Backtracking 2

Wednesday, 17 November 2021 9:02 PM

Find last digit of the number n.

n = 1386

n % 10 = 6

Remove the last digit of n.

n = 1386 → n = 138

n = 154 → n = 15

n / 10

$\frac{154}{10} = 15$

Find sum of all digits of a number n. $n \geq 0$

n = 1386   Ans = 1 + 3 + 8 + 6 = 18

```
int sumDigits(n) {
    if(n < 10) return n;
    return (n%10) + sumDigits(n/10);
}
```

SC = O(#digits)   TC = O(#digits) → $\log_{10}(n)$

$\log_{10} n = x$ ↓ $10^x = n$

x = #digits of n

$10^4 \leq 13842 \leq 10^5$

```
int sumDigits(n) {
    sum = 0;
    while(n > 0) {
        sum = sum + (n%10);
        n = n / 10;
    }
    return sum;
}
```

TC = O(#digits)

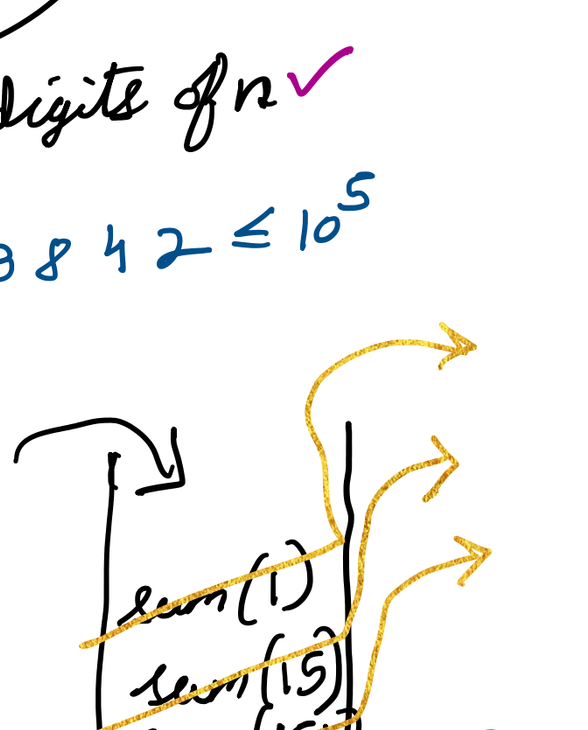
SC = O(1)

10 (Ans)

```
int sum(154) {
    if(n < 10) X
    return 4 + sum(15)
}
```

sum(15): if(n < 10) X; return 5 + sum(1)

sum(1): if(n < 10) return 1;



3

$\log_b a \to \frac{\log_e a}{\log_e b}$

$\log_{10} n \to \frac{\log(n)}{\log(10)} = \frac{1}{\log(10)} * \log(n)$

$= O(\log(n))$

Q → Find sum of digits till number is a single digit number.

n = 138695

sumDigits = 1+3+8+6+9+5 = 32

sumDigits = 3+2 = 5 (Ans)

n = 959595959595   Ans =

sumDigits = (9+5)*6 = 84

sumDigits = 8+4 = 12

sumDigits = 1+2 = 3 (Ans)

```
long solve(n) {
    → if(n < 10) return n;
    → n = sumDigits(n);
    → return solve(n);
}
```

very less iterations

```
long sumDigits(n) {
    if(n < 10) return n;
    return (n%10) +
        sumDigits(n/10);
}
```

TC = O(#digits) = O($\log_{10} n$)

SC = O(#digits)

4999....9 (18 digits) = 9 * 18 = 162 → 9

999....9 (100 digits) → 9 * 100 = 900

$10^{100} - 1$

Q → Fast Power

$a^b \% m$ → $((a*a)\%m)^{b/2} \% m$ b → even

→ $a * ((a*a)\%m)^{(b-1)/2} \% m$ b → odd

$3^{20} = (3^2)^{10} = 9^{10}$

$3^{20} \to 9^{10} \to 81^5$

$4^5 = 4 * (16)^2 = 4 * 5^2$

$= 4 * 25$

$= 100 \% 11$

$= 1$

```
int solve(a, b, m) {          // 3, 20, 11
    if(b == 0) return 1;
    if((b&1) == 0) {
        return solve((a*a)%m, b/2, m);     // 0 ≤ ↑ < m
    } else {
        return (a * solve((a*a)%m, (b-1)/2, m)) % m;
    }
}
```

