

* SORTING

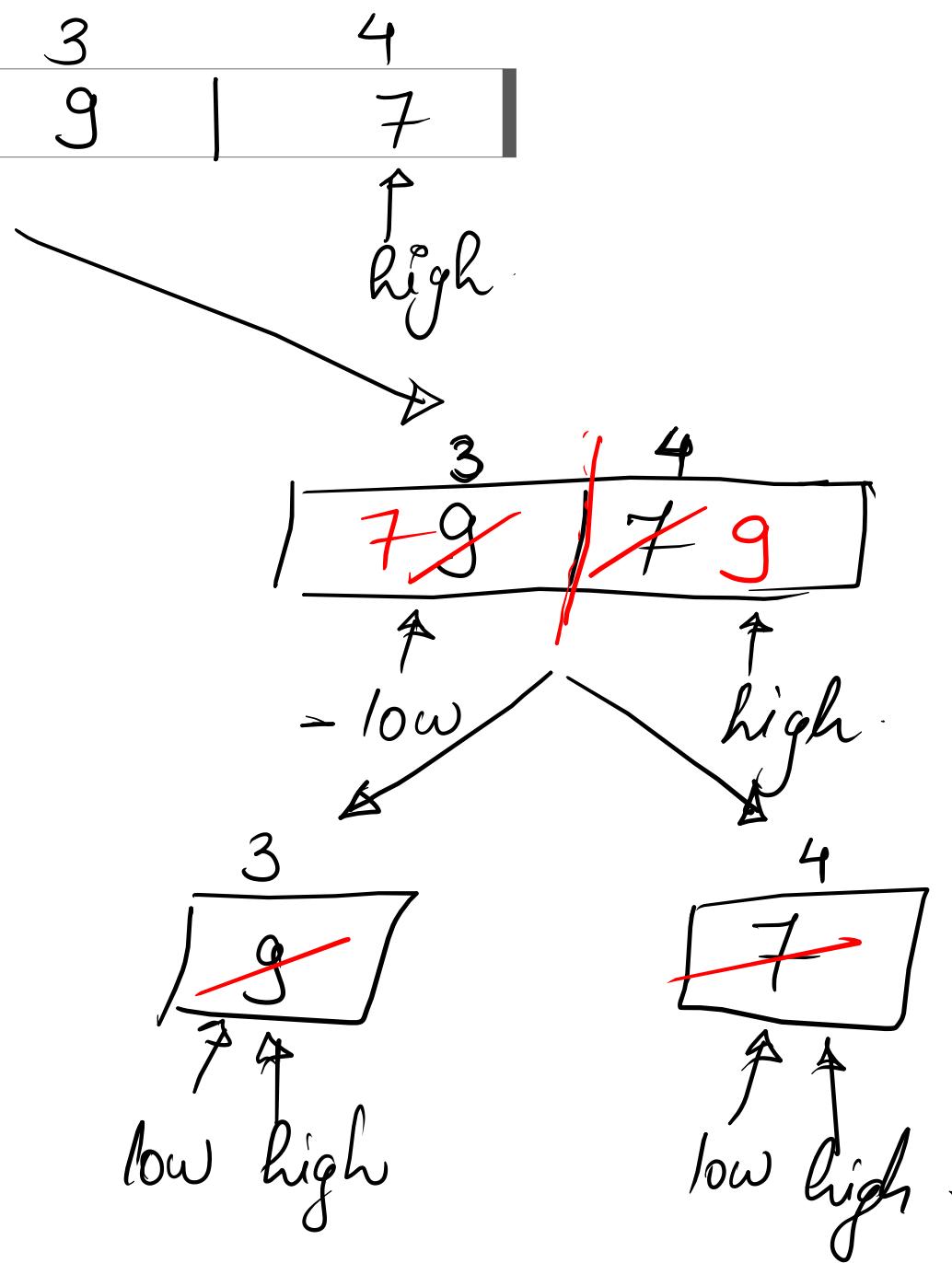
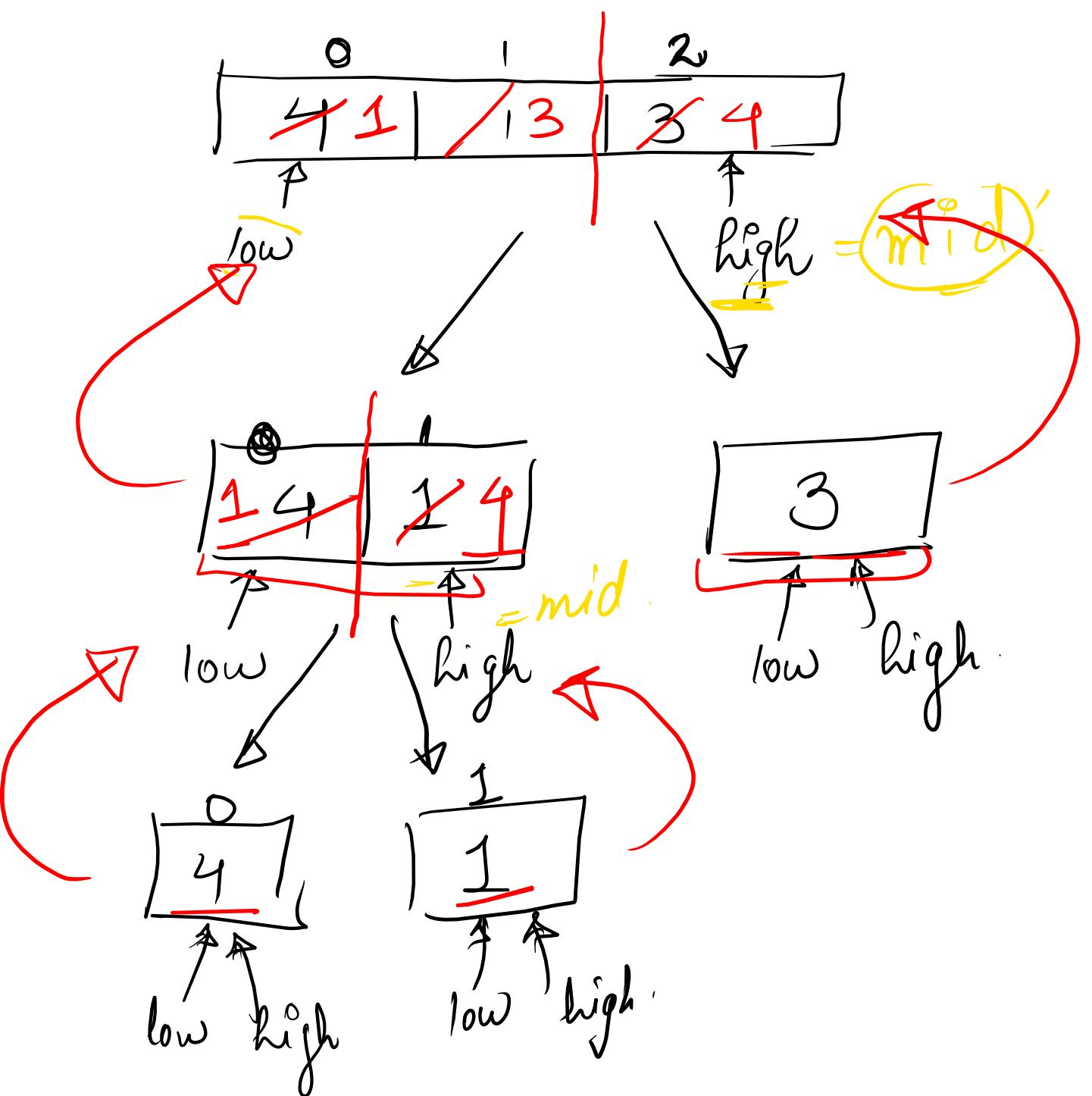
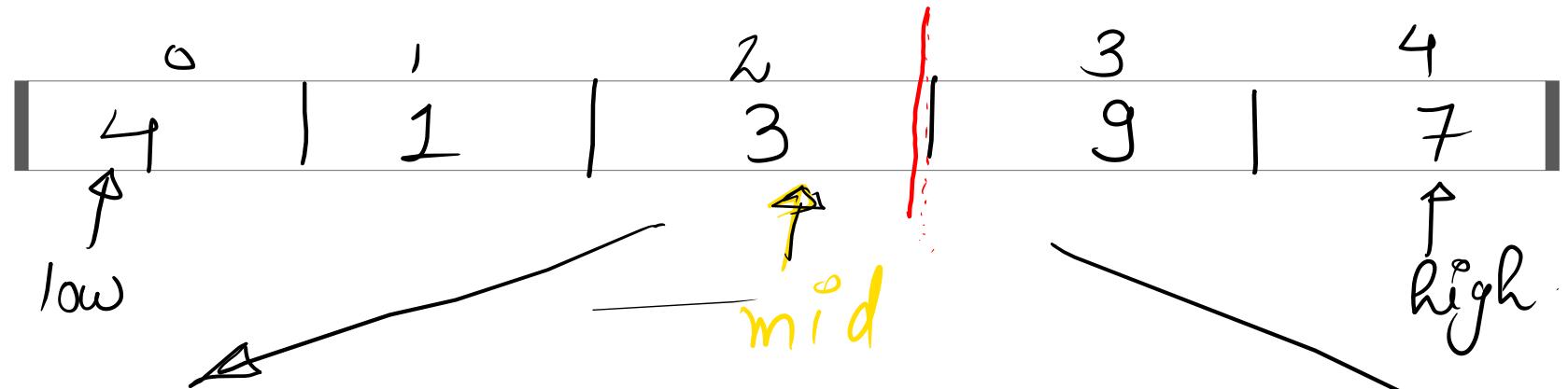
	<u>TC</u>	<u>SC</u>
1.) Bubble Sort	$O(N^2)$	—
2.) Insertion Sort	—	—
3.) Selection Sort	—	—
4.) Merge Sort	—	—
5.) Quick Sort.	—	—

