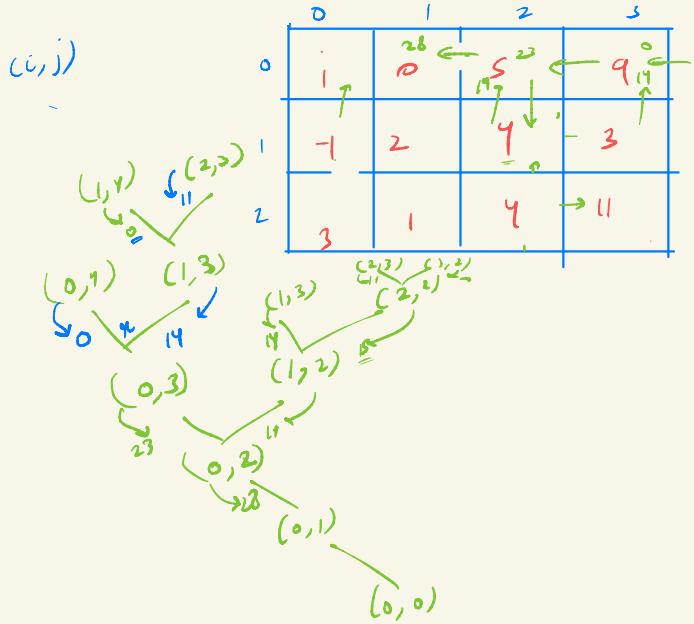
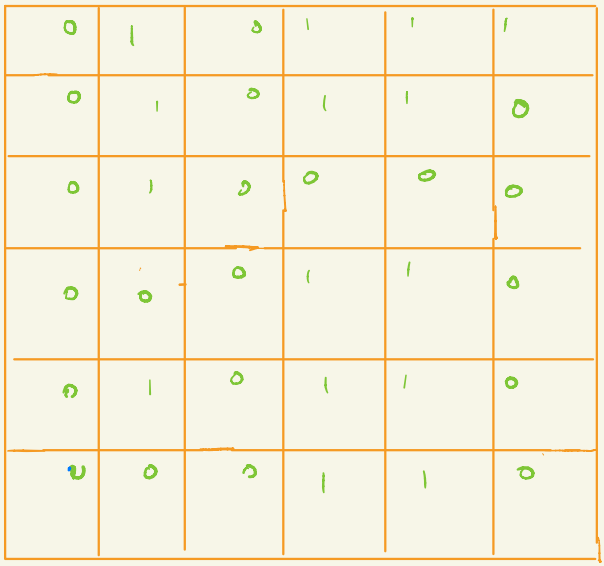


max path sum in matrix from (0,0) to (n-1,m-1)

```java
public static int maxSum(int i, int j, int[][] matrix, int n, int
    if(i>=n || j>=m || i<0 || j<0){
        return 0;
    }

    if(i==n-1 && j==m-1){
        return matrix[i][j];
    }

    // right direction
1)  int a = maxSum(i, j+1, matrix, n, m);

    // downwards direction
2)  int b = maxSum(i+1, j, matrix, n, m);

    int ans = Math.max(a,b) + matrix[i][j];

    return ans;
}
```

---

flood fill

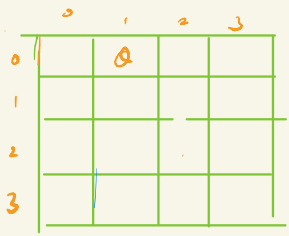


```java
public static void floodFill(int i, int j, int[][] mat, int
    if(i<0 || j<0 || i>=n || j>=m || mat[i][j]==1 || visite
        return;
    }

    if(i==n-1 && j==m-1){
        System.out.println(psf);
        return;
    }

    visited[i][j] = true;
    floodFill(i-1, j, mat, n, m, psf+"t",visited);
    floodFill(i, j-1, mat, n, m, psf+"l",visited);
    floodFill(i+1, j, mat, n, m, psf+"d",visited);
    floodFill(i, j+1, mat, n, m, psf+"r",visited);
    visited[i][j] = false;
}
```