TC = O($\log_2 b$)

SC = O($\log_2 b$)

(9*9) % 11 = 81 % 11 = 4

solve(3, 20, 11) → solve(9, 10, 11) → solve(4, 5, 11)

(4 * solve(5, 2, 11)) % 11   (4*4) % 11 = 16 % 11 = 5

3 * solve(9, 0, 11) ← solve(3, 1, 11)

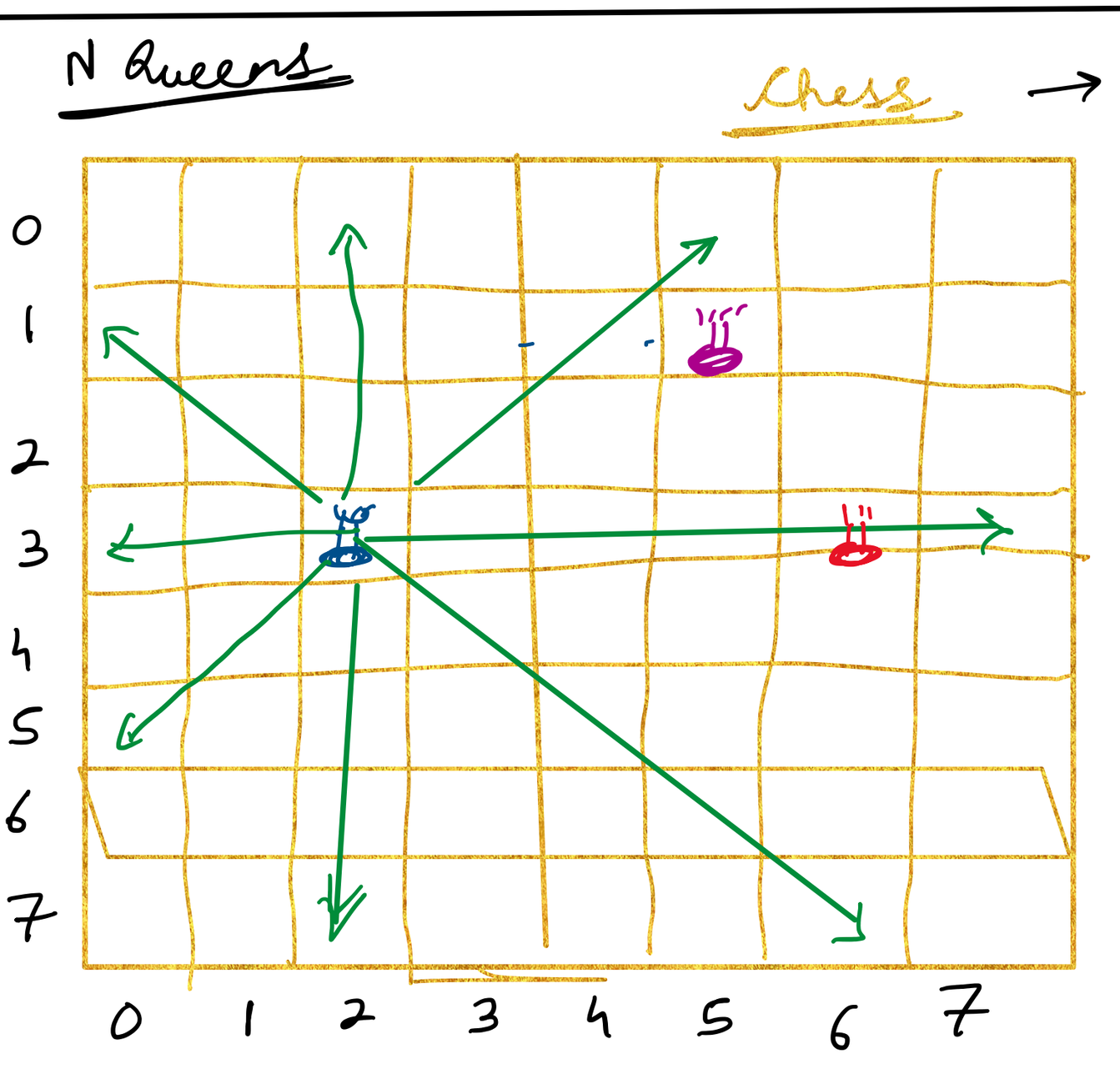