* Merge Sort

↳ Divide and Conquer

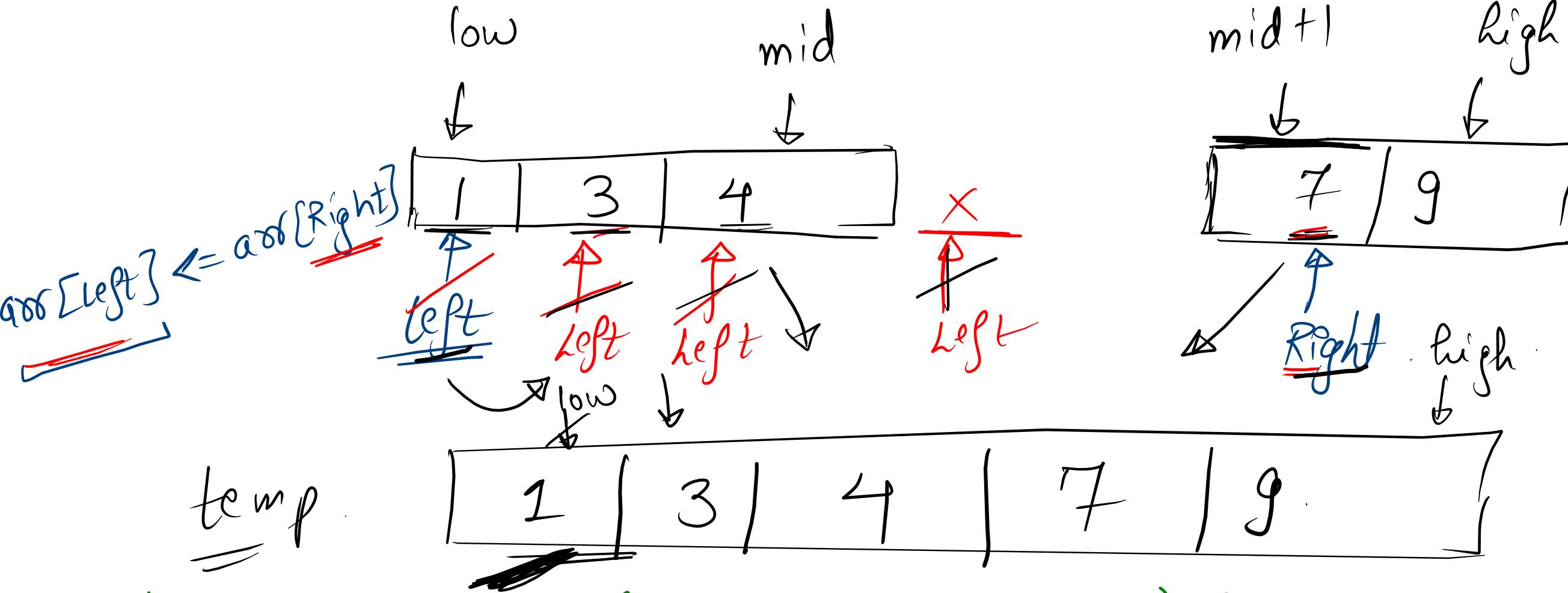
↳

$$mid = \frac{l+h}{2}$$



$$\frac{3+4}{2} \Rightarrow 3$$

```
[ f(arr ; low , high) {  
    if( low < high ) {  
        → mid = (low+high)/2 . ✓✓  
        → f(arr, low, mid) ; Left arrg pass  
        → f(arr, mid+1, high); Right arrg pass  
    }  
}
```



