Todays Content

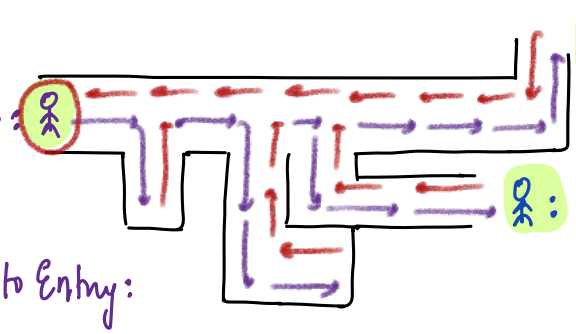
a) Intro
b) N digit
c) N-Queens

Note:

1. All of you will get PDF notes
2. Sessim Recording = Current Scaler Recording.
3. Class Timing = 2hrs 30mins

Back Tracking:

Maze:



: Outside maze ⇄ Solution / Path

: Outside maze ⇄ Solution / Path

Note:

Back to Entry:

Stop.

Idea: Generating all solutions,

1. If we cannot go any further
2. If we reached solution

Go back & choose alternative path.

Above way of generating all solutions is

Backtracking

1. Genarate all solutions n all paths
2. With Recursim.

18) Given N, print all N digit numbers formed only b 1, 2 in increasing order

N = 2 :

1	1
1	2
2	1
2	2

print in IncOrder:

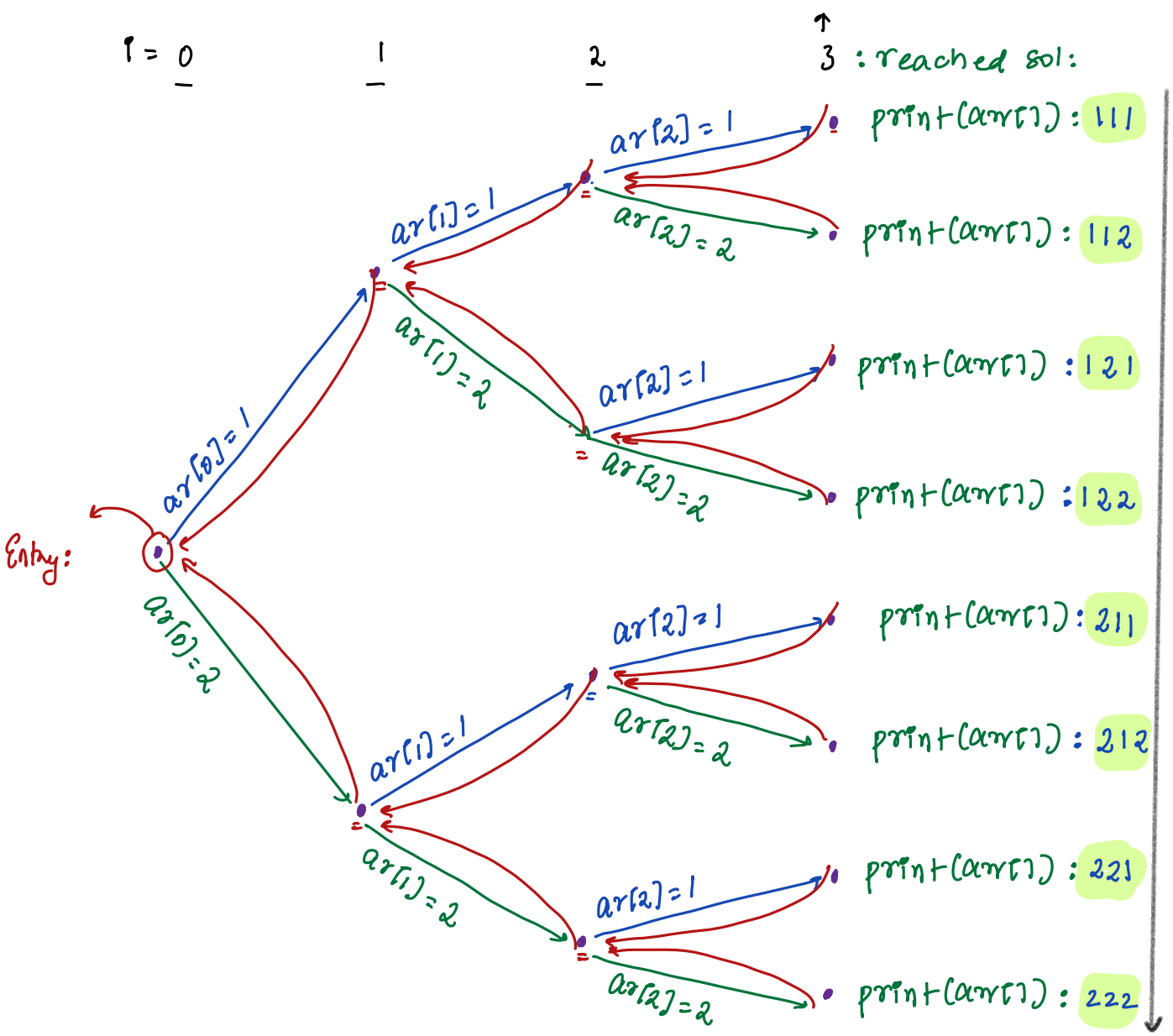
N = 3 :

