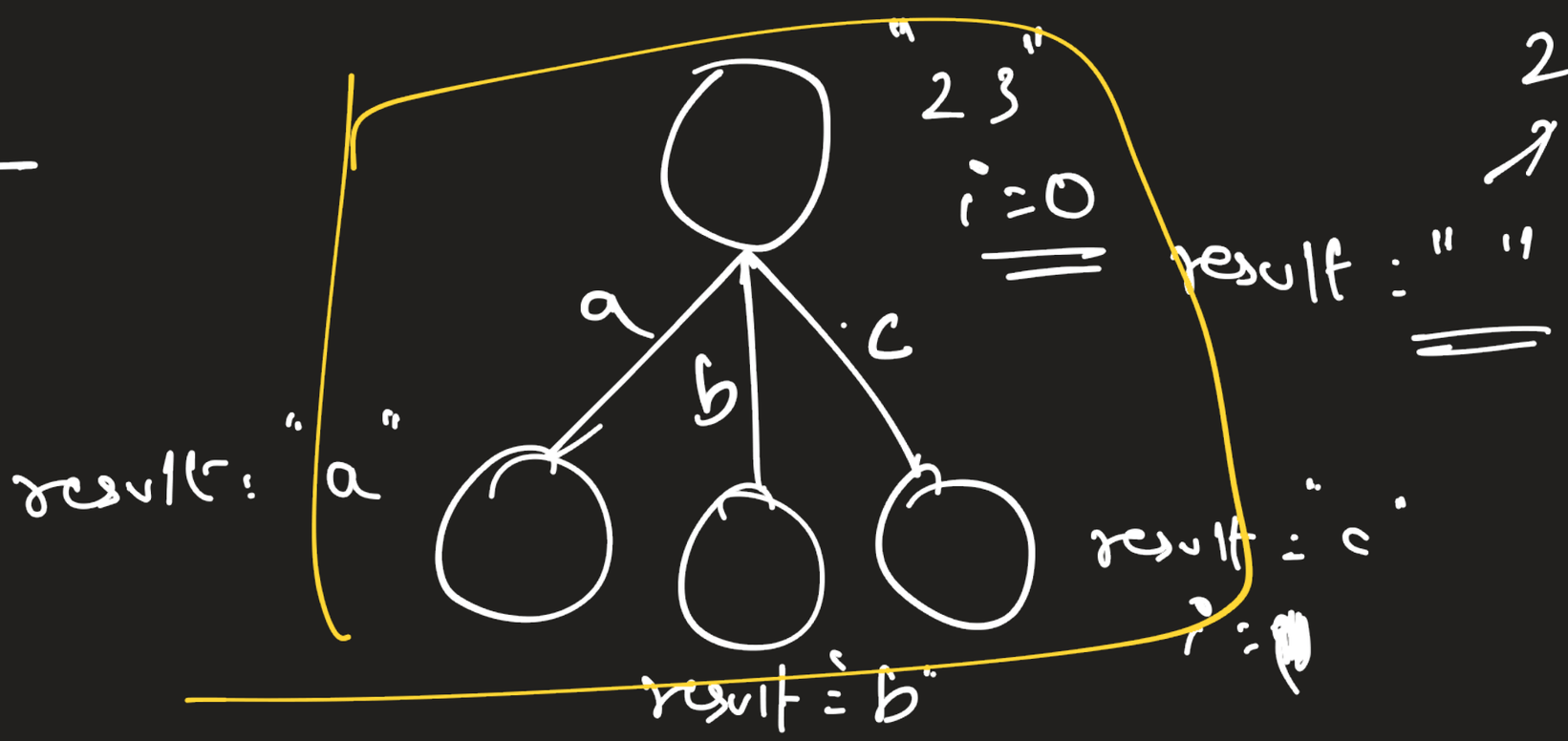
(4) Dial pad

input = "23"

output = {"ad", "ae", ...}



c
b
a
— —



5 Generate parenthesis

$n = 3$ pairs

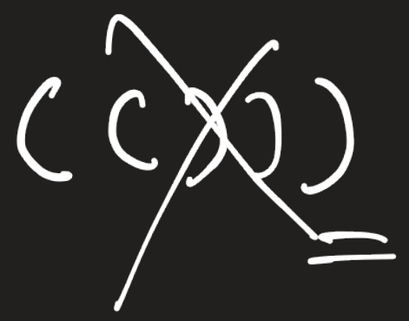
✓ ((()))