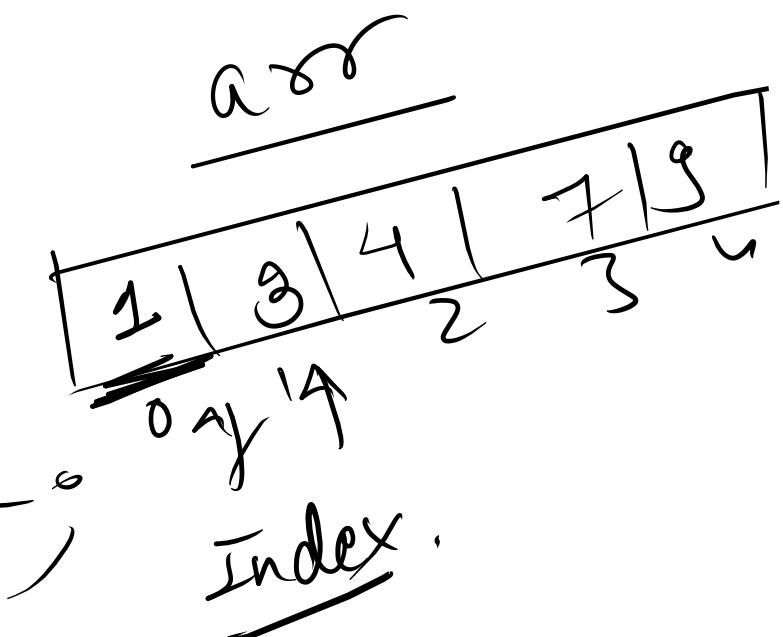
while ($\text{low} \leq \text{mid} \& \& \text{right} \leq \text{high}$) {

} if ($\text{arr}[\text{left}] \leq \text{arr}[\text{right}]$)

temp.add (arr[left]); left++;

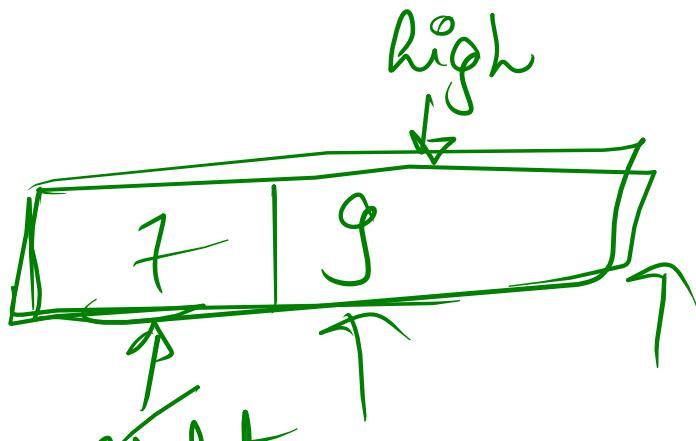
} else

temp.add (arr[right]); right++;



Case -2

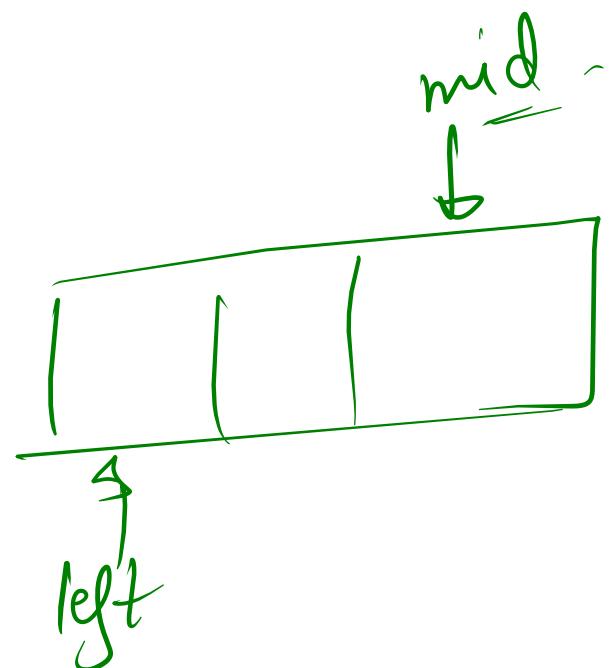
X



// while (Right <= high) temp.add (arr[Right]) Right++;

Case -3

X



while (left <= mid) temp.add (arr[left]) left++

* Ex - $[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

use merge sort.

