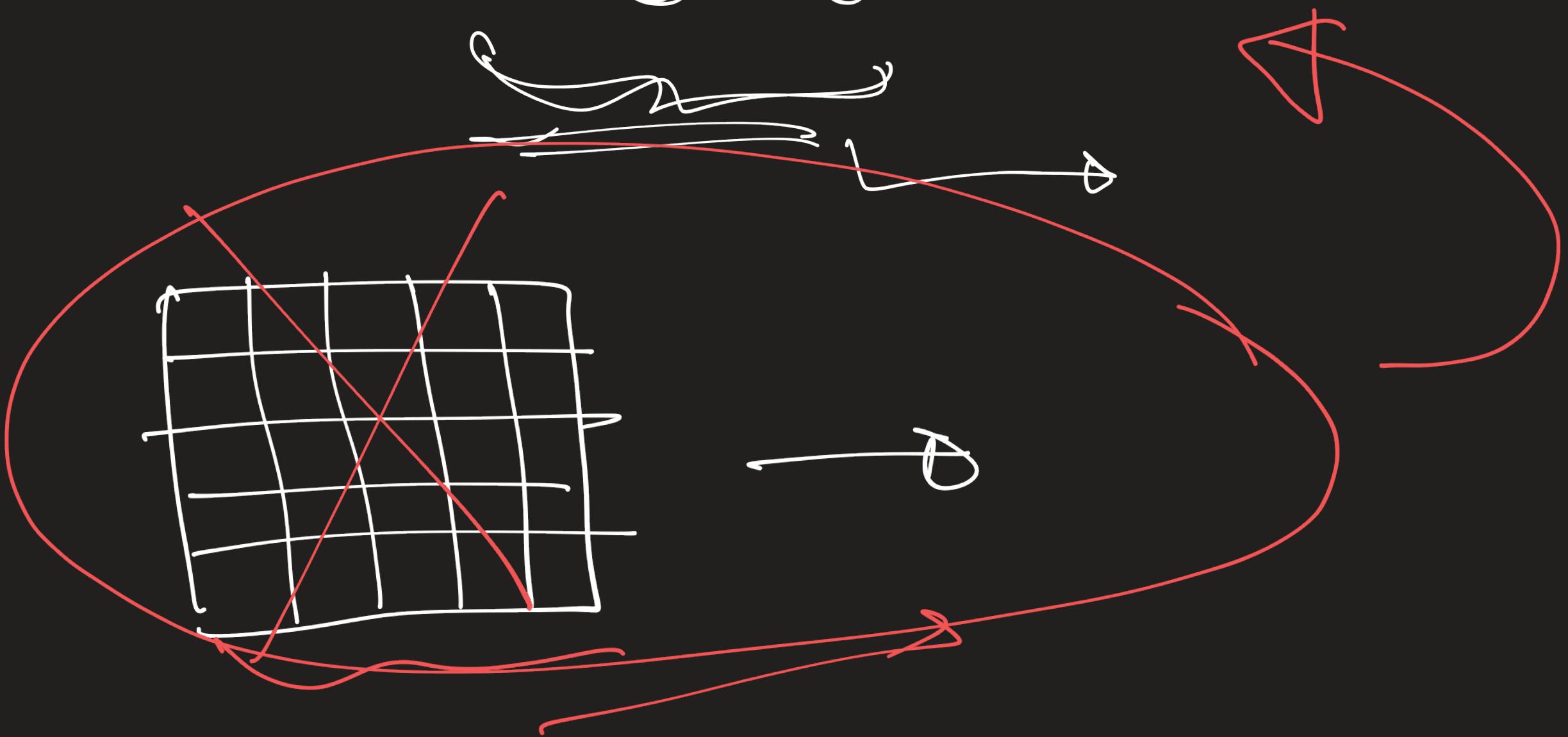
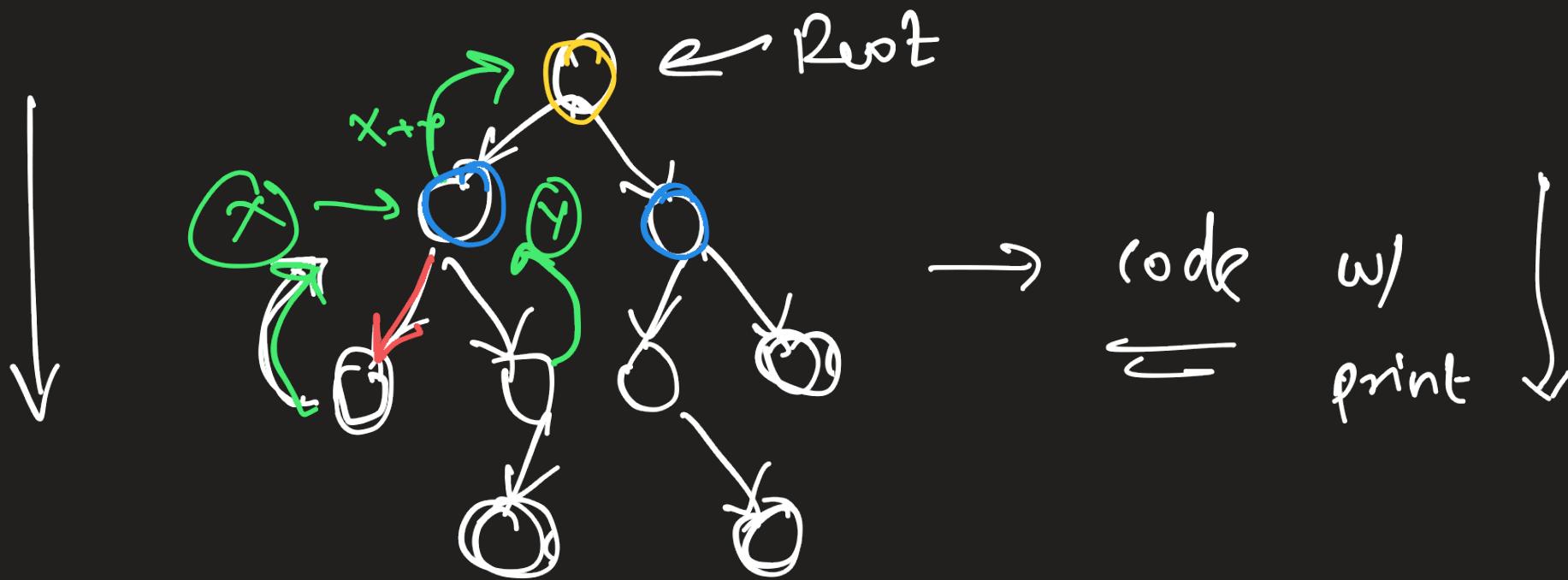
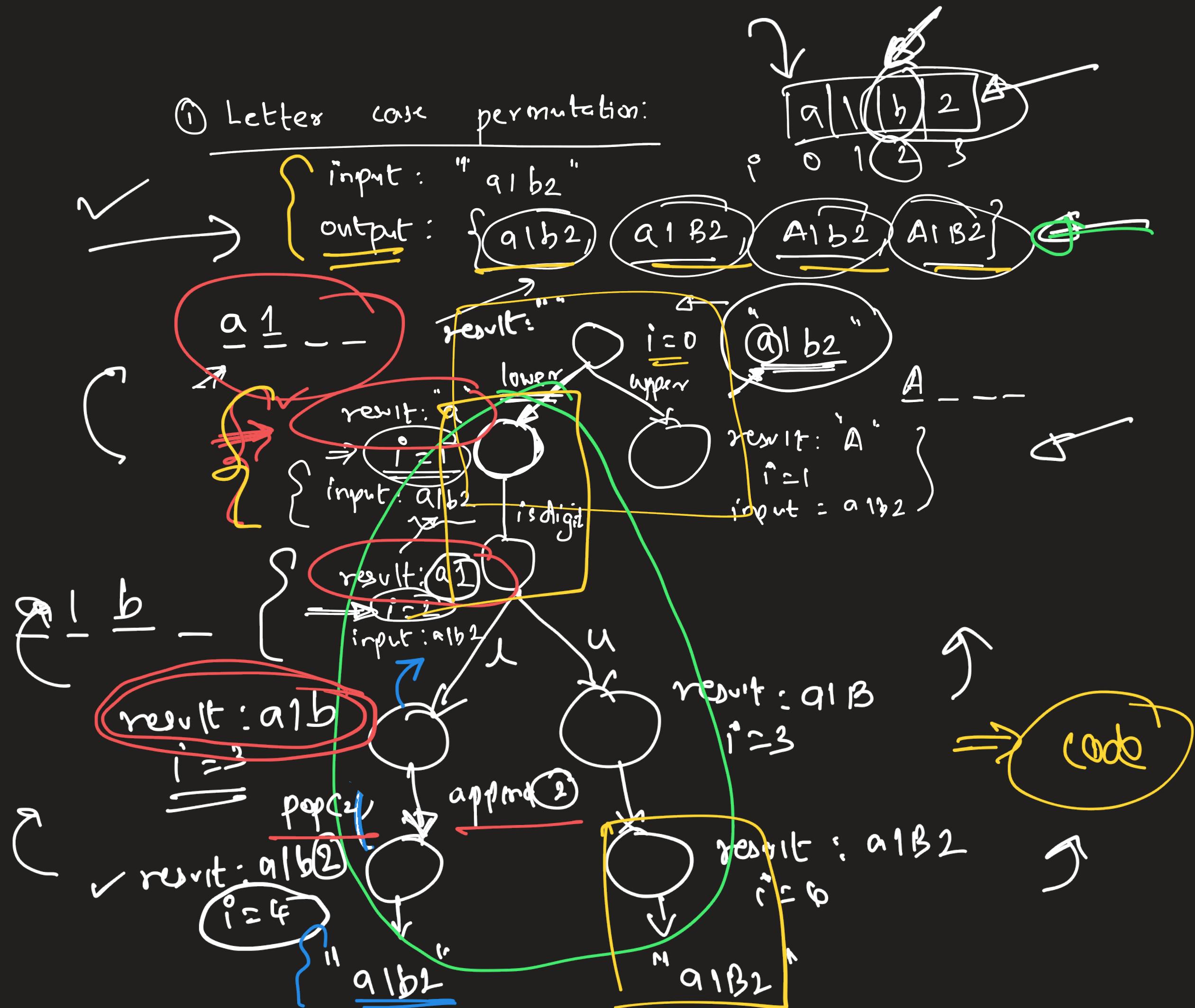