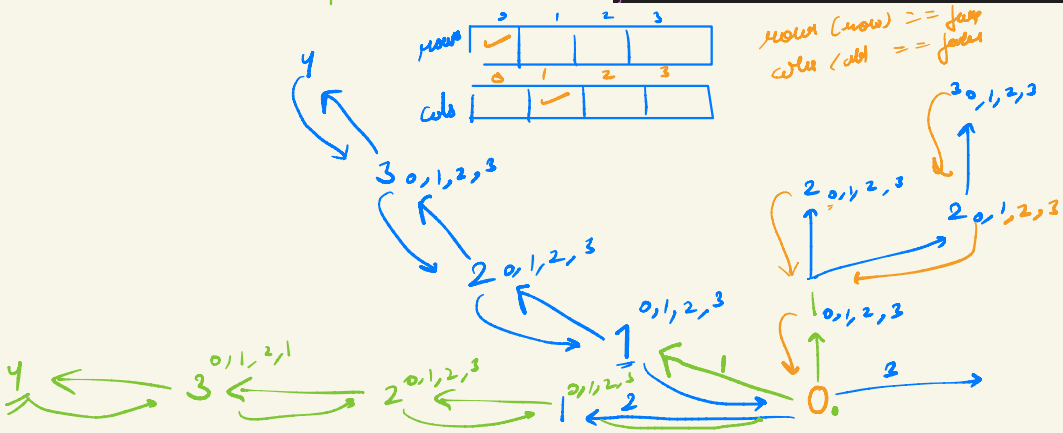
break till 10:15

every row will have exactly 1 queen

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

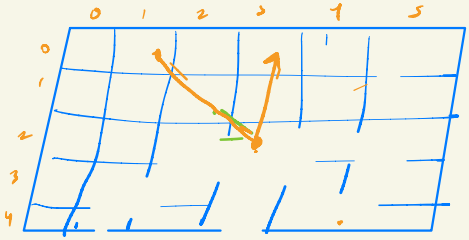


```java
public static void NQueens(int row, int n,boolean[][] board, String asf){
    if(row==n){
        System.out.println(asf);
        return;
    }

    for(int col=0; col<n; col++){
        if(isQueenSafe(row,col,board)==true){
            board[row][col] = true;
            NQueens(row+1, n, board, asf+"( "+row+" , "+col+" )");
            board[row][col] = false;
        }
    }
}
```



int[][] dirs = {{-1,-1},{-1,0},{-1,1},{0,1},{1,1},{1,0},{-1,-1},{0,-1}};

⇒ {{-2,-2},{-2,0},{-2,2},{0,2},{2,2},{2,0},{2,-2},{0,-2},
{-3,-3},{-3,0},{-3,3},{0,3},{3,3},{3,0},{3,-3},{0,-3}}



row=2
col=3

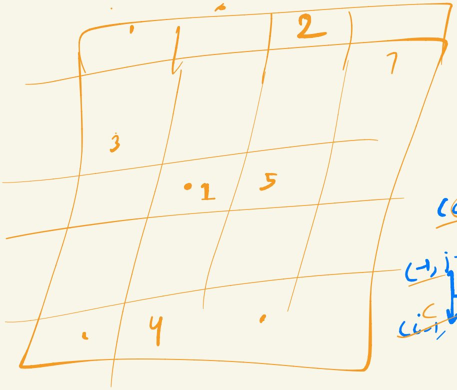
dir = i, j

```java
public static boolean isQueenSafe(int row, int col, boolean[][] board){
    int n = board.length;

    int[][] dirs = {{-1,-1},{-1,0},{-1,1},{0,1},{1,1},{1,0},{-1,-1},{0,-1}};

    for(int rad=1; rad<n; rad++){
        for(int[] dir : dirs){
            int i = row + rad * dir[0];
            int j = col + rad * dir[1];

            if(i>=0 && j>=0 && i<n && j<n && board[i][j]==true) return false;
        }
    }

    return true;
}
```