Merge Sort

Time Complexity : $O(N \log N)$

Space Complexity : $O(N)$

```
static void mergeSort(int[] arr,int low,int high){  
    if(low >= high) return ;  
    int mid =(low + high) / 2 ;  
    mergeSort(arr,low,mid); // this is my left array  
    mergeSort(arr,mid+1,high); // this is my right array  
  
    merge(arr,low,mid,high);  
}  
static void merge(int[]arr , int low , int mid , int high){  
    ArrayList<Integer>temp = new ArrayList<>();  
    int left = low ;  
    int right = mid+1;  
  
    while(left <= mid && right <= high){  
        if(arr[left] <= arr[right]){  
            temp.add(arr[left]);  
            left++;  
        }  
        else{  
            temp.add(arr[right]);  
            right++;  
        }  
    }  
  
    //if elements of left arrays are left(bachai hue hai)  
    while(left <= mid){  
        temp.add(arr[left]);  
        left++;  
    }  
  
    //if elements of right arrays are left(bachai hue hai)  
    while(right <= high){  
        temp.add(arr[right]);  
        right++;  
    }  
  
    int index = 0 ;  
    while(low <= high){  
        arr[low] = temp.get(index);  
        low++;  
        index++;  
    }  
}
```



QUICK SORT

.

Approach:

Now, let's understand how we are going to implement the logic of the above steps. In the intuition, we have seen that the given array should be broken down into subarrays. But while implementing, we are not going to break down and create any new arrays. Instead, we will specify the range of the subarrays using two indices or pointers(i.e. **low** pointer and **high** pointer) each time while breaking down the array.

The algorithm steps are the following for the **quickSort()** function:

1. Initially, the **low** points to the first index and the **high** points to the last index(as the range is n i.e. the size of the array).
2. After that, we will get the index(*where the pivot should be placed after sorting*) while shifting the smaller elements on the left and the larger ones on the right using a **partition()** function discussed later.

Now, this index can be called the **partition index** as it creates a partition between the left and the right unsorted subarrays.

3. After placing the pivot in the partition index(within the **partition()** function specified), we need to call the function **quickSort()** for the left and the right subarray recursively. So, **the range of the left subarray will be [low to (partition index - 1)]** and **the range of the right subarray will be [(partition index + 1) to high]**.