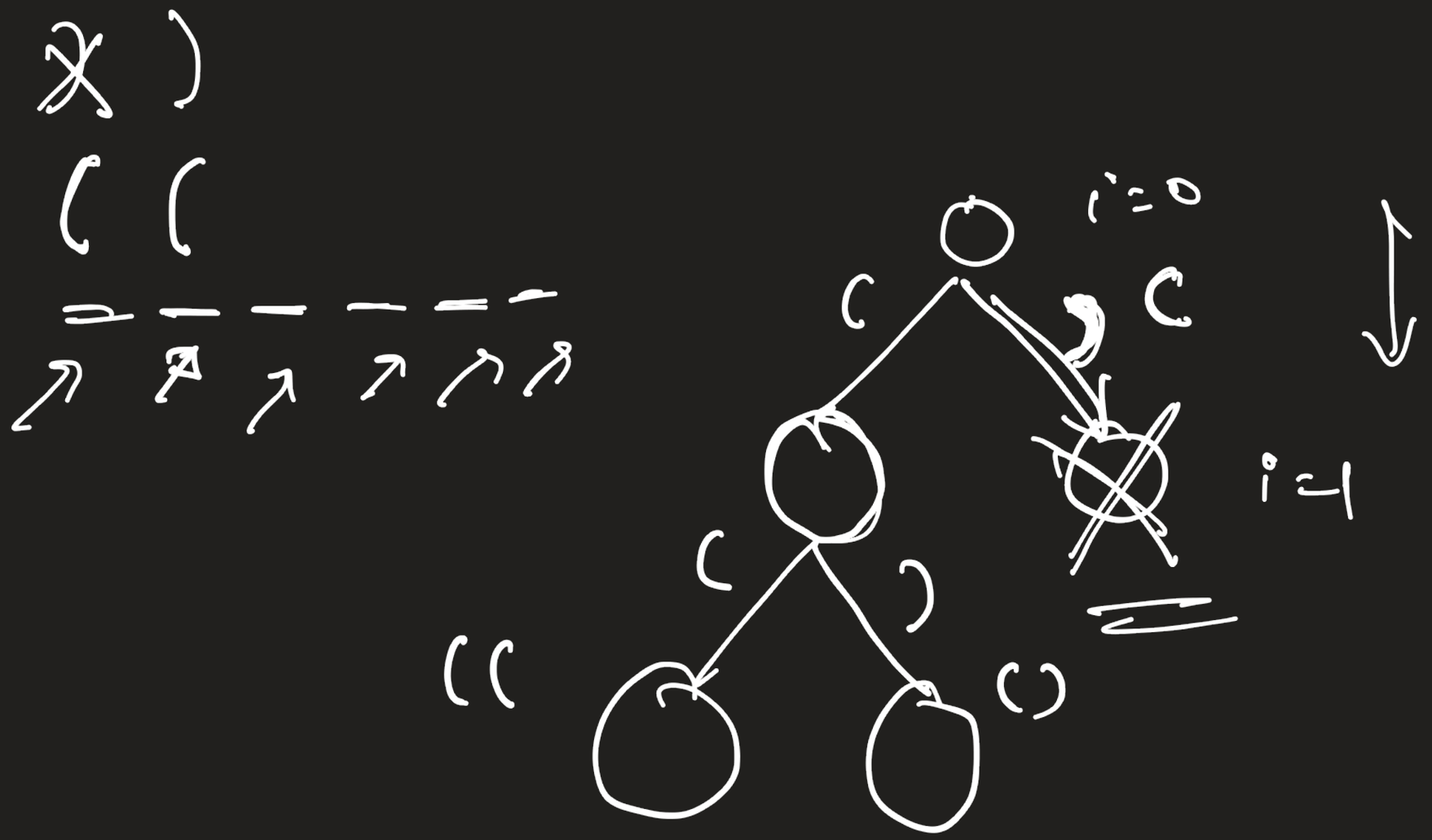
✓ ()()()

✓ (()())

✓ ()(())