1	1	1
1	1	2
1	2	1
1	2	2
2	1	1
2	1	2
2	2	1
2	2	2

print in IncOrder:

Tracing N = 3 : ar[3] = { 2 2 2 }, each digit in a index. → all 3 digit numbers

0	1	2	3
2	2	2	



func(): 0 1 2 3 . . . . N-1 N^{th}

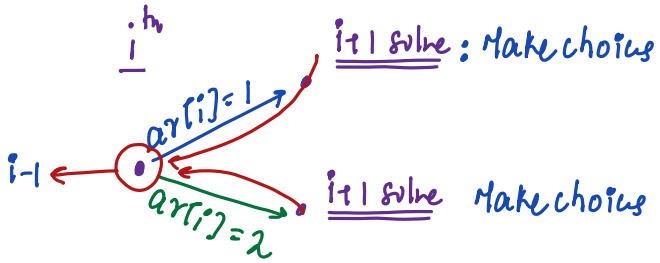
parameters: ar[N] = {To store N digit Number} outside i = {current index}

Choices : 2 choices

Returntype : void.

↳ current index position.

```
void printAll(int ar[], int i, int N){
    if( i==N) { //outside = solution.
        print(ar[])
        return;
    }
    //At i^th index:
    ar[i] = 1;
    printAll(ar, i+1, N)
    ar[i] = 2;
    printAll( ar, i+1, N)
    return;
}
```




```
void main(){
    int N; //Read input to N.
    int ar[N]=0;
    printAll(ar, 0, N);
}
```

roughcode

```
if( reached soln){
    do something
    return
}
Choices can
    function calls
choice done: return
```

```
void solve(int N=2)
{
    int ar[] = new int[2];
    printall(ar, 0, N)
}
```

```
          0  1         1 1
ar[2] = { 2  2 }       1 2
                       2 1
                       2 2
```

```
void printAll(int arr[], int i=0, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1; // ar[0] = 1.
    3. printAll(ar, i+1, N)
    4. ar[i] = 2; // ar[0] = 2
    5. print All (ar, i+1, N)
    6. return
}
```

```
void printAll(int arr[], int i=1, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1; // ar[1] = 1
    3. printAll(ar, i+1, N)
    4. ar[i] = 2; // ar[1] = 2
    5. print All (ar, i+1, N)
    6. return
}
```

```
void printAll(int arr[], int i=1, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1; // ar[1] = 1
    3. printAll(ar, i+1, N)
    4. ar[i] = 2; // ar[1] = 2
    5. print All (ar, i+1, N)
    6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1;
    3. printAll(ar, i+1, N)
    4. ar[i] = 2;
    5. print All (ar, i+1, N)
    6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1;
    3. printAll(ar, i+1, N)
    4. ar[i] = 2;
    5. print All (ar, i+1, N)
    6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1;
    3. printAll(ar, i+1, N)
    4. ar[i] = 2;
    5. print All (ar, i+1, N)
    6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
    1. if(i == N){ print(arr[]) return }
    2. ar[i] = 1;
    3. printAll(ar, i+1, N)
    4. ar[i] = 2;
    5. print All (ar, i+1, N)
    6. return
}
```

Output:

Like it?

a. Goal: Good Engineer: {Understands, code, / optimize / scale / Debug / Bug-Free / }

b. How to we achieve goal?

a. Teach: DSA / Design / Database / Web-development : live Project.

b. How do we Teach:

↳ Eg: Good Company:

c. Doubts: { Unmute & discuss a Doubt session : as long }

d. How to improve : Every class : Assignments.