Quick Sort

→ Pivot ↗ sorted place

7, 2, 1, 10 → 1, 2, 7, 10

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

Pivot

2 | 1 3

Pivot

Smaller

1 2 3

Smaller

Greater

sorted

4

6

5

7

9

Pivot

Greater

5

Smaller

6

9, 7

Greater

Pivot

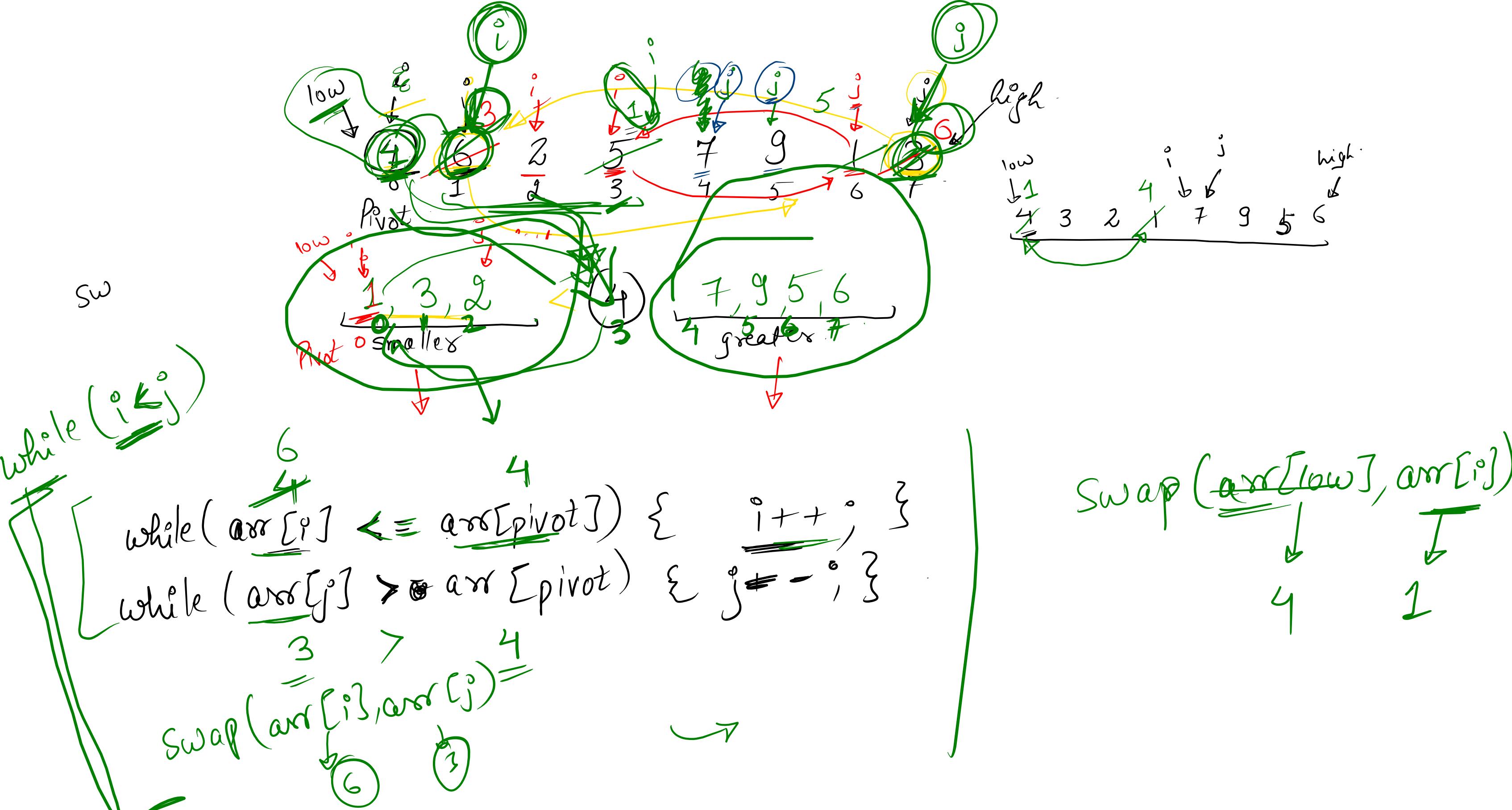
7 9

S

G

Chosing first element as a pivot

1. pick a pivot and place it in its correct place in the sorted array
2. smaller on the left and greater on the right



Quick Sort

Time Complexity : $\underline{\underline{O(N \log N)}}$

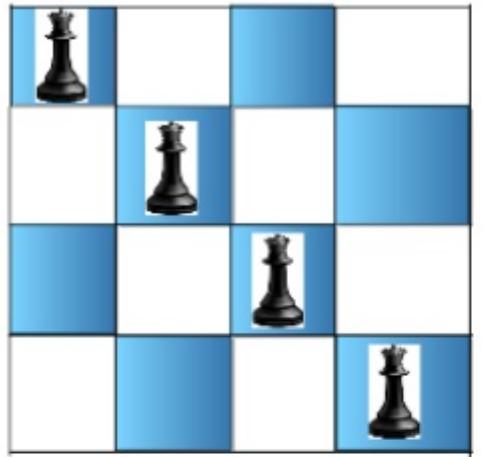
Space Complexity : $\underline{\underline{O(1)}}$

Q Why we prefer Quick Sort
Over Merge Sort ?

H/w

```
static void quickSort(int[] arr){  
    int n=arr.length;  
    qs(arr,0,n-1);  
}  
public static void qs(int[] arr,int low,int high){  
    if(low<high){  
        int partitionIndex=partition(arr,low,high);  
        qs(arr,low,partitionIndex-1);  
        qs(arr,partitionIndex+1,high);  
    }  
}  
public static int partition(int[] arr,int low,int high){  
    int pivot=arr[low];  
    int i=low;  
    int j=high;  
  
    while(i<j){  
        while(arr[i]<=pivot && i<=high-1){  
            i++;  
        }  
  
        while(arr[j]>pivot && j>=low+1){  
            j--;  
        }  
        if(i<j){  
            swap(arr,i,j);  
        }  
    }  
    swap(arr,low,j);  
    return j;  
}  
public static void swap(int[] arr,int i, int j){  
    int temp=arr[i];  
    arr[i]=arr[j];  
    arr[j]=temp;  
}
```