Recursion :-



① Letter case permutation:



$s[n]$

for ($i : 0 \dots n$) $\leftarrow O(1)$

$s = s + "f"; \cancel{O(n)}$



$O(n)$ time

PT

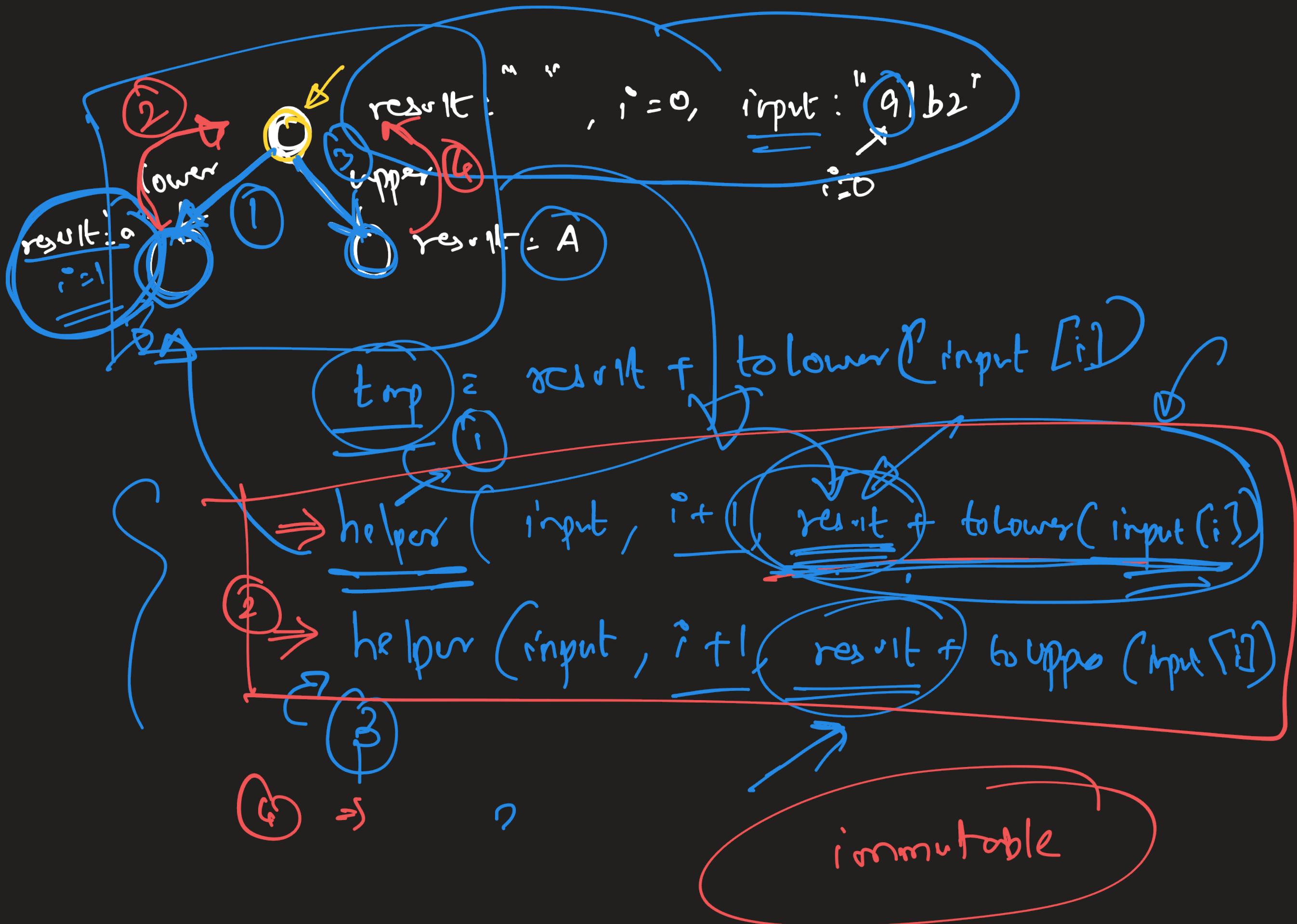
$\rightarrow O(N)$

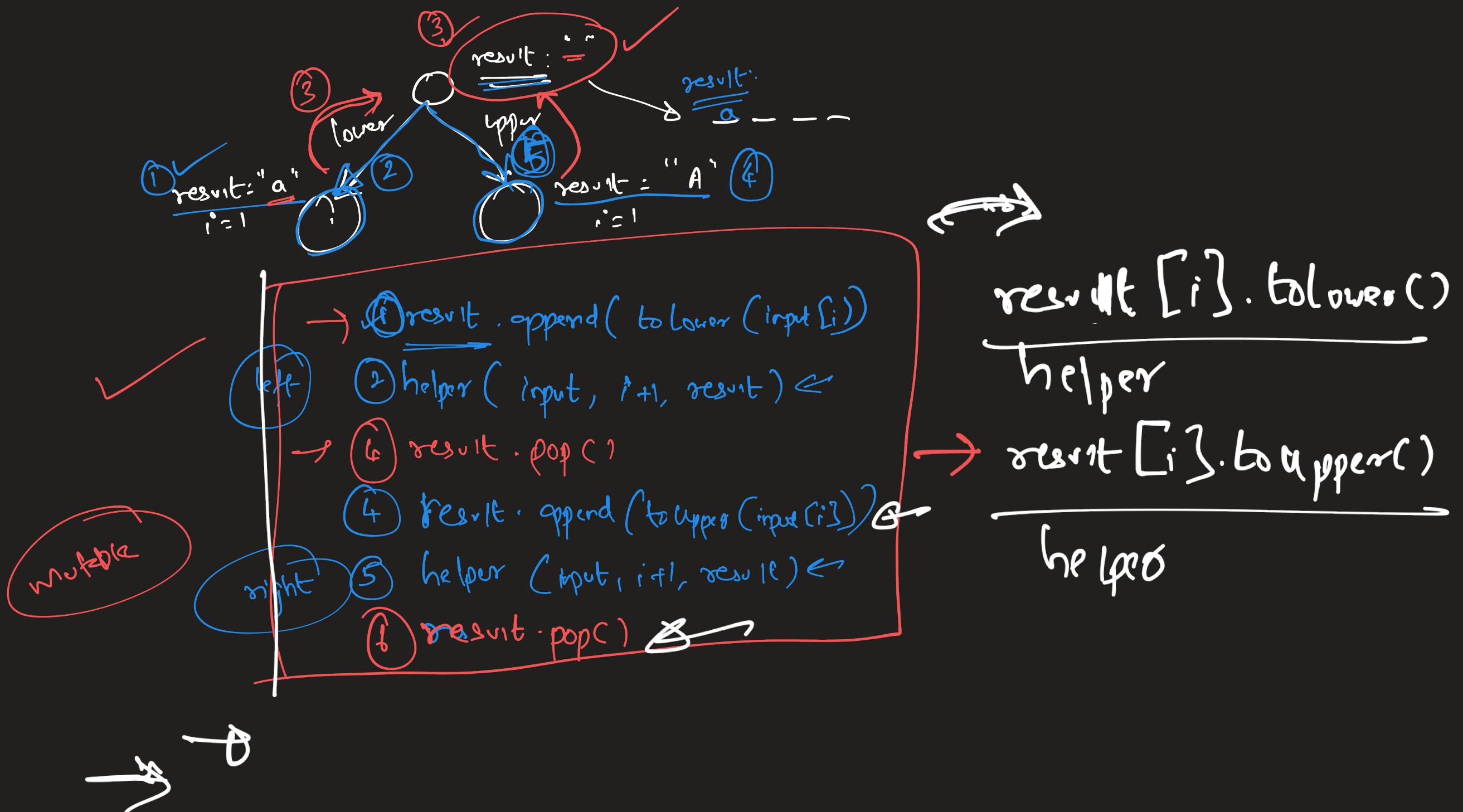
$\rightarrow O(N^2)$

```

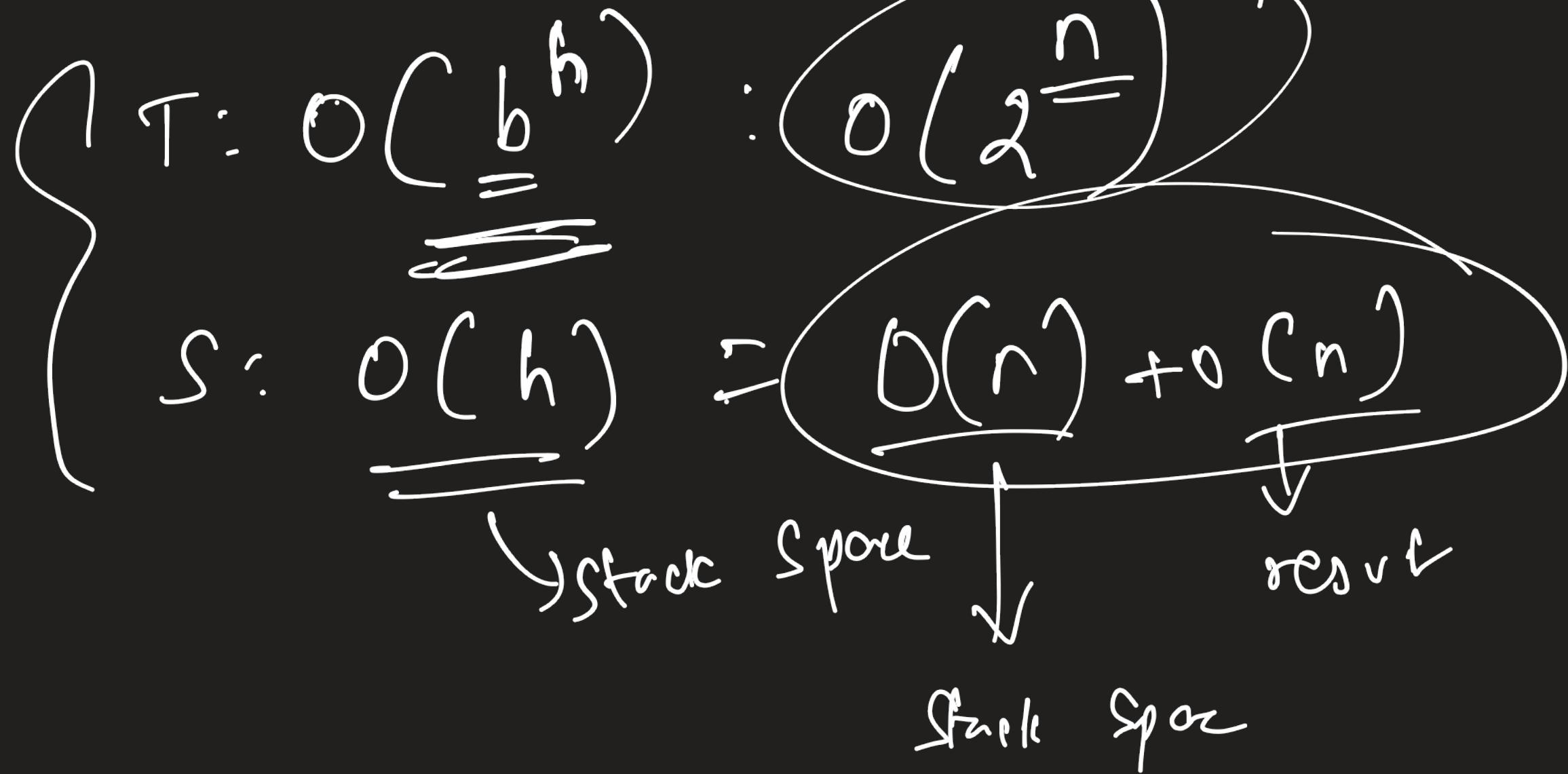
    | lepermute( string input) {
    |     return helper( input, 0, "" );
    |
    |     void helper( string input, int i, string result), vector<string>
    |     results ) {
    |         if( i == input.length() ) {
    |             print( result ); → results.append( result )
    |             return ;
    |
    |         } 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] ) )
    |
    |         }
    |
    |     }