4 * 3 % 11

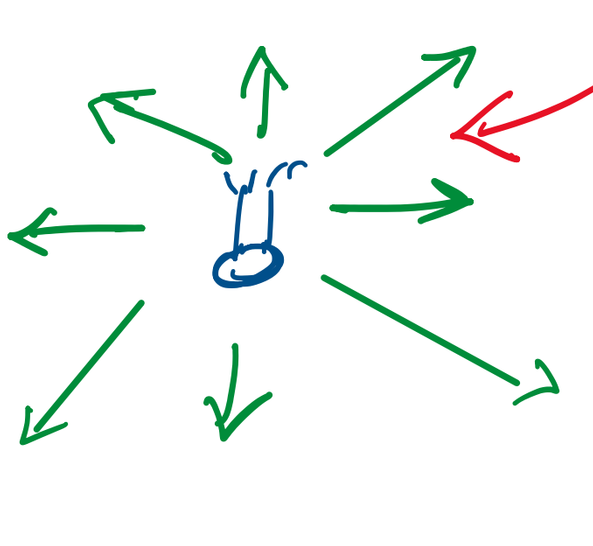
= 12 % 11

= 1

N Queens

Chess → pawns, rook, knight, bishops, queen, king





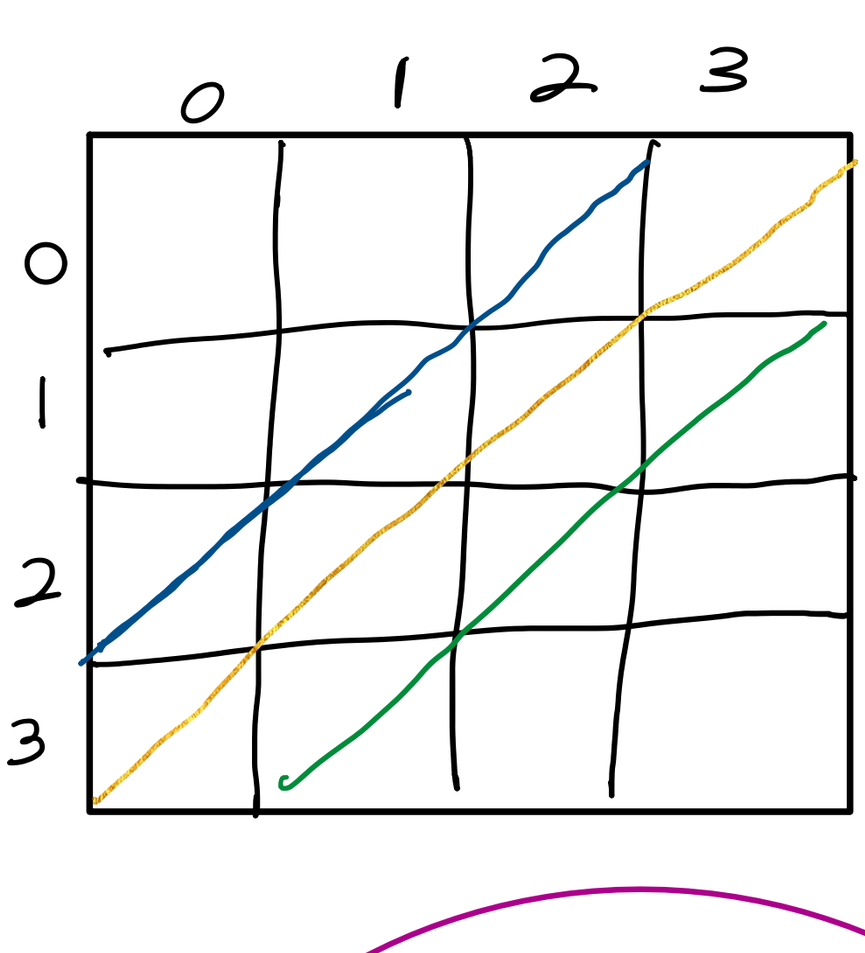
check if 2 queens at (x_1, y_1) & (x_2, y_2) can attack each other in one move.