⋮

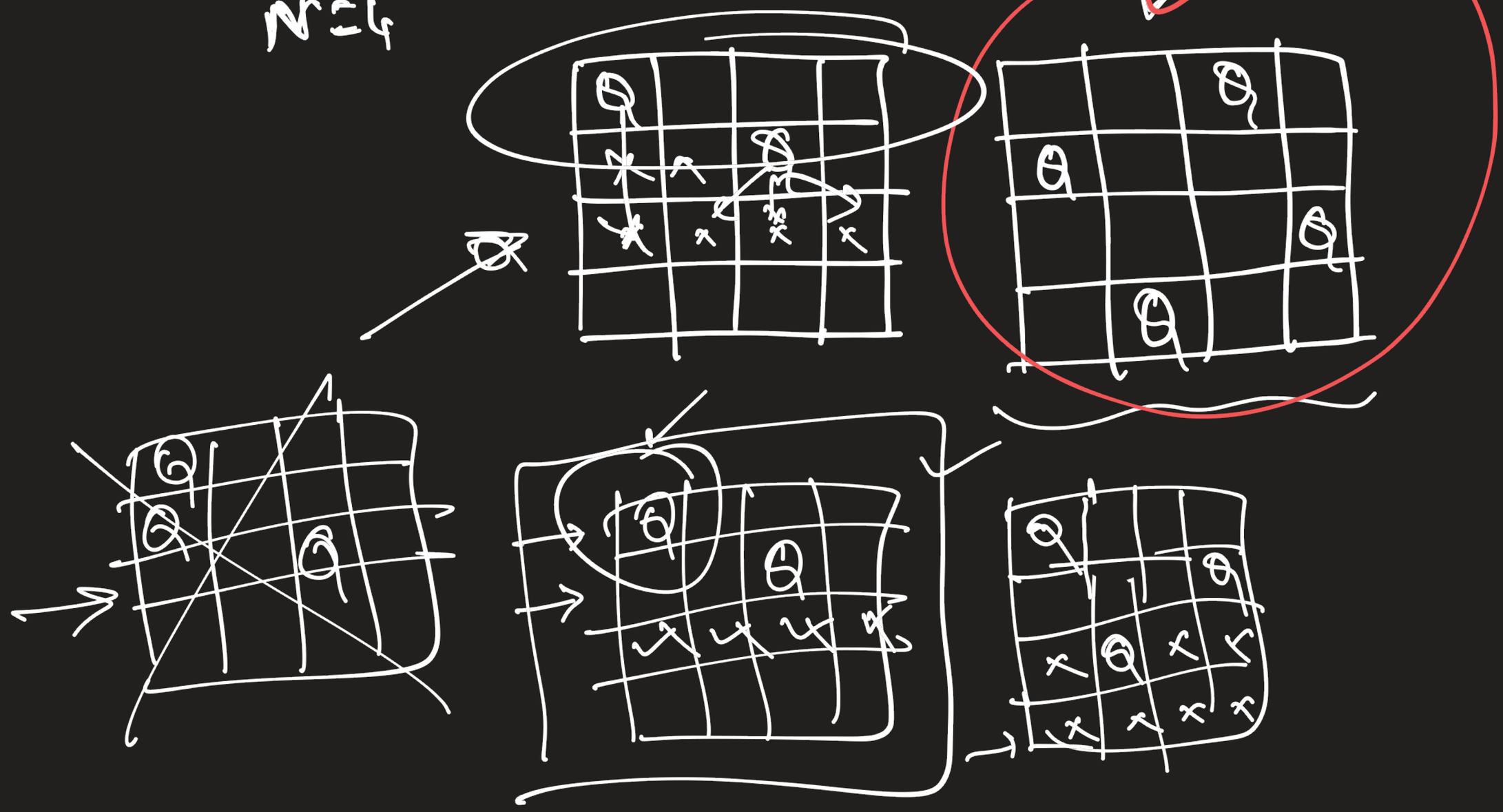



i=0
i=1

6 N-Queens: ✓

chess board : N × N

N = 4



hasConflict(board, i, j)

O(N²) → O(N)

board:

i = 0

i = 1

i = 2

(N-1)

