Lecture :- Backtracking

Qu:1 Print all subsets of an array.

Example

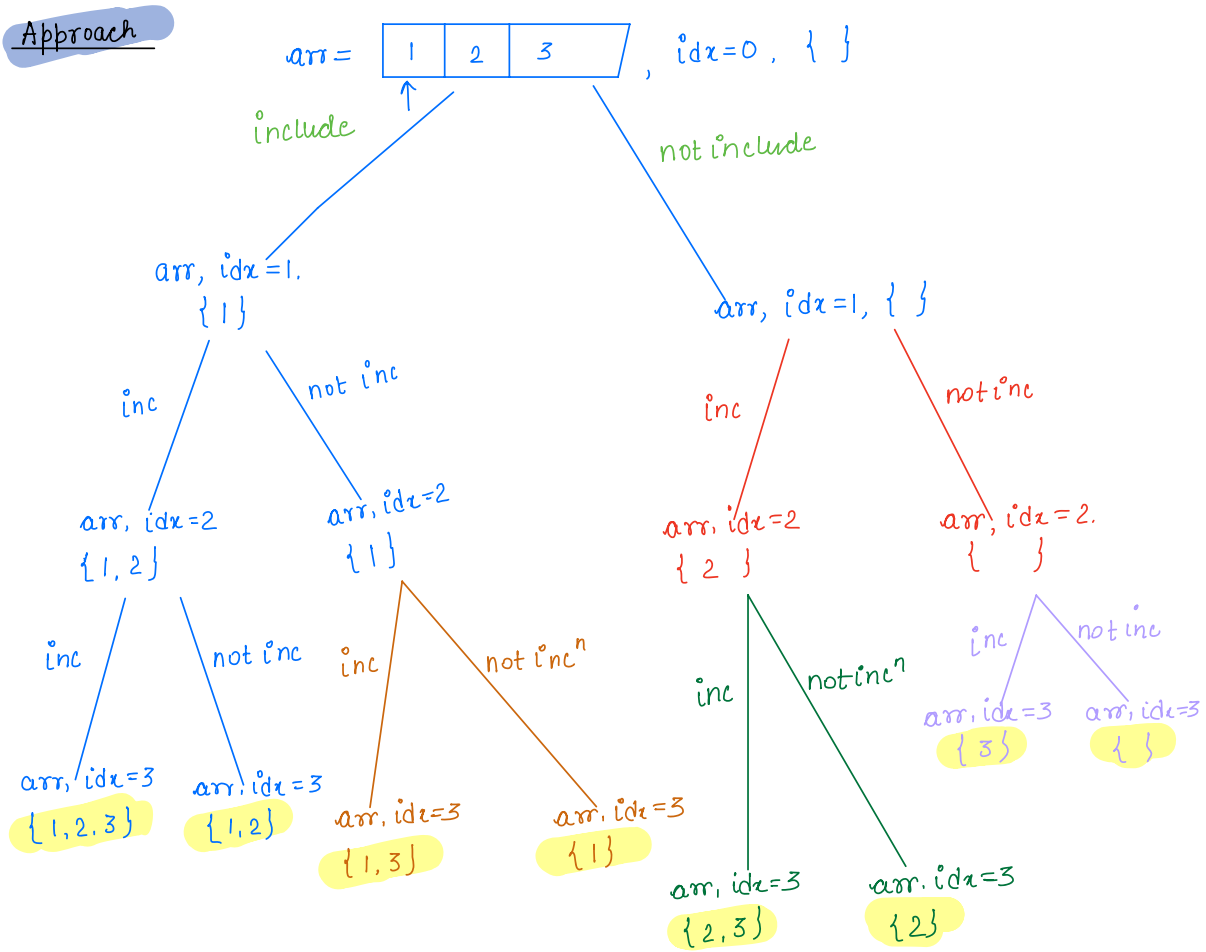
1	2	3

Ans = { }
1
1 2
1 2 3
2
2 3
3
1 3

abcd ⟶ { }
d
c
cd
b
bd
bc
bcd
a
ab
ac
acd
abd
abcd
ad
abc

n ⟶ 2^n subsets.