```





Rec: / ^{other case}
permutation



②

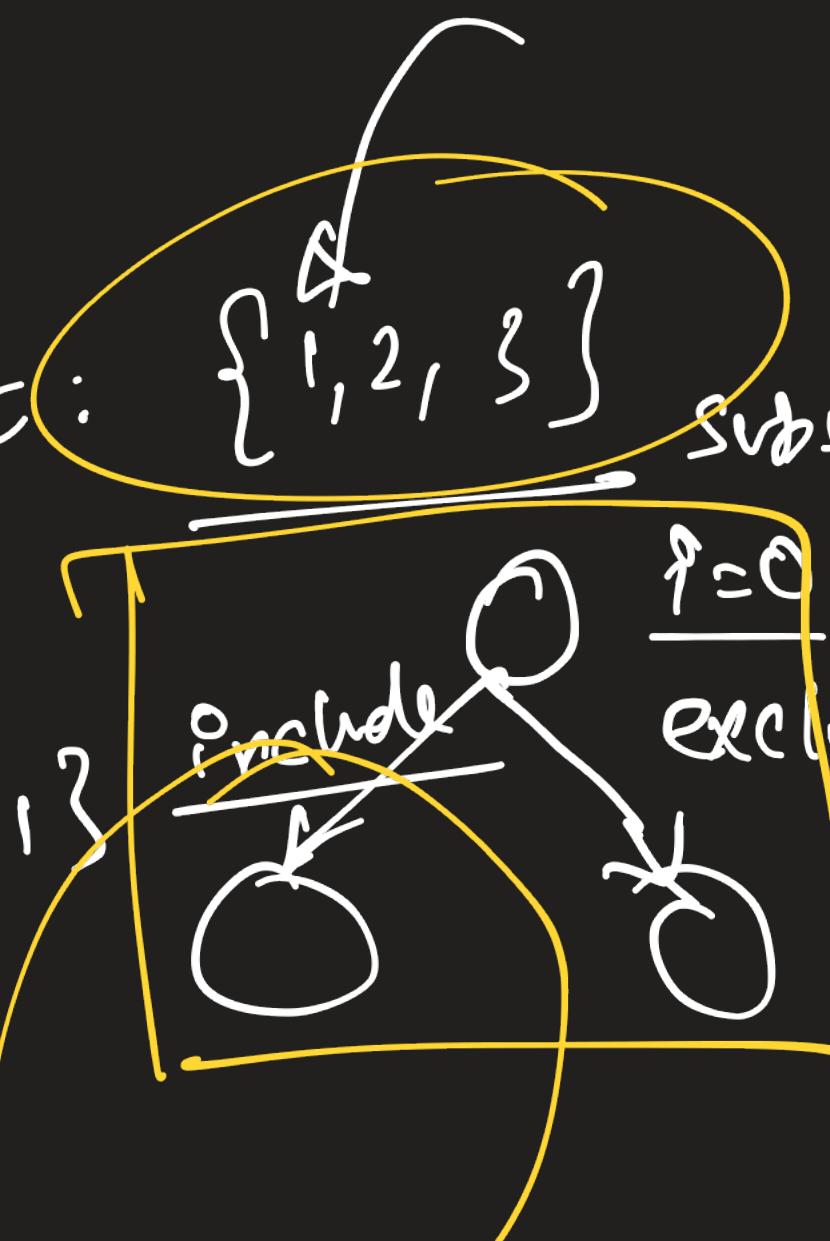
Subset:

input:

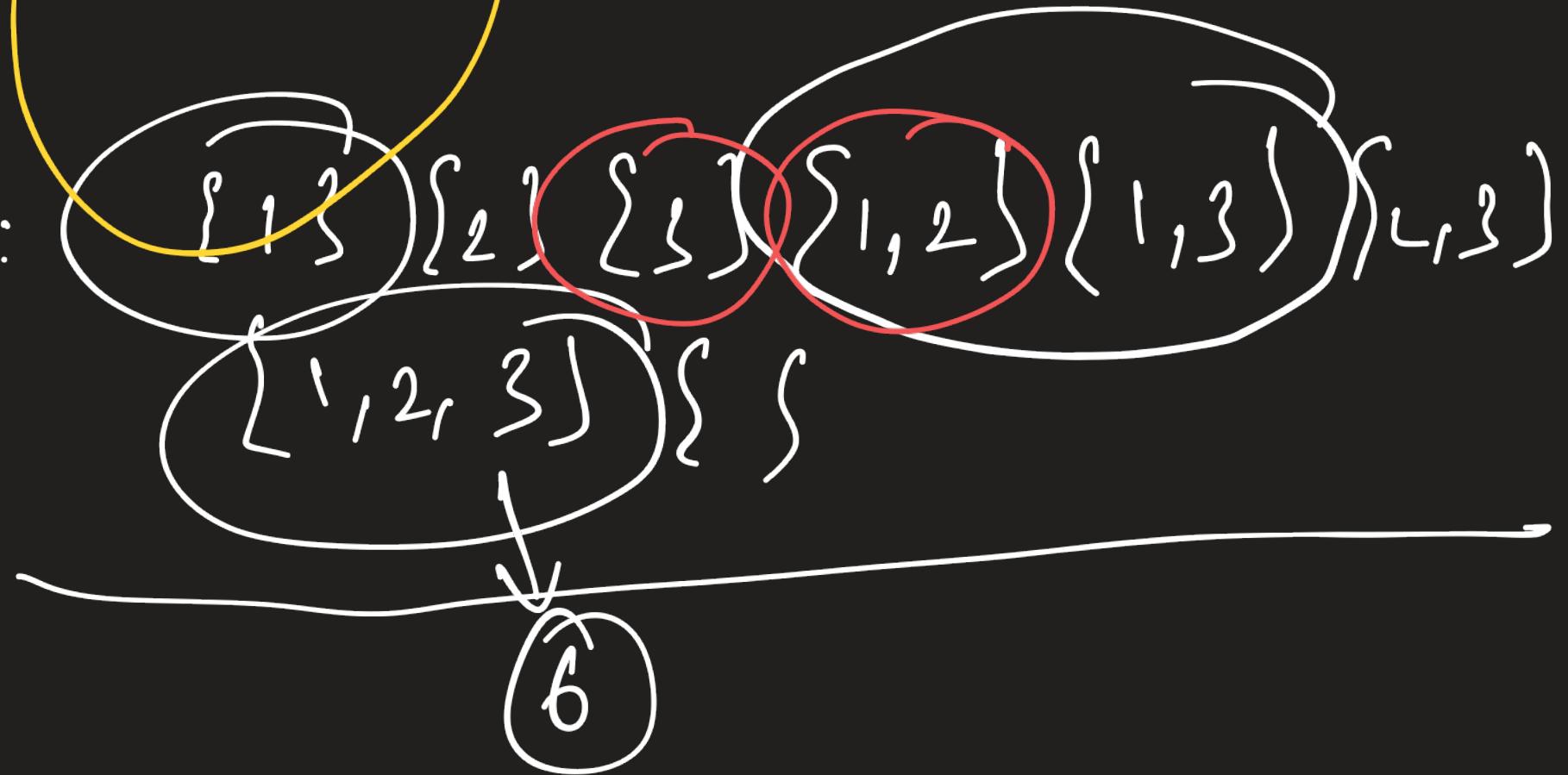
{1, 2, 3}

Subset: {3}

Subset:
 \checkmark $i=1$
input:



Output:



input: $\{1, 2, 3\}$

Subset: $\{\}$
 $i=0$

Subset: $\{1\}$
 $i=1$

$\{\}$
 $i=1$

Subset: $\{1, 2\}$
 $i=2$

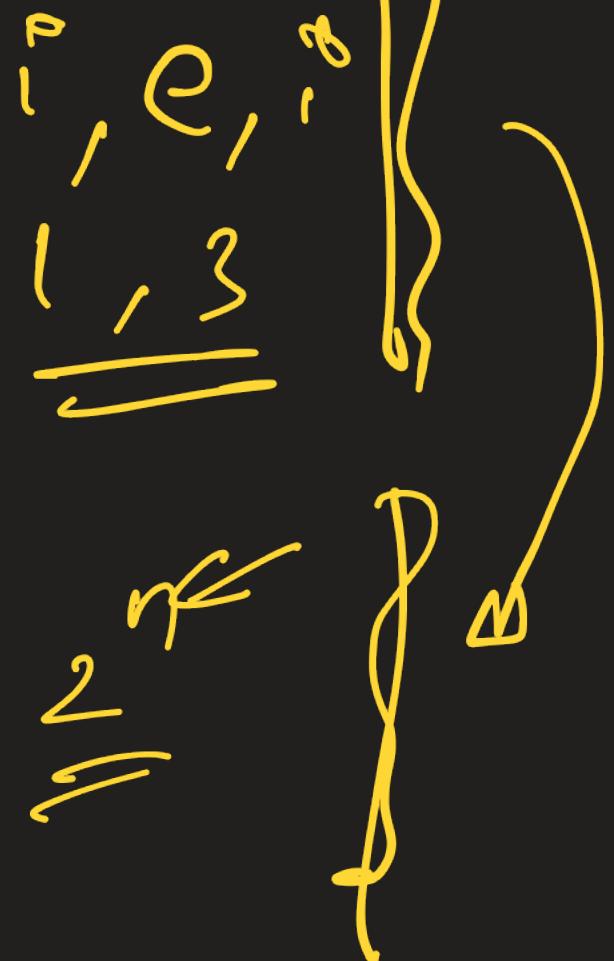
$\{1, 2, 3\}$

halting

Condition

$\{1, 2, 3\}$

$\{1, 2\} \quad \{1, 3\} \quad \{1\}$



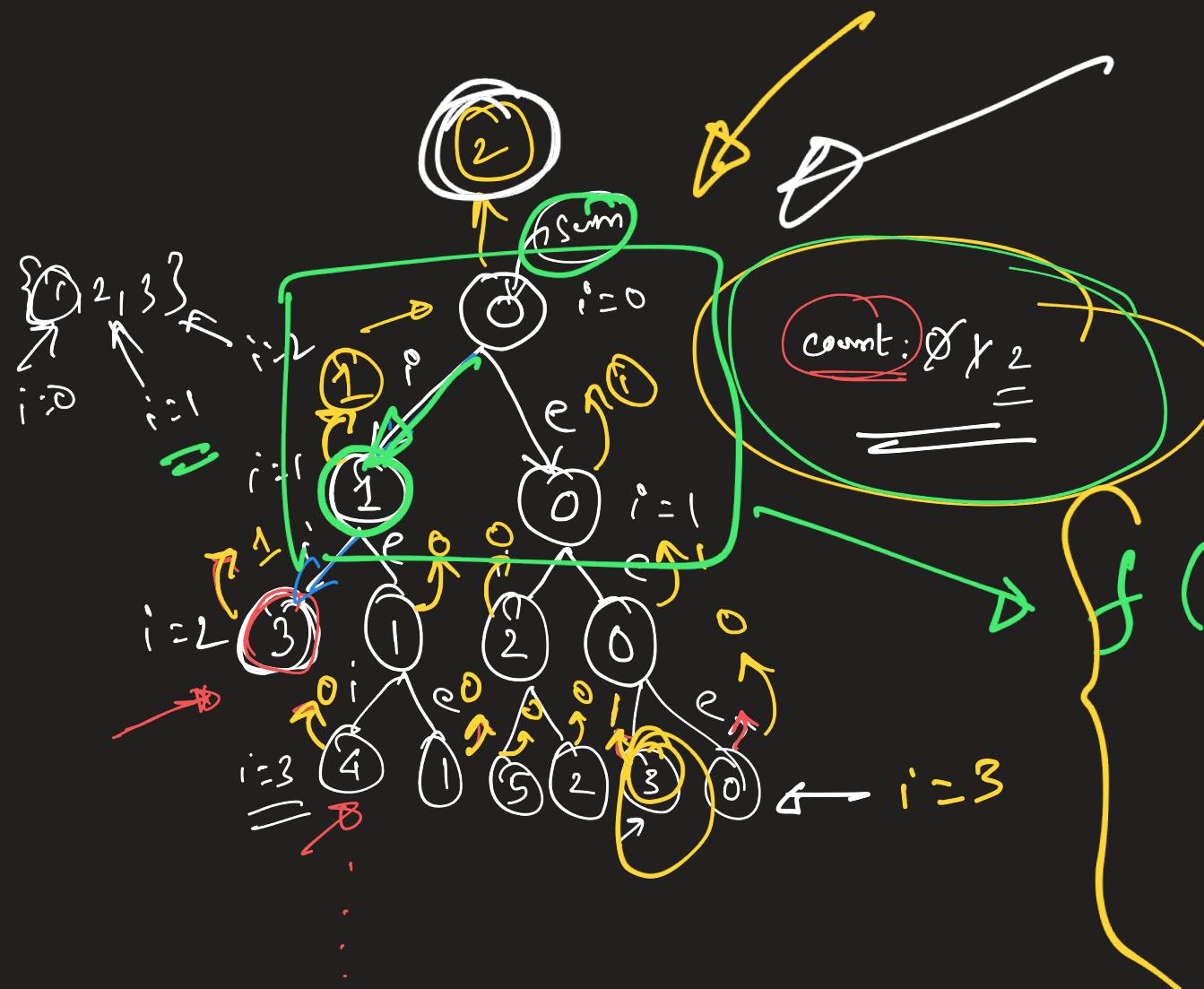
3

Subset Sum :- (Counting Subsets)