(3, 2) (3, 6) → Ans = True

boolean check(x_1, y_1, x_2, y_2) {...}

same row → $x_1 == x_2$

same column → $y_1 == y_2$

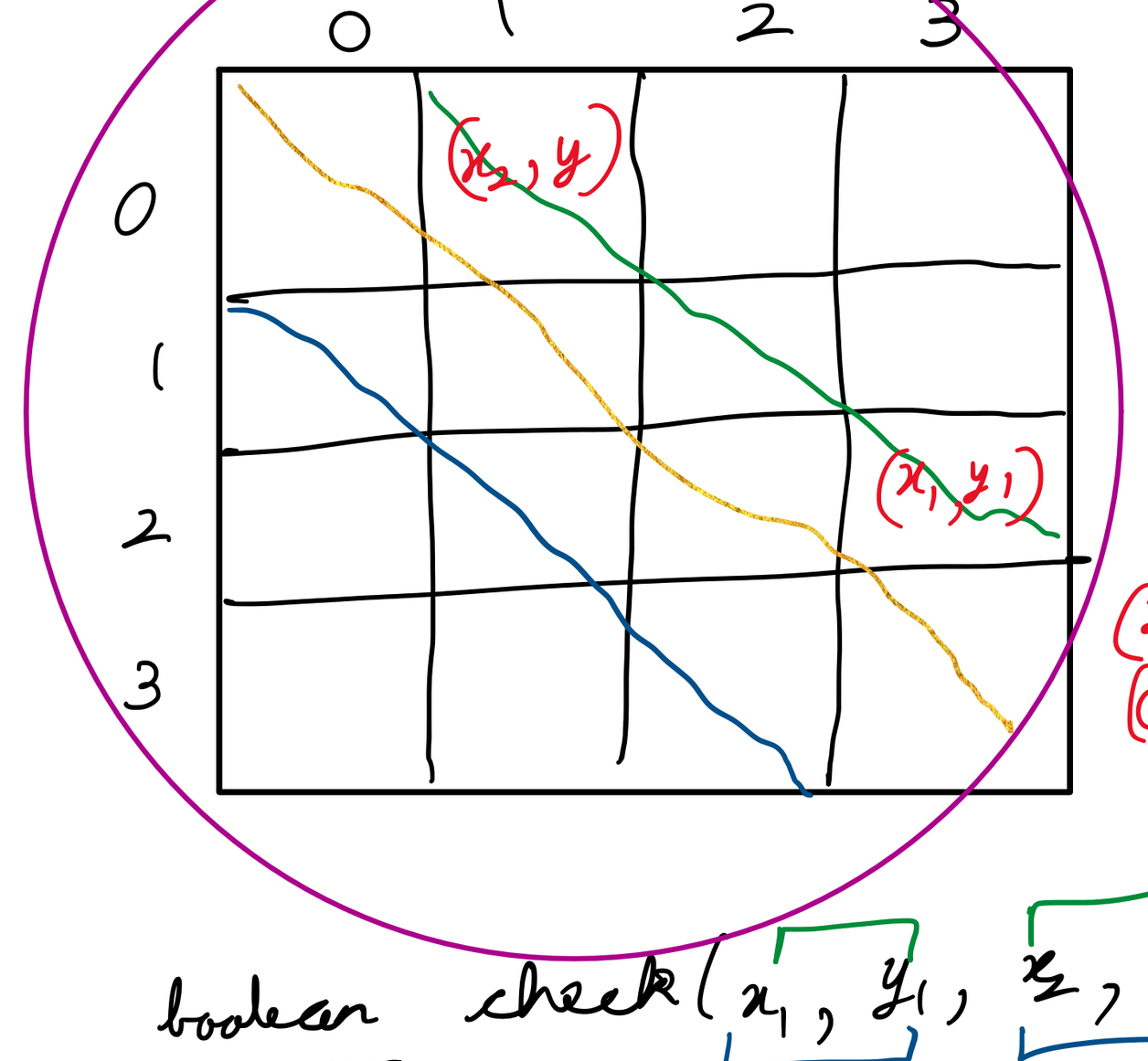


same diagonals

(2,0) (1,1) (0,2) | (3,0) (2,1) (1,2) (0,3) | (3,1) (2,2) (1,3)