Approach

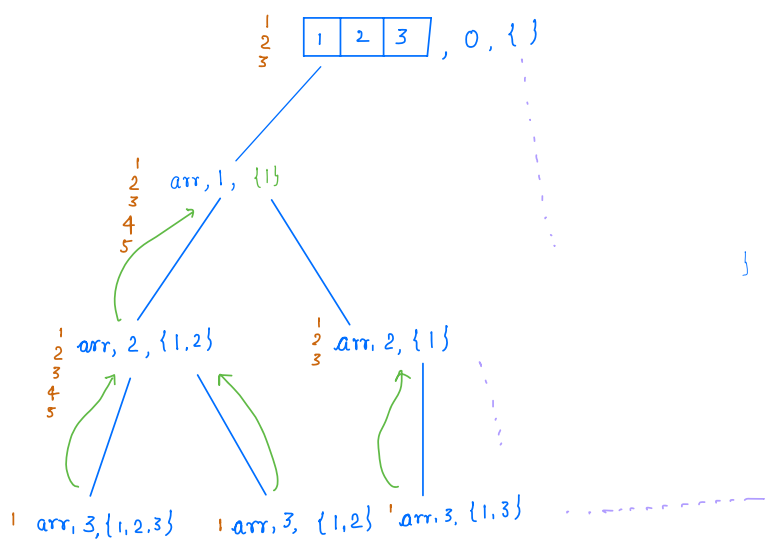

arr = 1 2 3 , idx=0 , { }
include
not include
arr, idx=1, { 1 }
arr, idx=1, { }
inc
not inc
inc
not inc
arr, idx=2 { 1, 2 }
arr, idx=2 { 1 }
arr, idx=2 { 2 }
arr, idx=2, { }
inc
not inc
inc
not inc
inc
not inc
inc
not inc
arr, idx=3 { 1, 2, 3 }
arr, idx=3 { 1, 2 }
arr, idx=3 { 1, 3 }
arr, idx=3 { 1 }
arr, idx=3 { 2, 3 }
arr, idx=3 { 2 }
arr, idx=3 { 3 }
arr, idx=3 { }

Algorithm

```
void subsets ( int[] arr , int idx, List<Integer> res ) {
        if( idx == arr.length) {
                print (res);
                return;
        }
        res.add ( arr[idx] );
        subsets ( arr, idx+1, res );
        res.remove ( res.size()-1);
        subsets ( arr, idx+1, res );
}
```

included

Dry run



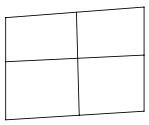

```
Void subsets ( int[] arr , int idx, List<Integer> res) {
1  if( idx == arr.length) {
       print (res);
       return;
   }
2  res.add ( arr[idx] );
3  subsets ( arr, idx+1, res);
4  res. remove ( res.size()-1);
5  subsets ( arr, idx+1, res);
}
```

included

[1 2 3]
[1 2]
[1 3]

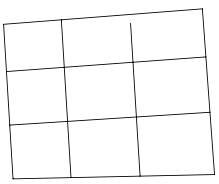
Qu2 Given n*n board. place n queens such that no 2 queens attack each other.

n = 2



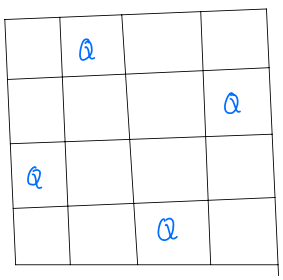
false.