rad=2 dir={-1,0}

i = 2 + 2*(-1) ⇒ 2-2 = 0

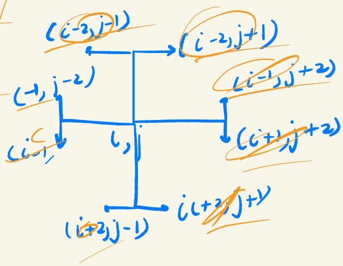
j = 3 + 2*(0) ⇒ 3+0 = 3



8×8

if (move == 25)
print 2d array
)

chess[r][c] = move;



move+1

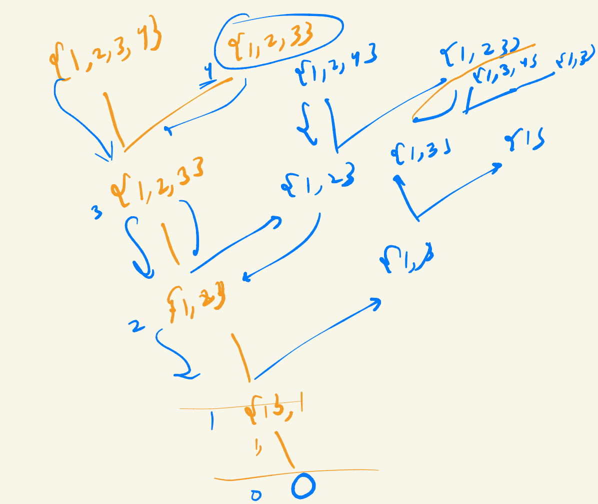
chess[r][c] = 0;

{{1,2,3}}

arr = {1, 2, 3, 4} (indices 0, 1, 2, 3)

tar = 6

"abcdef"




```java
public static void solve(int[] arr, int tar, int idx, int csum,
    if(csum>tar) return;
    if(idx == arr.length){
        if(tar==csum){
            ans.add(new ArrayList<>(curr));
        }
        return;
    }

    // include
    curr.add(arr[idx]);
    solve(arr,tar,idx+1,csum+arr[idx],curr,ans);
    curr.remove(curr.size()-1);

    // not include
    solve(arr,tar,idx+1,csum,curr,ans);
}
```

integer

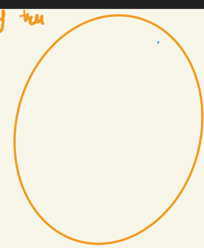
→ ans.add(curr)

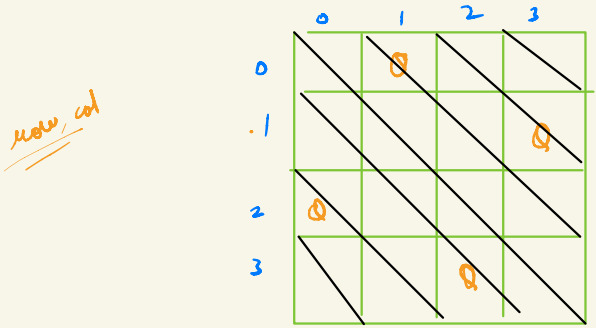
curr ↳ address

ans ↳ address

TC = (number of calls)$^{\text{height of tree}}$

2^n






```java
public static void NQueens(int row, int n, boolean[] row_visited, boolean[] col_visited
    if(row==n){
        System.out.println(asf);
        return;
    }

    for(int col=0; col<n; col++){
        if(row_visited[row] == false && col_visited[col] == false){
            row_visited[row] = true;
            col_visited[col] = true;
            NQueens(row+1, n, row_visited,col_visited, asf+"("+row+" , "+col+")");
            row_visited[row] = false;
            col_visited[col] = false;
        }
    }
}
```

	0	1	2	3	4	5	6	7	8
0	5	3			7				
1	6			1	9	5			
2		9	8					6	
3	8				6				3
4	4			8		3			1
5	7				2				6
6		6					2	8	
7				4	1	9			5
8					8			7	9