N

hasConflict

base

i == N

print (board)

return

i = 3

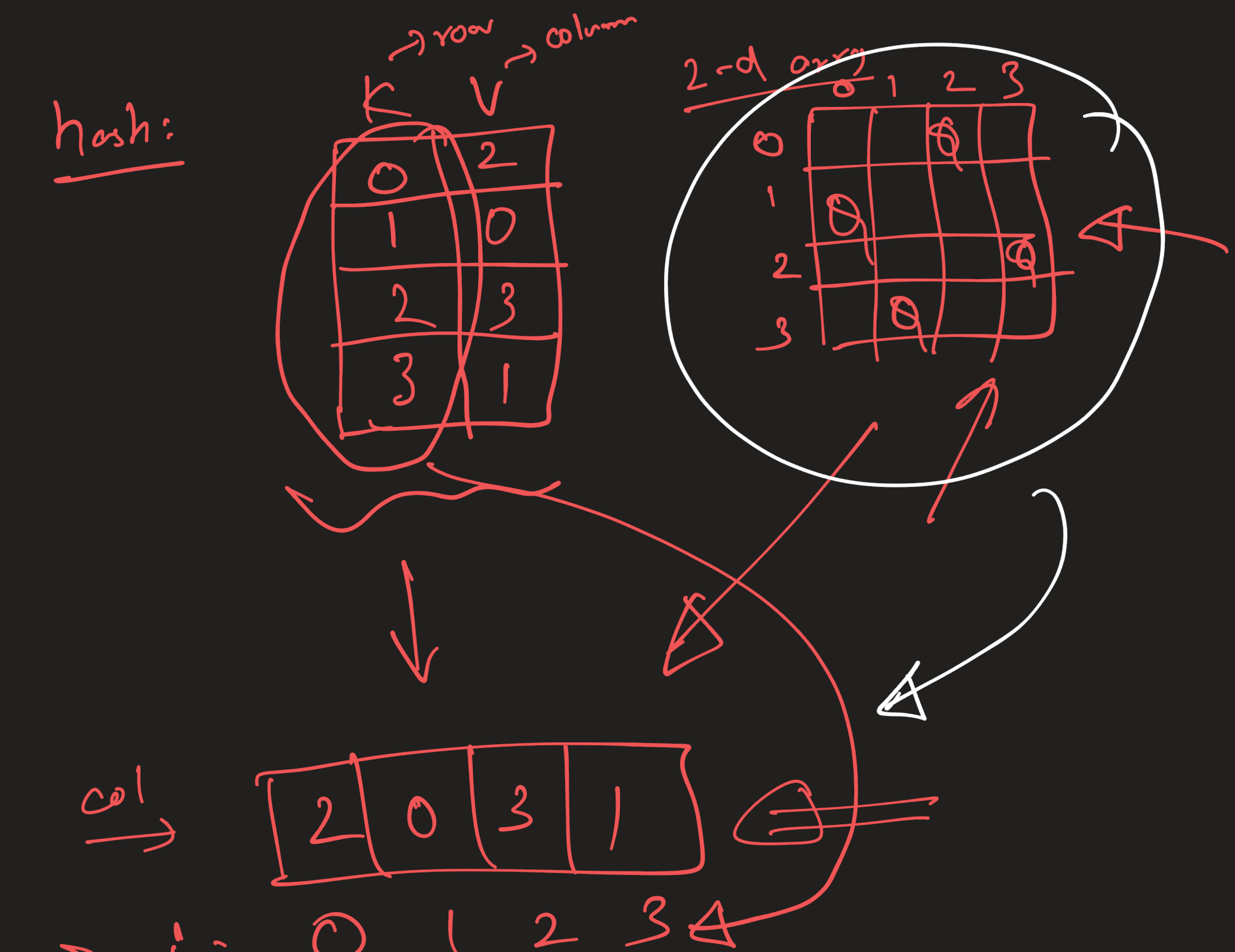
(N-2)

i = 4

```
void NQ(int N) {
    int board[N];
    helper(N, board, 0);
}

void helper(int N, int[] board, int i) {
    if (i == N) {
        print(board);
        return;
    }
    for (int j = 0; j < N; j++) {
        if (!hasConflict(board, i, j)) {
            board[i] = j;
            helper(N, board, i+1);
        }
    }
}
```

hash:



T: $O(b^h)$. $O(N^N)$ → $O(N!)$

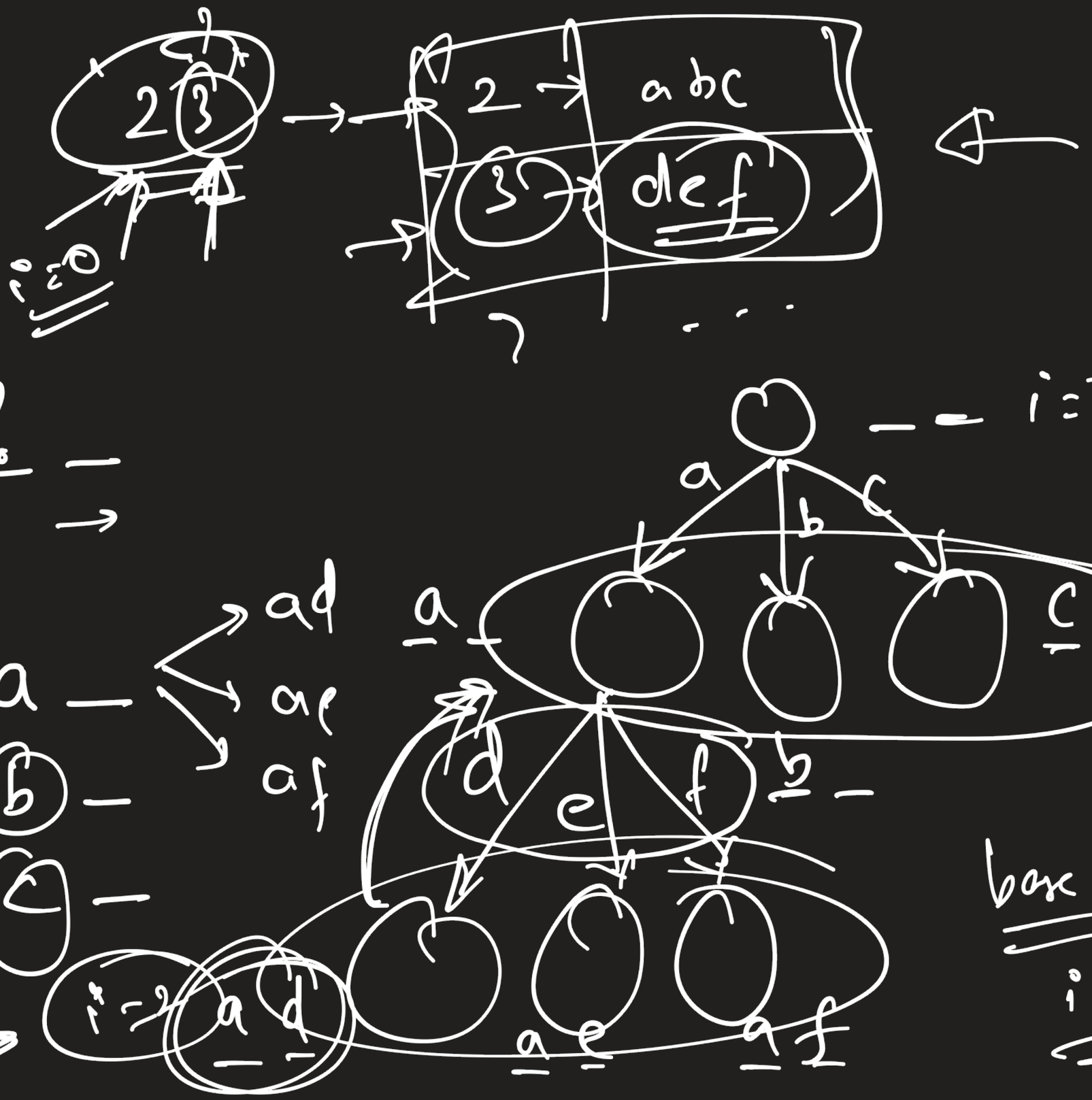
$O(h)$

S: $O(N)$ $+O(N)$ → $O(N)$

board

$O(N)$ $+O(N^2)$ → $O(N^2)$

↳ 2-d array


2
3
2
abc
3
def
i:=0
?
a
b
c
ad
ae
af
a
b
c
i:=1
d
e
f
i:=2
a
d
a
e
a
f
base:
i==N
print(Sbk)