x decrement = y increment   (x)(y)   (x-d, y+d)

$(x_1 + y_1) == (x_2 + y_2)$



(1,0) (2,1) (3,2) | (0,0) (1,1) (2,2) (3,3) | (0,1) (1,2) (2,3)

change in x = change in y

$(x_2 - x_1) == (y_2 - y_1)$

(2,3) (0,1) (0-2) = (1-3)

-2   -2

$x_2 - x_1 = y_2 - y_1$

$x_2 - y_2 = x_1 - y_1$

```
boolean check(x1, y1, x2, y2) {
    return (x1 == x2) || (y1 == y2) ||
           (x1 + y1) == (x2 + y2) || (x2 - x1) == (y2 - y1);
}
```

TC = O(1)

// If 2 queens at (x_1, y_1) & (x_2, y_2) can attack each other.

⇒ check if number x is present in A[] . → loop the array → TC = O(N)

Given a N * N chessboard. Can we place N queens s.t they don't attack each other.

Max # queens that can be placed = N

∵ every row/col can have only one queen.

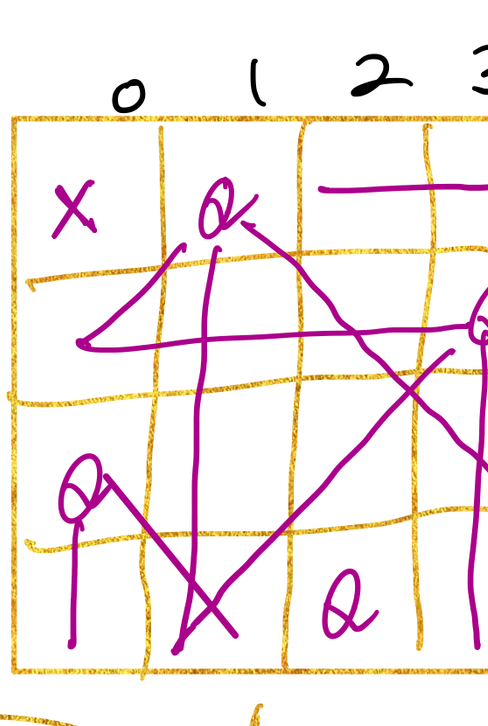
N = 1

Q ✓

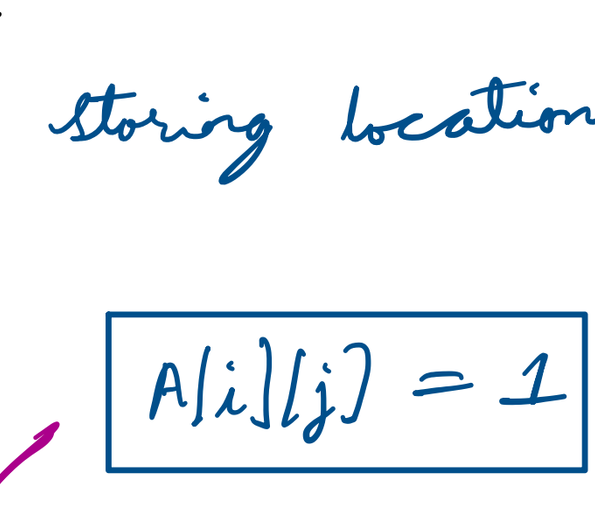
N = 2

X

N = 3



X

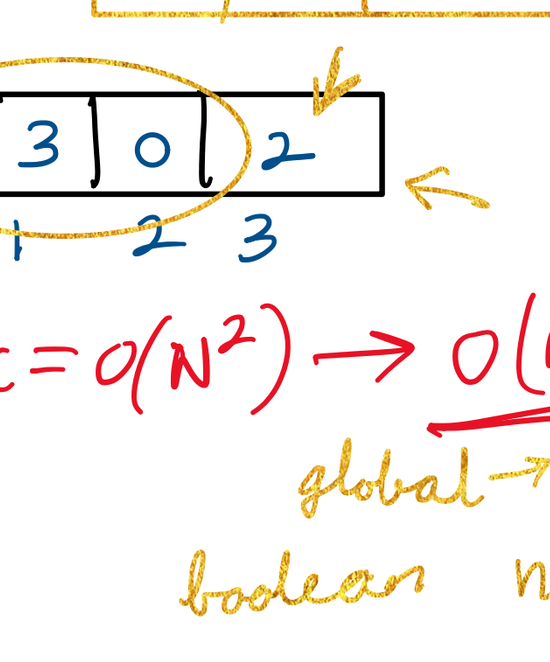


X

not possible

∵ every row should have 1 queen.

N = 4



storing locations of queens placed is important.

A[i][j] = 1 → queen is placed at (i,j)

1	3	0	2