Doubts in assignments: Live Teaching Assistant : Call.

e. Carer / Profesional / Doubts : Mentor Sessions

f. Cold feet before Interview : Mock Interviews ⇆ { }

g. Placements →

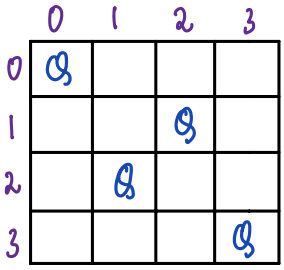
This is what we teach / How we teach & Why we teach / ...

30) Given N*N mat[], print all valid placement of N Queens such that no queen can kill other queen :

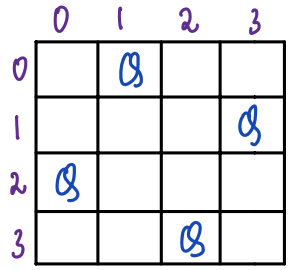
Note: If 2 queen belong to same row/column/diagnol they will kill

Eg:

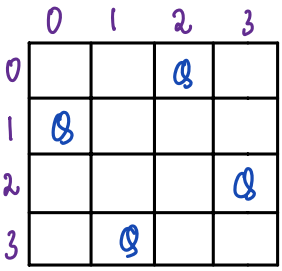
N=4



Invalid



Valid way.

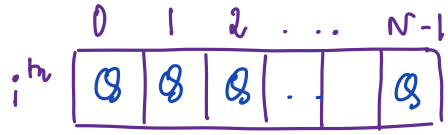


Valid way.

fun():

parameters: mat[N][N], i - row number, N - of Outside matrix

Choices:

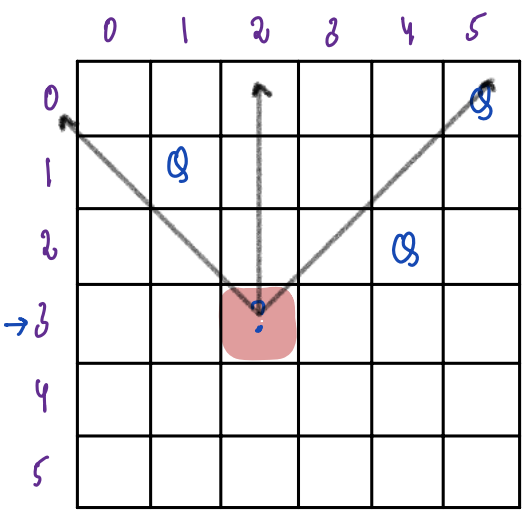


N:choices

for(int j=0; j<N; j++){ // choices }

Returntype: void.

Validfun:



valid: mat(i, j): check at mat(3, 2)

row: Not needed: In a row only 1 Q: placed.

col: (3 2) → (2 2) → (1, 2) → (0, 2) → (-1, 2) } Stop

(r c) → (r-1 c)

lefd: (3 2) → (2 1) → (1 0) → (0 -1) } Stop

(r c) → (r-1 c-1)

rigd: (3 2) → (2 3) → (1 4) → (0 5) : Queen:

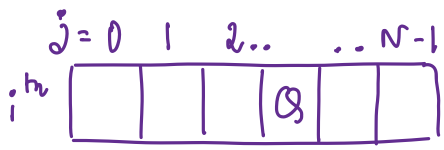
(r c) → (r-1, c+1) Return False.

Note: If No Queen at any of them, I can place Queen: Return True.

Note:

```
boolean valid(int mat[][], int N, int i, int j){

    int r=i, c=j;
    //check col.
    while(r>=0){
        if(mat[r][c]==1){return false}
    }   r=r-1;

    r=i, c=j;
    //check left diagnol;
    while(r>=0 && c>=0){
        if(mat[r][c]==1){return false}
    }   r=r-1; c=c-1;

    r=i, c=j;
    //check rightdiagnol;
    while(r>=0 && c<N){
        if(mat[r][c]==1){return false}
    }   r=r-1; c=c+1;

    return True;
}
```

```
void NQuens(int mat[][], int i, int N){   // Queens to place
    if ( i == N) { // reached sol.
        for(int i=0; i<N; i++){
            for(int j=0; j<N; j++){
                if(mat[i][j]==1){
                        print("Q  ");
                }
                else { print("_  "); }
            }
            println();
        }

        println("\n"); // After each solution take 2 lines
    }   return;
```




```
    // i^th row:
    for(int j=0; j<N; j++){ // choices
        //Place Q at [i][j]   → // If i can place Queen at mat[i][j];
        if( valid(mat, i, j) == true ){
            mat[i][j] = 1; // Place Queen.
            NQueen(mat, i+1, N); // Goto i+1^th row.
        }   mat[i][j] = 0; // Delete Queen
    }
    return;
}

void main(){
    int N; Read Input N.
    int mat[][] = new int[N][N]
}   NQuens(mat, 0, N)
```