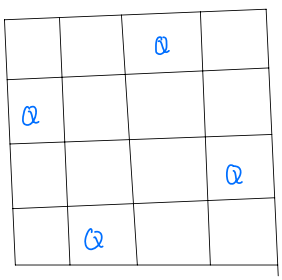
n = 3



false

n = 4

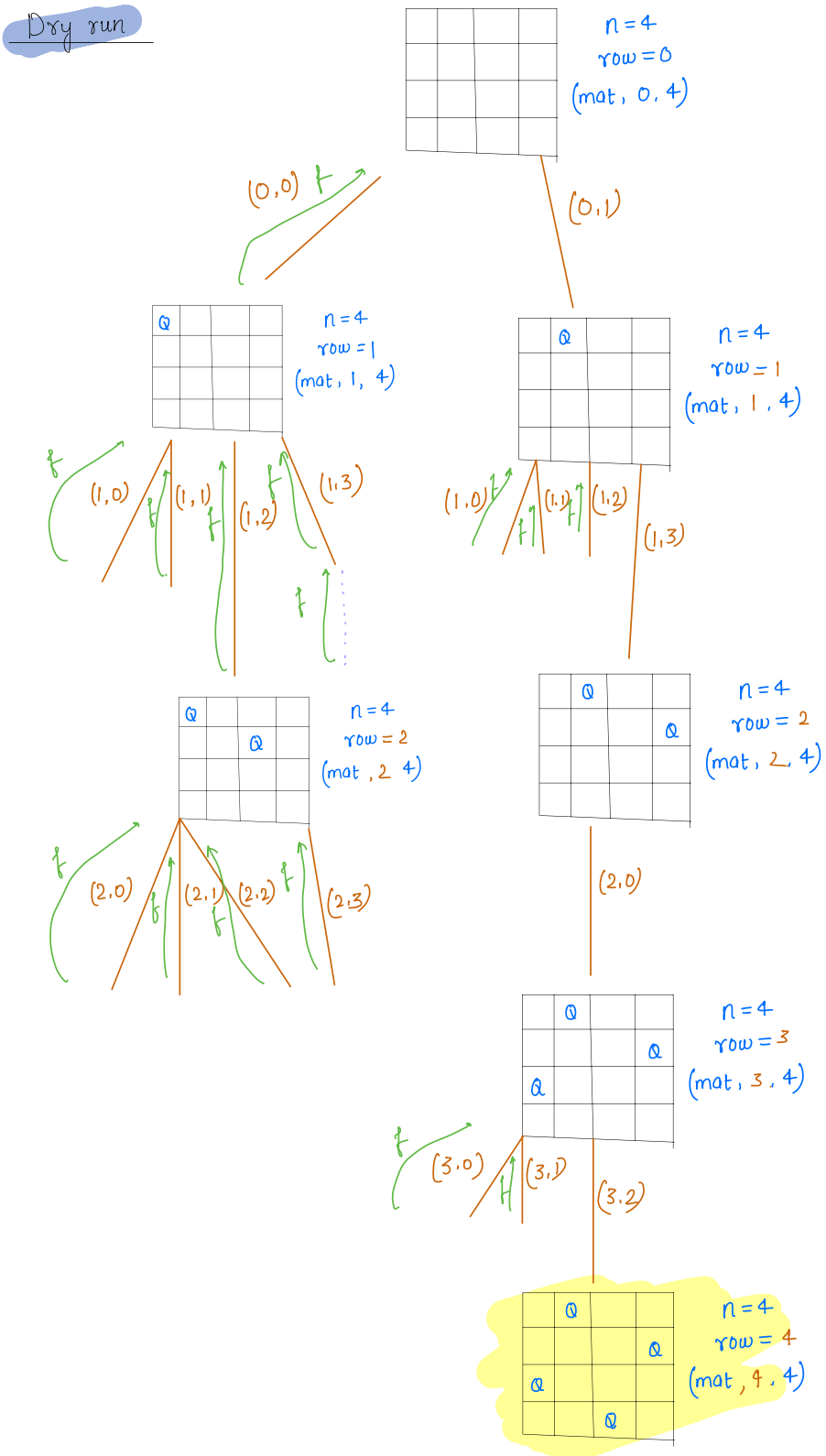




Observation can't have more than 1 Queen in a single row | col | diagonal

Plan placing our Queens row by row

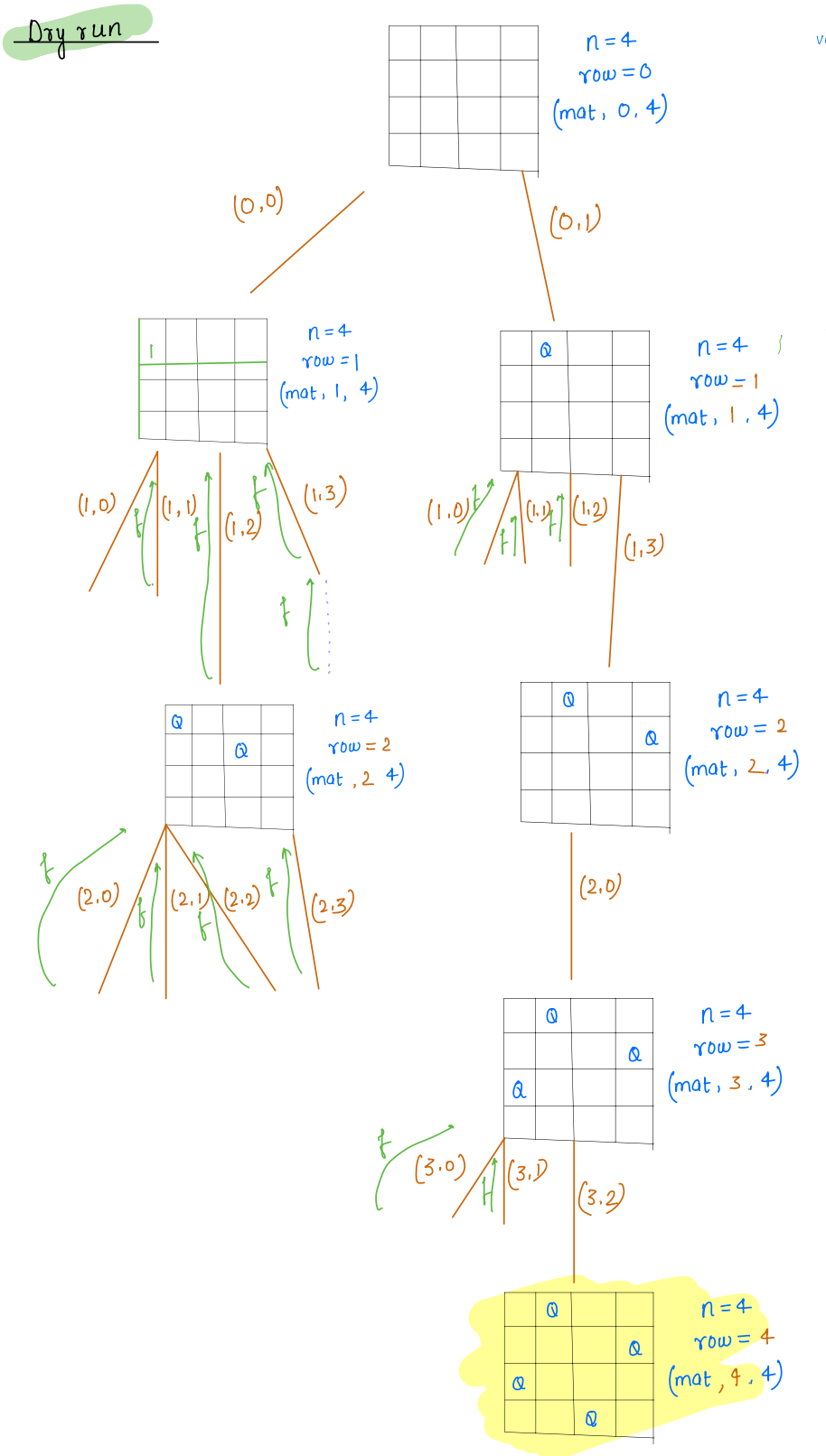
Dry run


n = 4
row = 0
(mat, 0, 4)
(0,0)
(0,1)
Q
n = 4
row = 1
(mat, 1, 4)
(1,0)
(1,1)
(1,2)
(1,3)
n = 4
row = 1
(mat, 1, 4)
(1,0)
(1,1)
(1,2)
(1,3)
Q
n = 4
row = 2
(mat, 2, 4)
(2,0)
(2,1)
(2,2)
(2,3)
n = 4
row = 2
(mat, 2, 4)
(2,0)
n = 4
row = 3
(mat, 3, 4)
(3,0)
(3,1)
(3,2)
n = 4
row = 4
(mat, 4, 4)

Pseudocode

```
void nQueens(mat[][], row, n) {
    1 if(row==n) {
        print(mat);