0 1 2 3

column[i] → column location of i^{th} row queen.

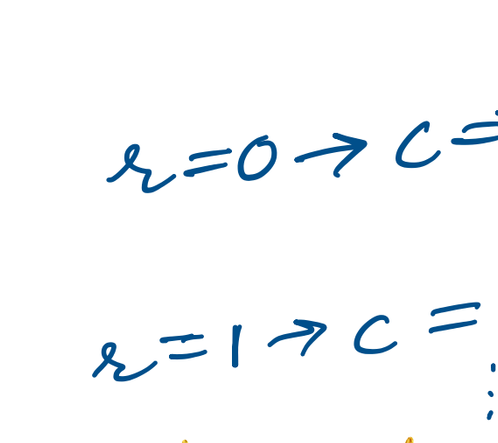
(i, column[i]) → location of queen.

SC = O(N^2) → O(N)

∀i column[i] = -1;

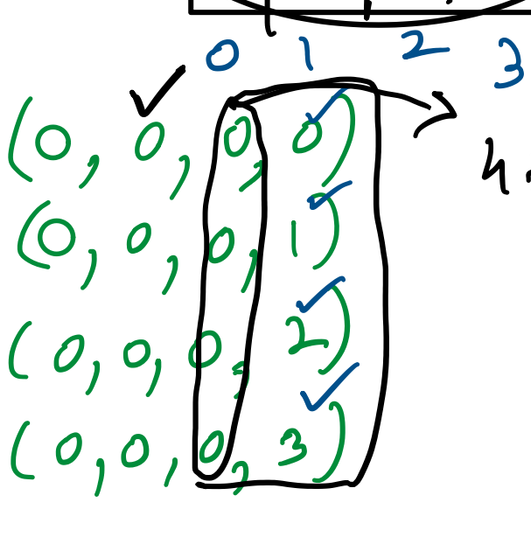
global → N

```
boolean nqueen(r, column[]) {        // r = 4
    if(r == N) return true;
    for(c = 0; c < N; c++) {
        if(isValid(r, c, column)) {
            column[r] = c;
            if(nqueen(r+1, column)) {
                column[r] = -1;
            }
        }
    }
    return false;
}
```



SC = O(N)

TC < O(N^N)



(0,0,0,0) (0,0,0,1) (0,0,0,2) (0,0,0,3)   4*4

r=0 → c=0 1

r=1 → c=3

return true;

r=2 → c=0

```
boolean isValid(r, c, column[]) {    // check if we can place a queen at (r, c);
    for(i = 0; i < r; i++) {         // Queens in same column
        if(column[i] == c) return false;
        // (r, c)  (i, column[i])
        if((r+c) == (i+column[i])) || (r-i) == (c-column[i]))
            return false;
    }
    return true;
}
```

TC = O(r) ≃ O(N)

N <<< N^N

10^{10} → 10000000000

10^{10}

"abc"

a b c