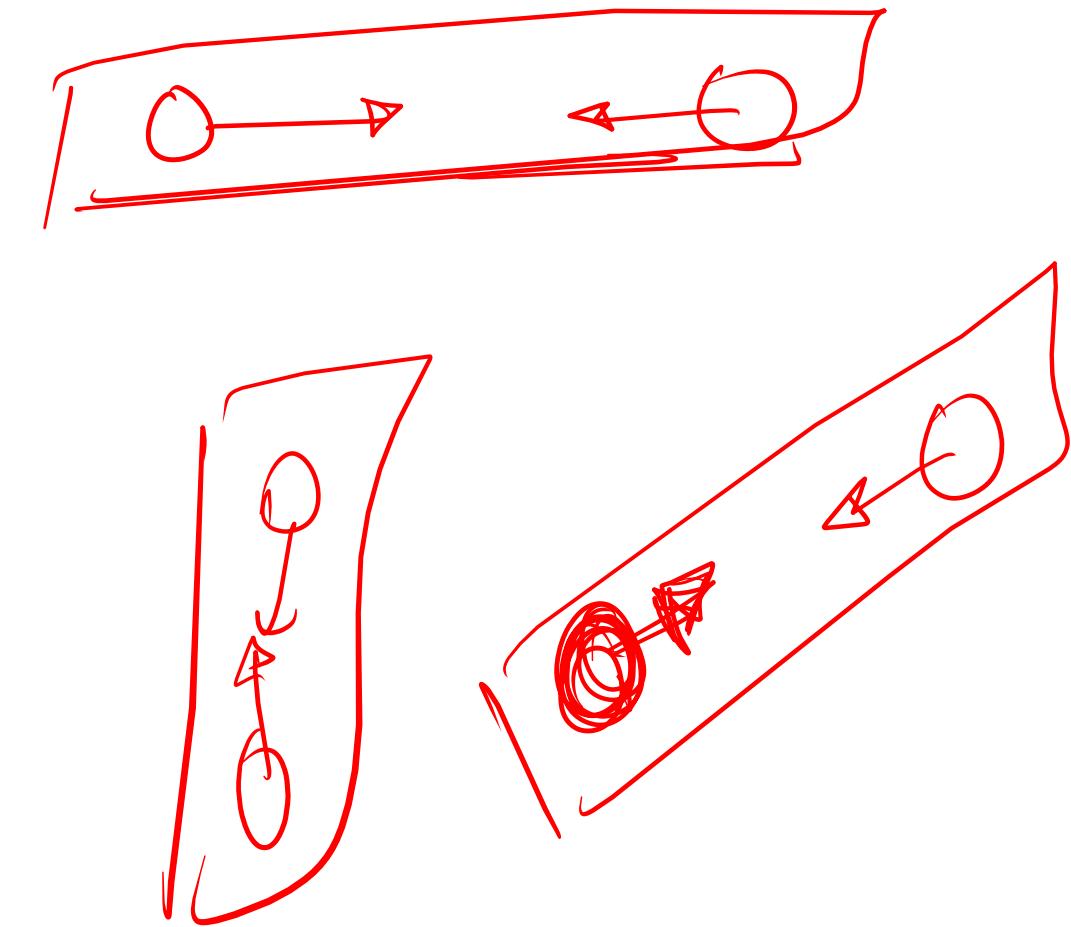
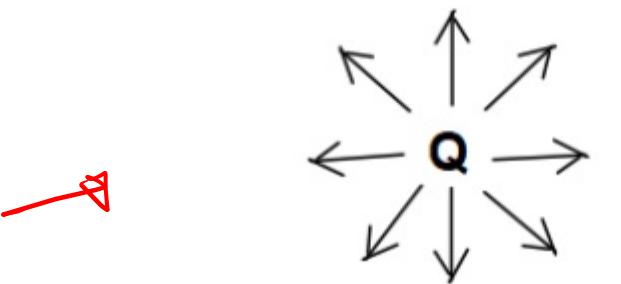
N - Queen



Rules for n-Queen in chessboard

1. Every row should have one Queen ✓
2. Every column should have one Queen ✓
3. No two queens can attack each other ✓

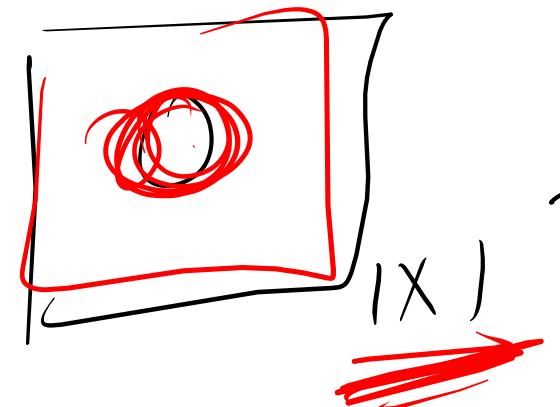
Queen attack can take place in following way