        return;
      }
    2 for(col=0; col<n; col++) {
        boolean safe = issafe(mat, row, col);
        if(safe==true) {
            mat[row][col] = 1;
            nQueens(mat, row+1, n);
            mat[row][col] = 0;
        }
    }
}
```

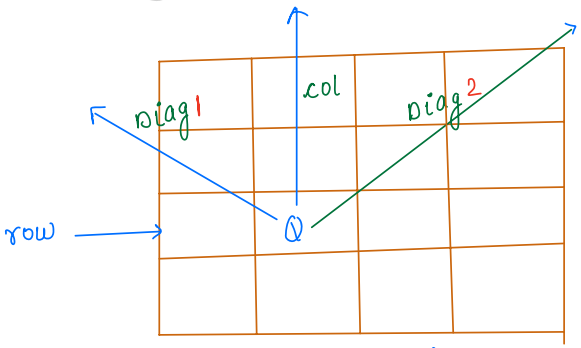
Dry run




```
void nQueens(mat[][], row, n) {
    if(row == n) {
        print(mat);
        return;
    }
    for(col = 0; col < n; col++) {
        boolean safe = issafe(mat, row, col);
        if(safe == true) {
            mat[row][col] = 1;
            nQueens(mat, row+1, n);
            mat[row][col] = 0;
        }
    }
}
```

Implementation issafe() ?