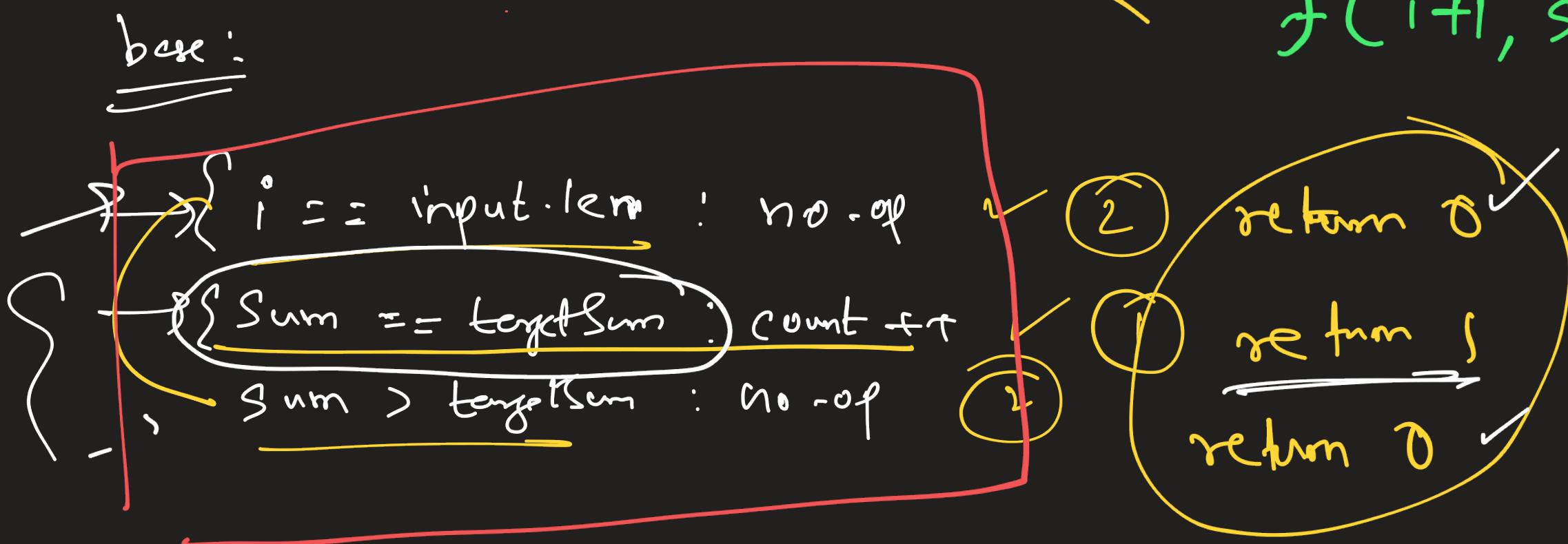
Input : $\{1, 2, 3\}$ ↗

target-Sum : $\begin{array}{c} 3 \\ = \end{array}$ ↙

result : $2 \rightarrow \{1, 2\} \quad \{3\}$



✓

$$f(i, \text{sum}) = f(i+1, \text{sum} + \text{input}[i]) + f(i+1, \text{sum})$$


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

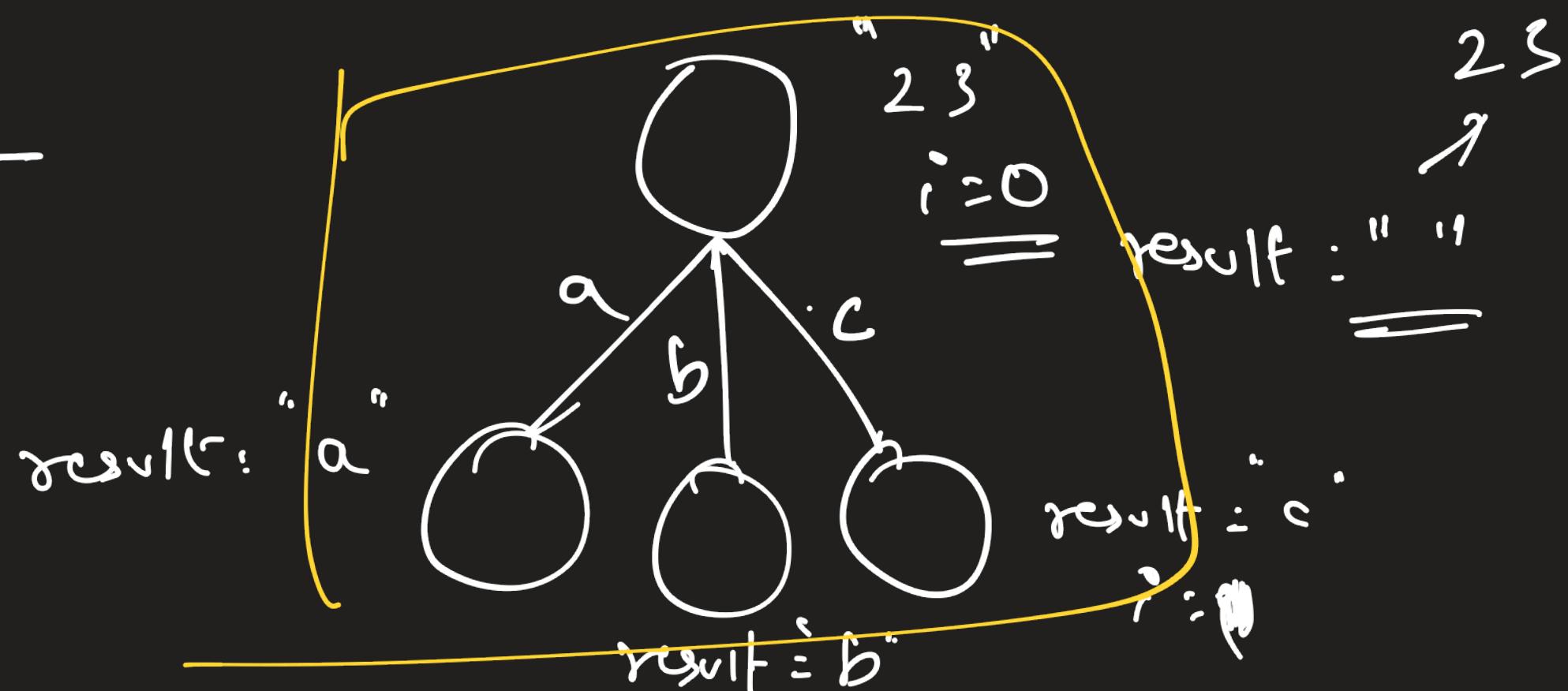
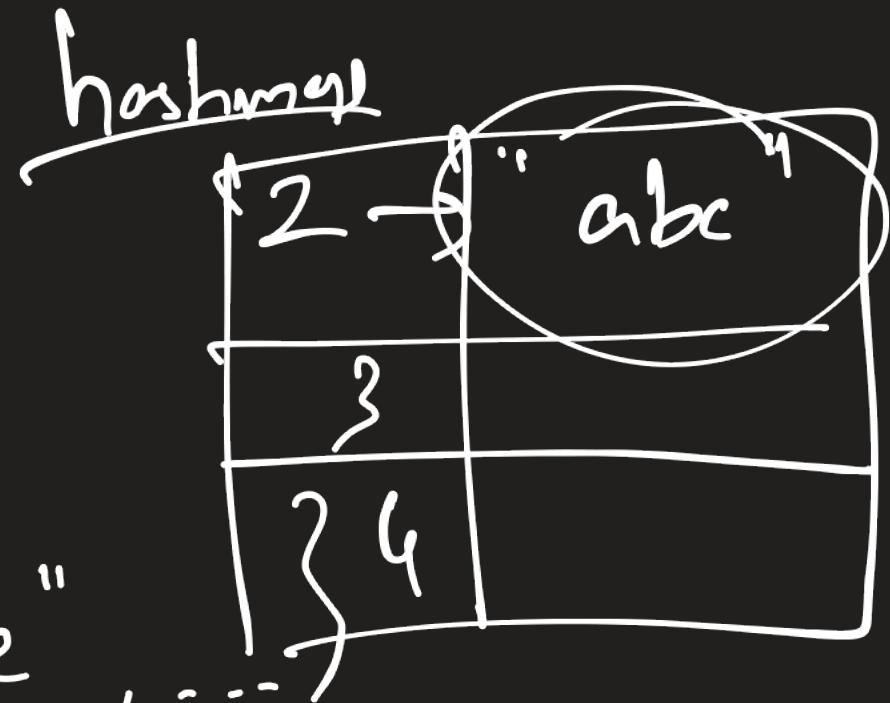
```
int helper( vector<int> input, int targetSum,  