8,3
(8/3)x3 ⇒ 2x3 ⇒ 6
(3/3)x3 ⇒ 1x3 ⇒ 3

row = 4 → sr = 3, sc = 0
col = 1
num = 8

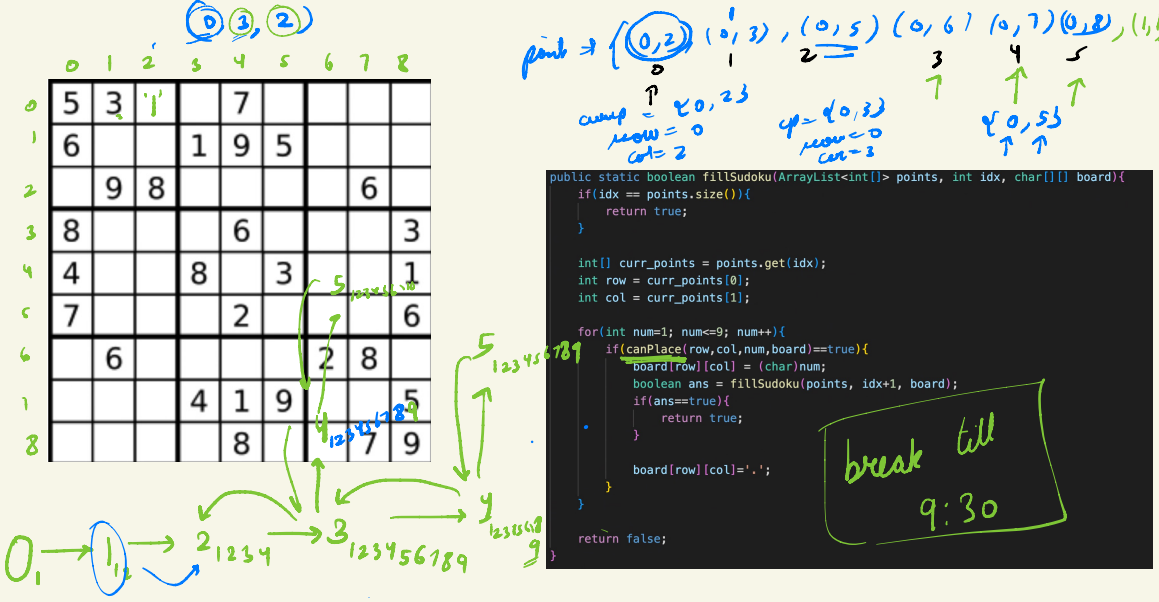
```java
public static boolean canPlace(int row, int col, int num, char[][] board){
    // check row
    for(int i=row,j=0; j<9; j++){
        if(board[i][j]-'0'==num){
            return false;
        }
    }

    // check col
    for(int i=0, j=col; i<9; i++){
        if(board[i][j]-'0'==num){
            return false;
        }
    }

    // check subGrid
}
```