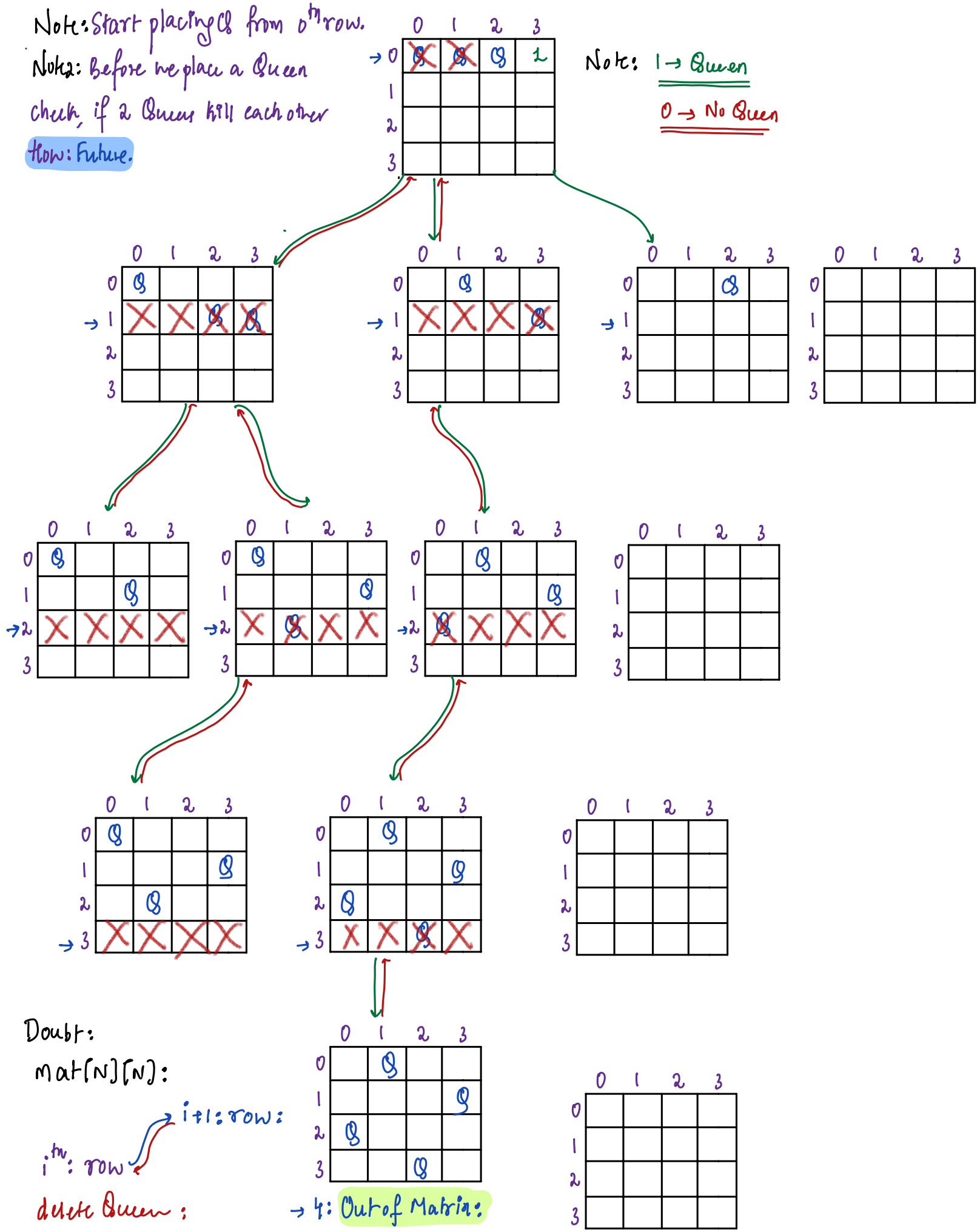
Note: Start placing Q from 0^{th} row.

Note2: Before we place a Queen check, if 2 Queens kill each other

How: Future.

Note: 1 → Queen

0 → No Queen



Doubt:

mat[N][N]:

i^{th}: row → i+1: row:

delete Queen:

→ 4: Out of Matrix:

Sol, Print it return.