            int i, int sum ) {  
  
    base = {  
        if ( sum == targetSum ) return 1; // base case  
        if ( sum > targetSum || i >= input.length() ) return 0;  
    };  
  
    return helper( input, targetSum, i+1, sum + input[i] )  
 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

④ Dial pad

input = "23"

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

c
b
a
—



5

Generate parenthesis

$n = 3$ pairs

✓ ((()))

~~((x))~~

✓ ()()()

~~(c)(x)~~

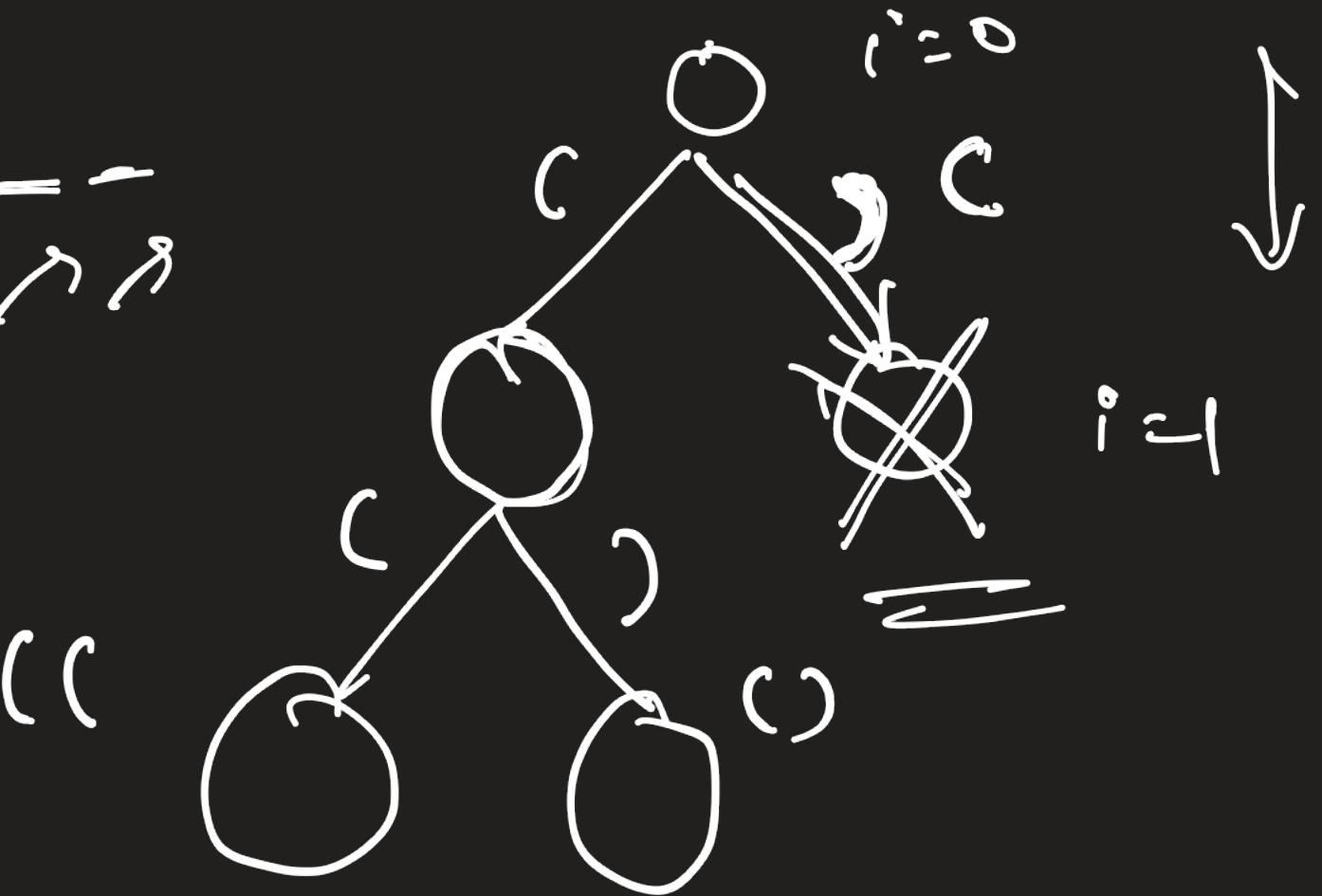
✓ (())()

✓ () (())

.

X)
((

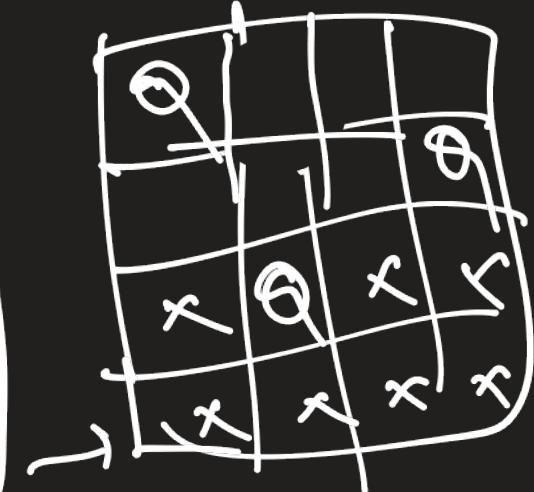
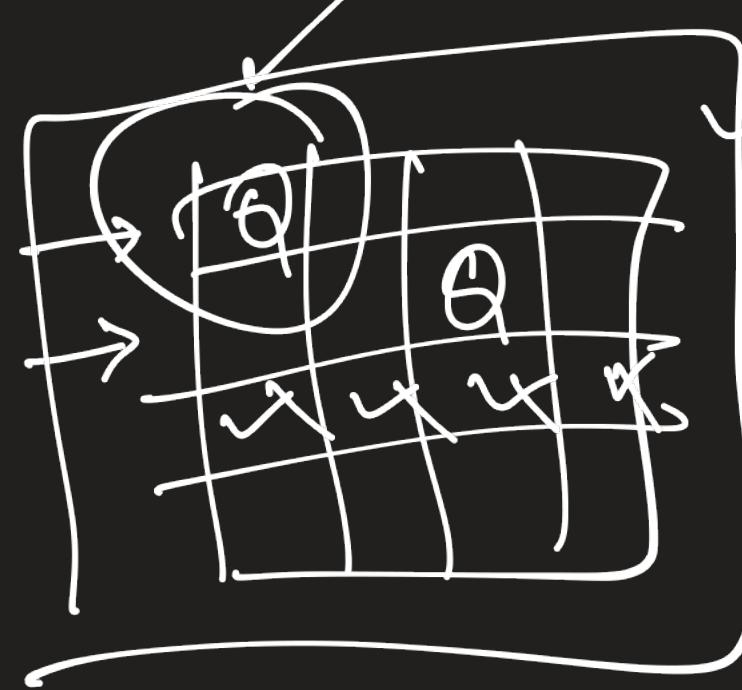
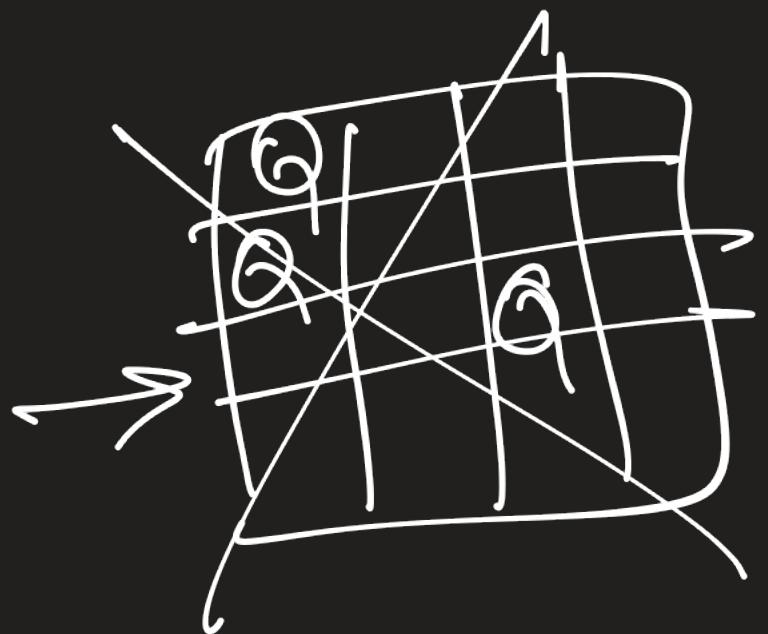
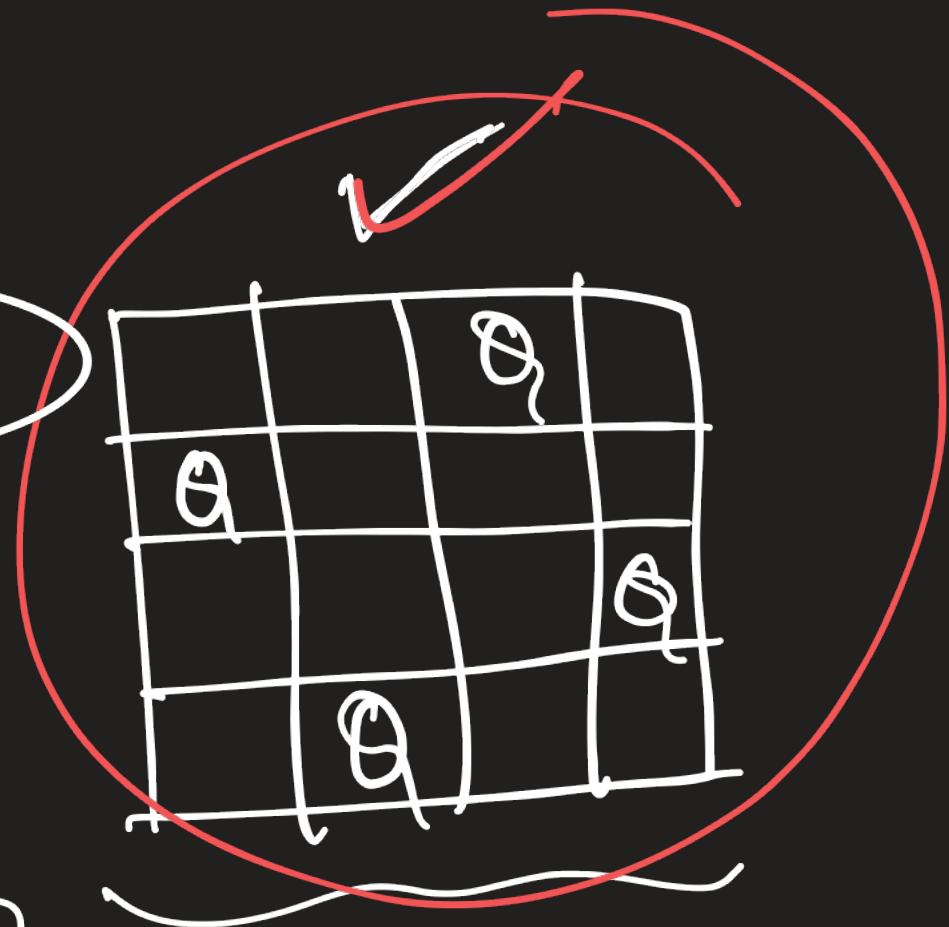
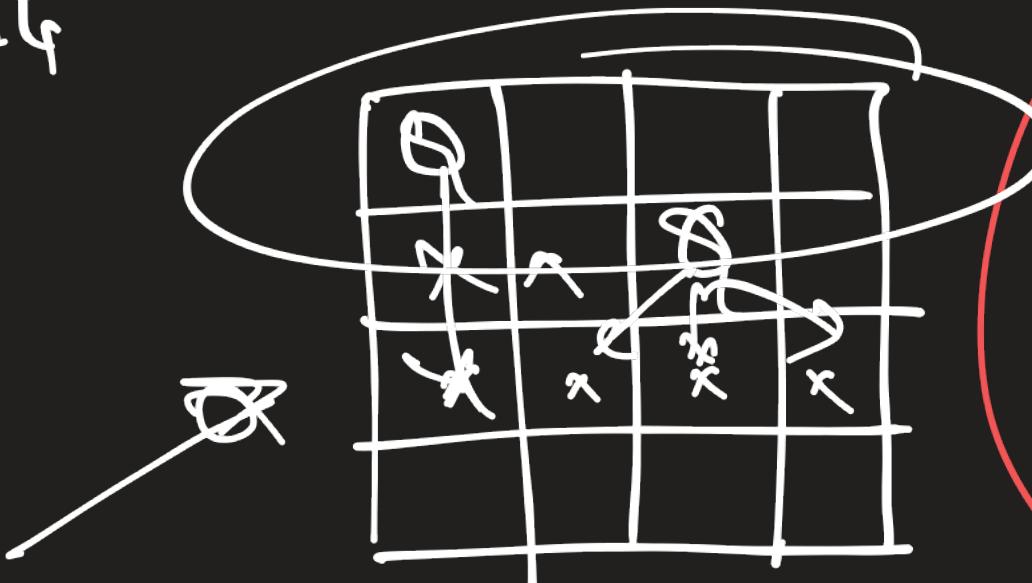
— — — — —
↗ ↗ ↗ ↗ ↗

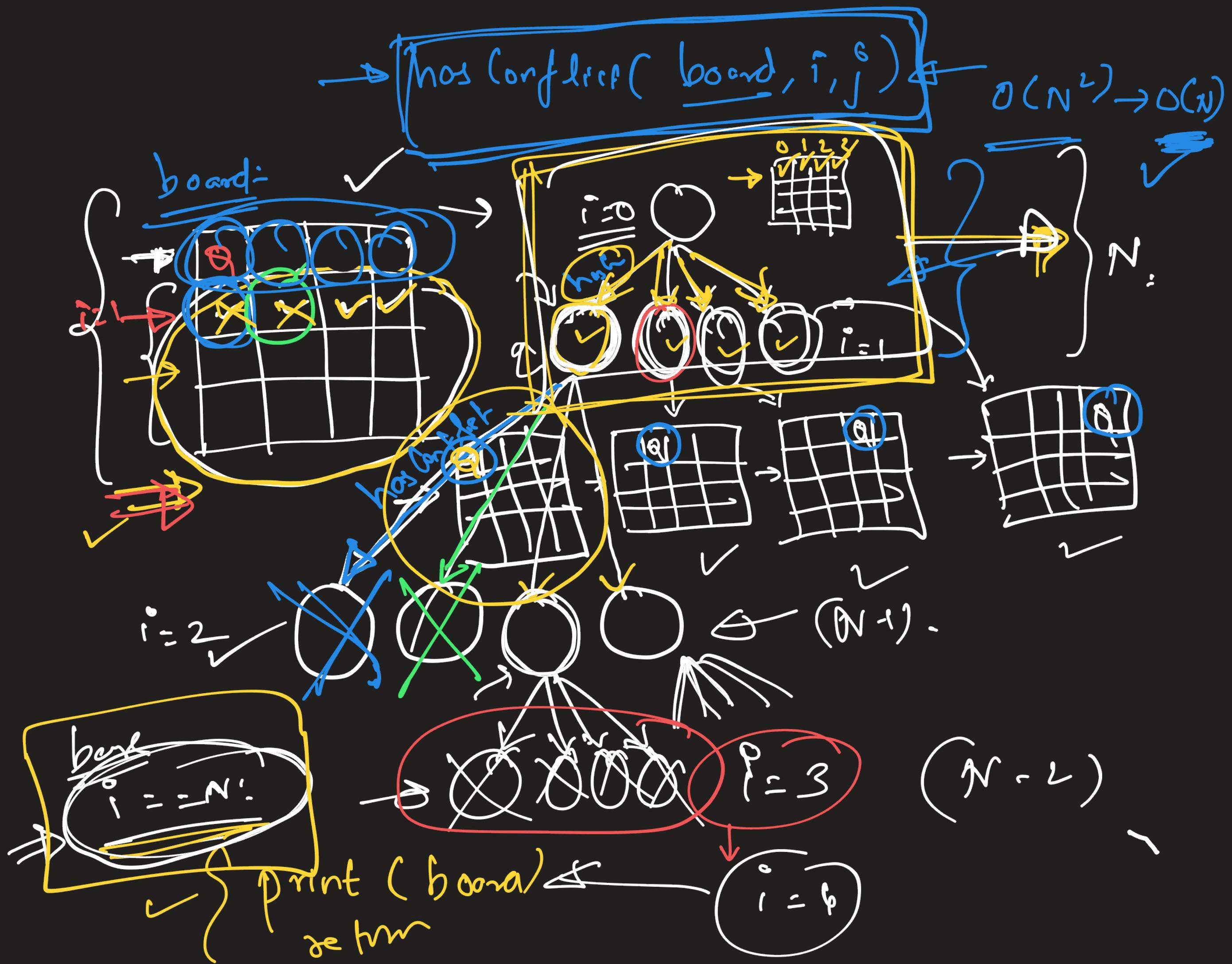


6 N-Queens: ✓

chess board : $N \times N$

$N=4$