5,8
(5/3)x3 ⇒ 1x3 = 3
(8/3)x3 ⇒ 6

(row/3)x3 ⇒ (4/3)x3 ⇒ 1x3 ⇒ 3
(col/3)x3 ⇒ (1/3)x3 ⇒ 0x3 ⇒ 0

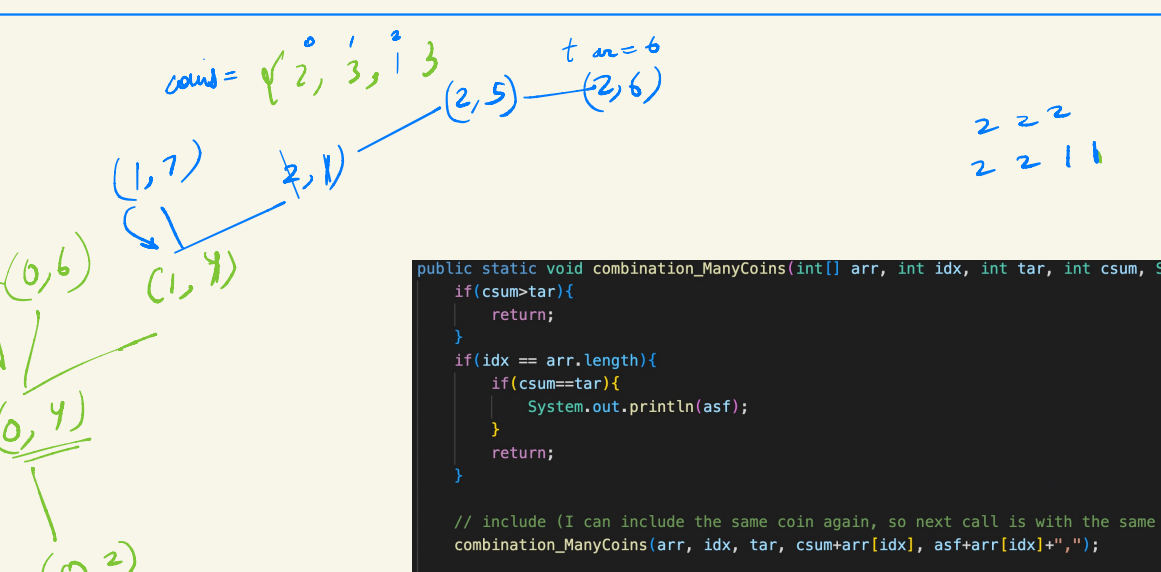
sr = 3
sc = 3

i = sr
j = sc
j < sc+3
i < sr+3

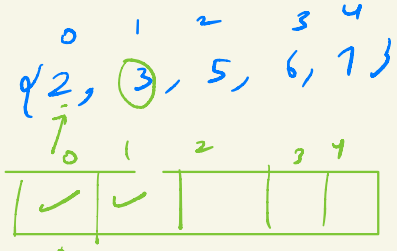


```
public static boolean fillSudoku(ArrayList<int[]> points, int idx, char[][] board){
    if(idx == points.size()){
        return true;
    }

    int[] curr_points = points.get(idx);
    int row = curr_points[0];
    int col = curr_points[1];

    for(int num=1; num<=9; num++){
        if(canPlace(row,col,num,board)==true){
            board[row][col] = (char)num;
            boolean ans = fillSudoku(points, idx+1, board);
            if(ans==true){
                return true;
            }

            board[row][col]='.';
        }
    }

    return false;
}
```

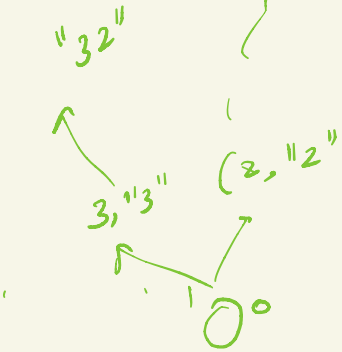
break till 9:30



```
public static void combination_ManyCoins(int[] arr, int idx, int tar, int csum, Str
    if(csum>tar){
        return;
    }
    if(idx == arr.length){
        if(csum==tar){
            System.out.println(asf);
        }
        return;
    }

    // include (I can include the same coin again, so next call is with the same id
    combination_ManyCoins(arr, idx, tar, csum+arr[idx], asf+arr[idx]+",");

    // not include
    combination_ManyCoins(arr, idx+1, tar, csum, asf);
}
```



valid sudoku
→ permutation 2
dry run every thing




```java
public static void getPermutations(int[] coins, int tar, int csum,boolean[] vis, 
    if(csum > tar){
        return;
    }
    if(tar == csum){
        System.out.println(asf);
        return;
    }

    for(int j=0; j<coins.length; j++){
        if(vis[j]==false){
            vis[j] = true; // using this coin
            getPermutations(coins, tar, csum+coins[j], vis, asf+coins[j]+" ");
            vis[j] = false;
        }
    }
}
```