issafe(mat, row, col) {




```
for(i=0; i<row; i++){
    if(mat[i][col]==1){
        return false;
    }
}
```

Diag1

```
i=row-1, j=col-1;
while(i>=0 && j>=0){
    if(mat[i][j]==1){
        return false;
    }
    i--;
    j--;
}
```

Diag2. —— h|w

```
return true;
}
```

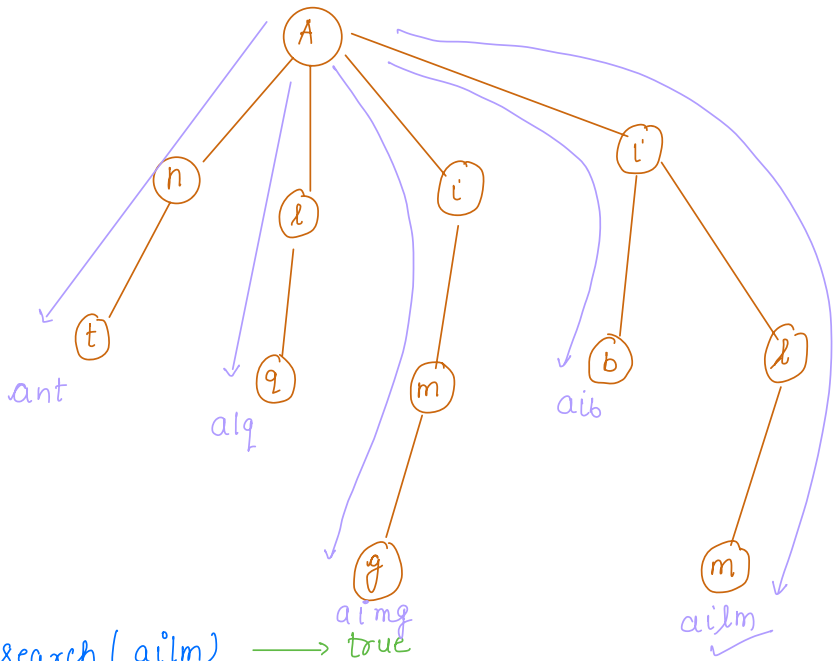
rowCheck ✓

col check [Do]

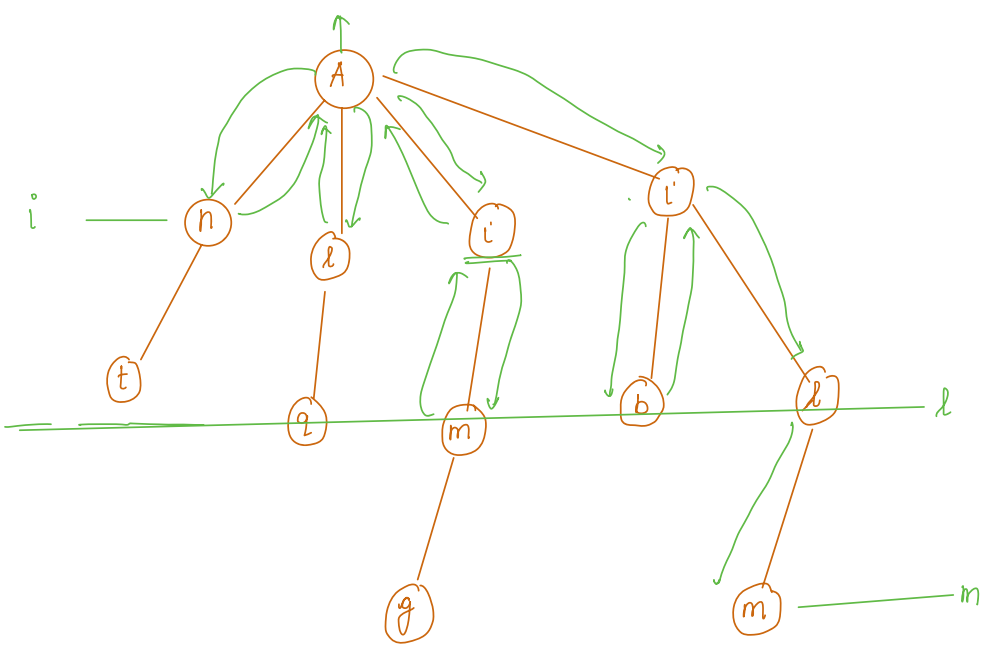
diagonal (h|w)

Backtracking

{ Rec + Certain smartness
Brute force ↗ }



search (ailm) ⟶ true



search (ailm)

Thankyou ☺