1]

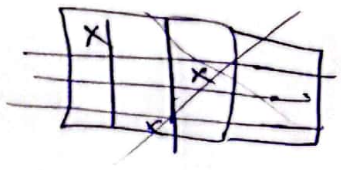
Recursion & Back Tracking

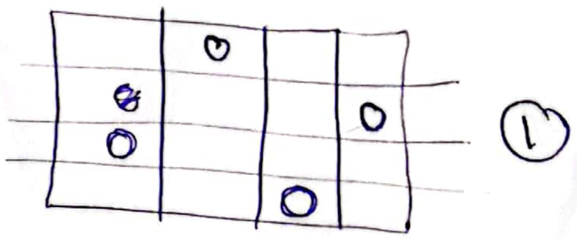
* لازم تكون فى ال Base ال recursion يخرج منها

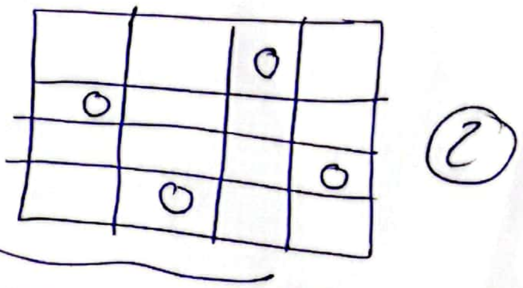
Q1] nxn chess Board, print all possibilities To put n Queens

Soln

n=4 =>



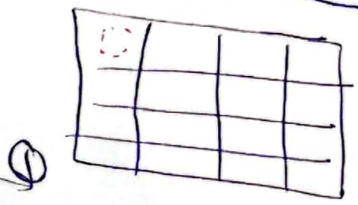




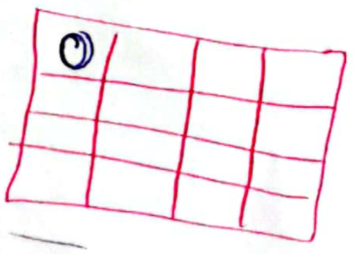
دى كل الاحتمالات at n=4

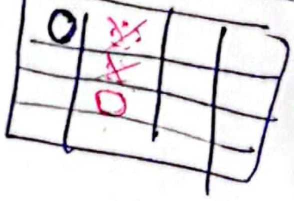
كمثال هل هينفع احط Q عند (0,0) !!

① لو نعم احطها



السه ما عملناش recursion





هنبدأ ندخل على col اللى بعده

هل ينفع !! لو ← نعم حطها وادخل على col اللى بعده

```
fun(col){
    for(row=0
}
```

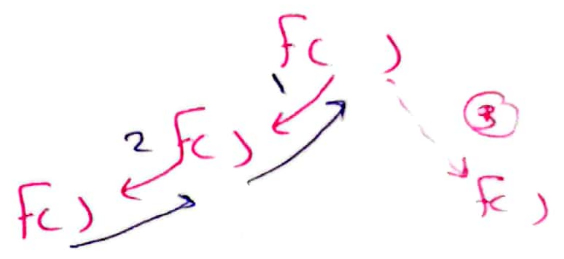
2]