```
void NO(int N) {  
    int board[N]; ✓  
    helper(N, board, 0); ✓  
}
```

```

void helper( int N, int[ ] board, int i) {
    if( i == N) {
        print( ^board)
        return;
    }

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

```

Diagram illustrating the backtracking algorithm:

- Row:** A vertical red line on the left representing the current row being filled.
- Column:** A horizontal red line at the top representing the current column being filled.
- Choices:** A horizontal red line at the bottom representing the choices available for the current row.
- Board State:** The state of the board is shown as a sequence of circles labeled 1, 2, 3, 4, ... along the row line.
- Conflict:** A red circle labeled 0 is shown near the start of the row line, indicating a conflict or an empty slot.
- Print:** A red oval labeled "print(^board)" indicates the output of a valid configuration.
- Return:** A red arrow labeled "return;" indicates the exit from the recursive call.
- For Loop:** A red arrow labeled "for(j=0; j < N; +j)" points to the loop variable.
- If Condition:** A red arrow labeled "if(!hasConflict(board, i, j))" points to the condition that checks for conflicts.
- Assignment:** A red arrow labeled "board[i] = j;" points to the assignment of a value to the current cell.
- Recursive Call:** A red arrow labeled "helper(N, board, i+1)" points to the recursive call for the next row.
- Annotations:**
 - A yellow double underline is placed under the parameter `i` in the function header.
 - A yellow double underline is placed under the condition `i == N`.
 - A yellow double underline is placed under the assignment `board[i] = j;`.
 - A yellow double underline is placed under the parameter `N` in the recursive call.
 - A yellow double underline is placed under the parameter `board` in the recursive call.
 - A yellow double underline is placed under the parameter `i+1` in the recursive call.
 - A yellow arrow labeled "row" points to the row line.
 - A yellow arrow labeled "column" points to the column line.
 - A yellow arrow labeled "choice" points to the choices line.
 - A yellow arrow labeled "board" points to the board state.
 - A yellow arrow labeled "conflict" points to the conflict circle.
 - A yellow arrow labeled "print" points to the print annotation.
 - A yellow arrow labeled "return" points to the return annotation.
 - A yellow arrow labeled "for" points to the for loop annotation.
 - A yellow arrow labeled "if" points to the if condition annotation.
 - A yellow arrow labeled "assignment" points to the assignment annotation.
 - A yellow arrow labeled "recursion" points to the recursive call annotation.

hash:

row

column

	K	V
1	0	2
2	3	0
3	1	3

2-d array

0	1	2	3
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0

col

2	0	3	1
---	---	---	---

$i = 0 \quad 1 \quad 2 \quad 3 \quad 4$

\rightarrow (row)