الـ Back Tracking ⇐ بعد ما أرجع ⇐ امسح Q

انت كل شوية بتعمل call للـ function بـ (col+1) ⇐ لما col==n يعني كدا انت حليت

Code

what is Recursion Tree?? ⇒ انك ترسم Function



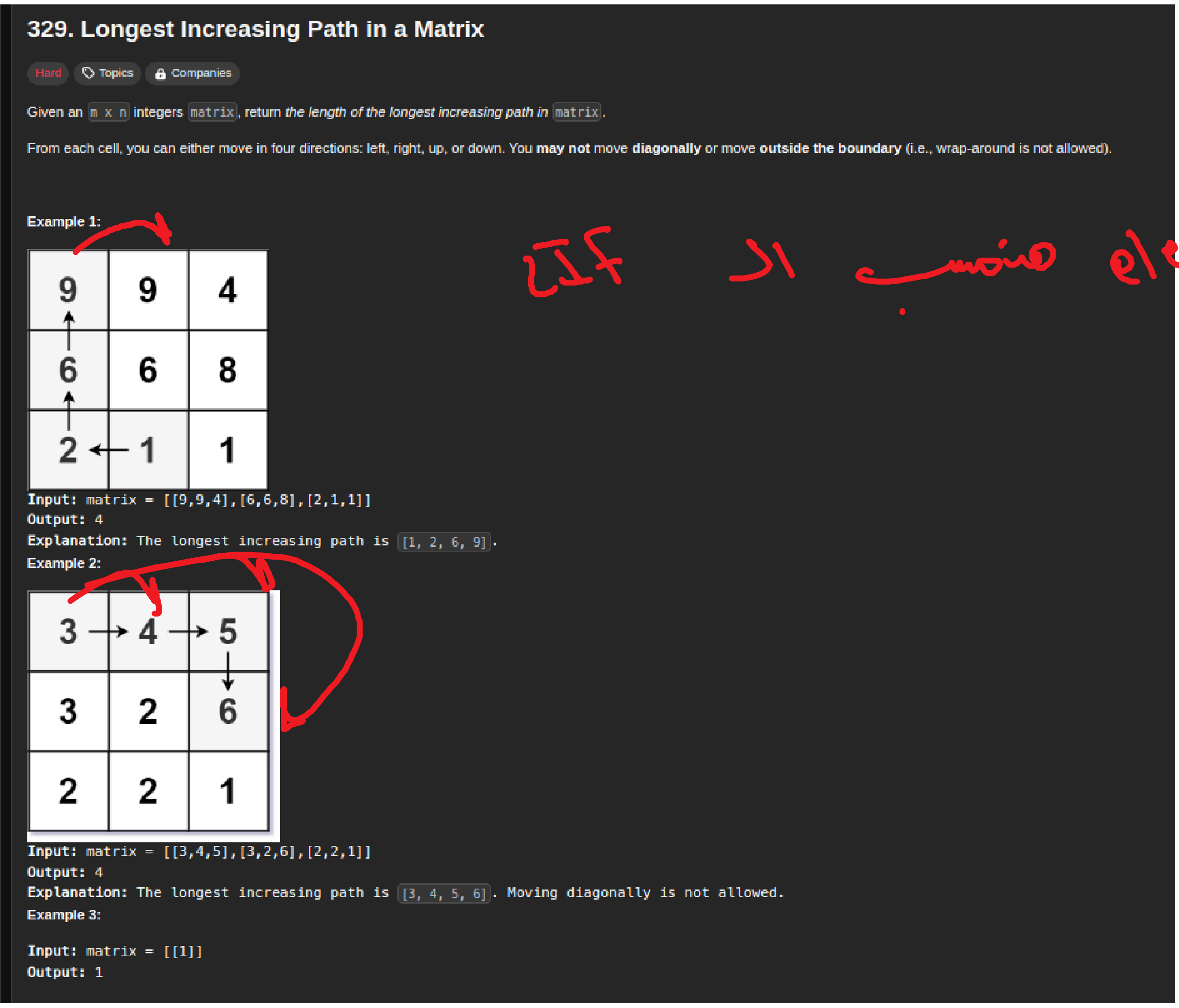
Q2] print (1→10)

في طريق الـ Backward

```
void( i ){
    i==0 → return
  print(i-1) ↷ 10 ↷ 9 - - - 1
  cout<<i<<" ";
main → prin(10)
```

output is 1,2,3 - - - 10

بعد ماوصلت النهاية ابدأ خد الاكشن



329. Longest Increasing Path in a Matrix

Given an `m x n` integers `matrix`, return *the length of the longest increasing path in* `matrix`.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

كل elements منصب ال [illegible]

Example 1:

9	9	4
6	6	8
2	1	1

```
Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:

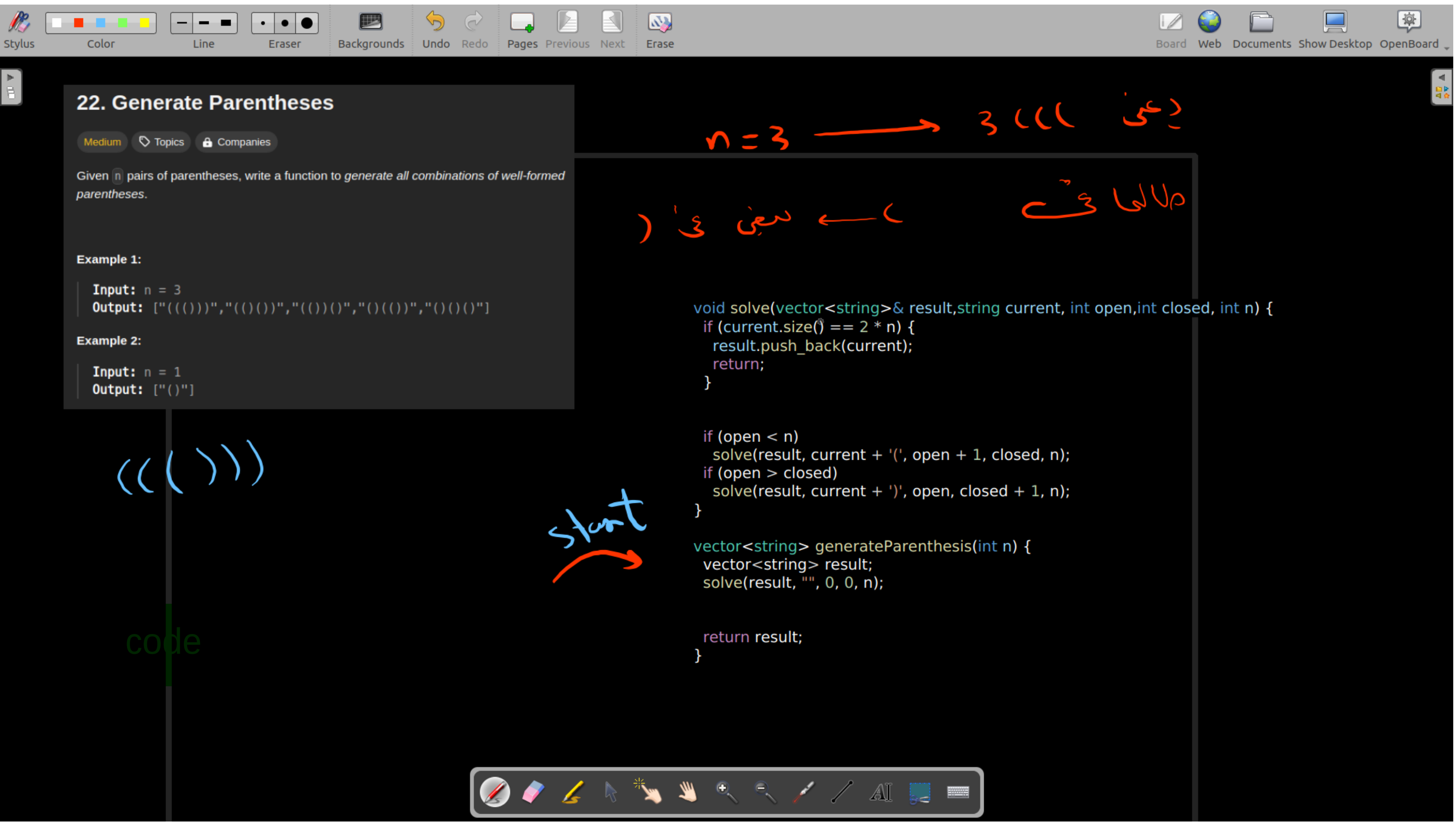
3	4	5
3	2	6
2	2	1

```
Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.
```

Example 3:

```
Input: matrix = [[1]]
Output: 1
```

Code



22. Generate Parentheses

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

Example 1:

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

Example 2:

```
Input: n = 1
Output: ["()"]
```

n=3 → 3 (((يعني

طالما في (← يعني في)

((( )))

start

code

```
void solve(vector<string>& result,string current, int open,int closed, int n) {
  if (current.size() == 2 * n) {
    result.push_back(current);
    return;
  }

  if (open < n)
    solve(result, current + '(', open + 1, closed, n);
  if (open > closed)
    solve(result, current + ')', open, closed + 1, n);
}

vector<string> generateParenthesis(int n) {
  vector<string> result;
  solve(result, "", 0, 0, n);

  return result;
}
```