$1 \times 1 \Rightarrow 1$

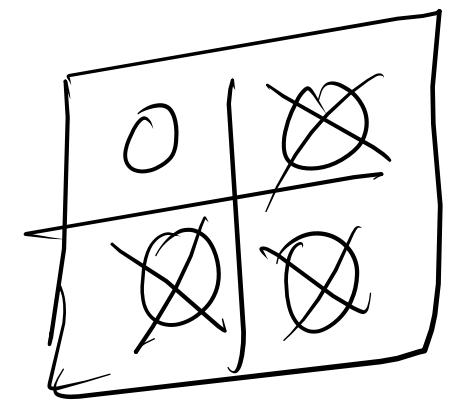
$3 \times 3 \Rightarrow 0$

$4 \times 4 \Rightarrow 2$



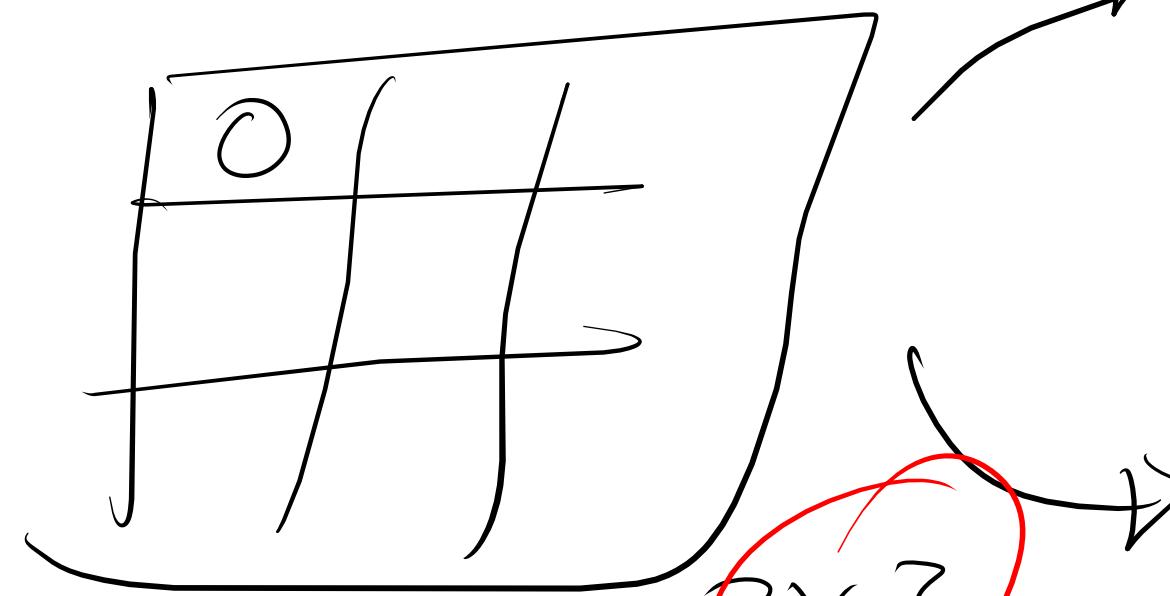
1×1

$\text{cnt} = 1$



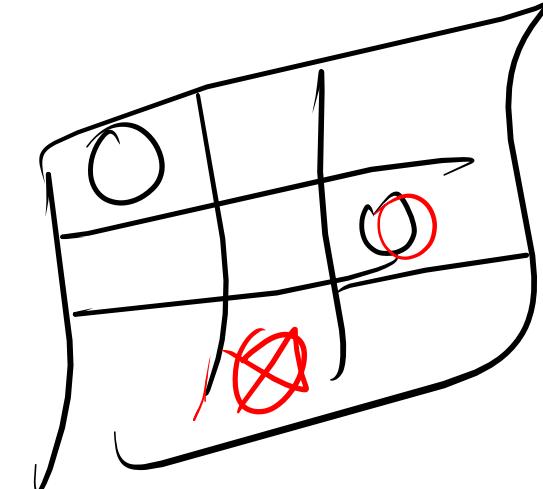
2×2

1

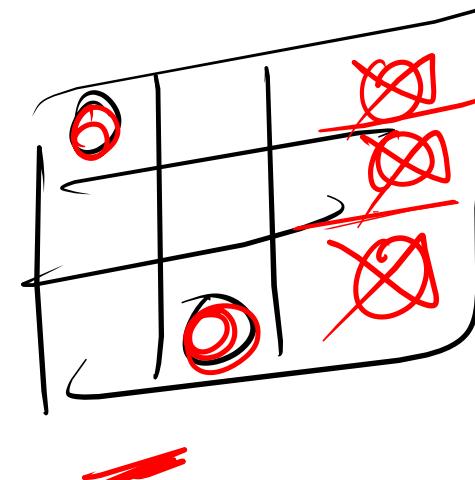


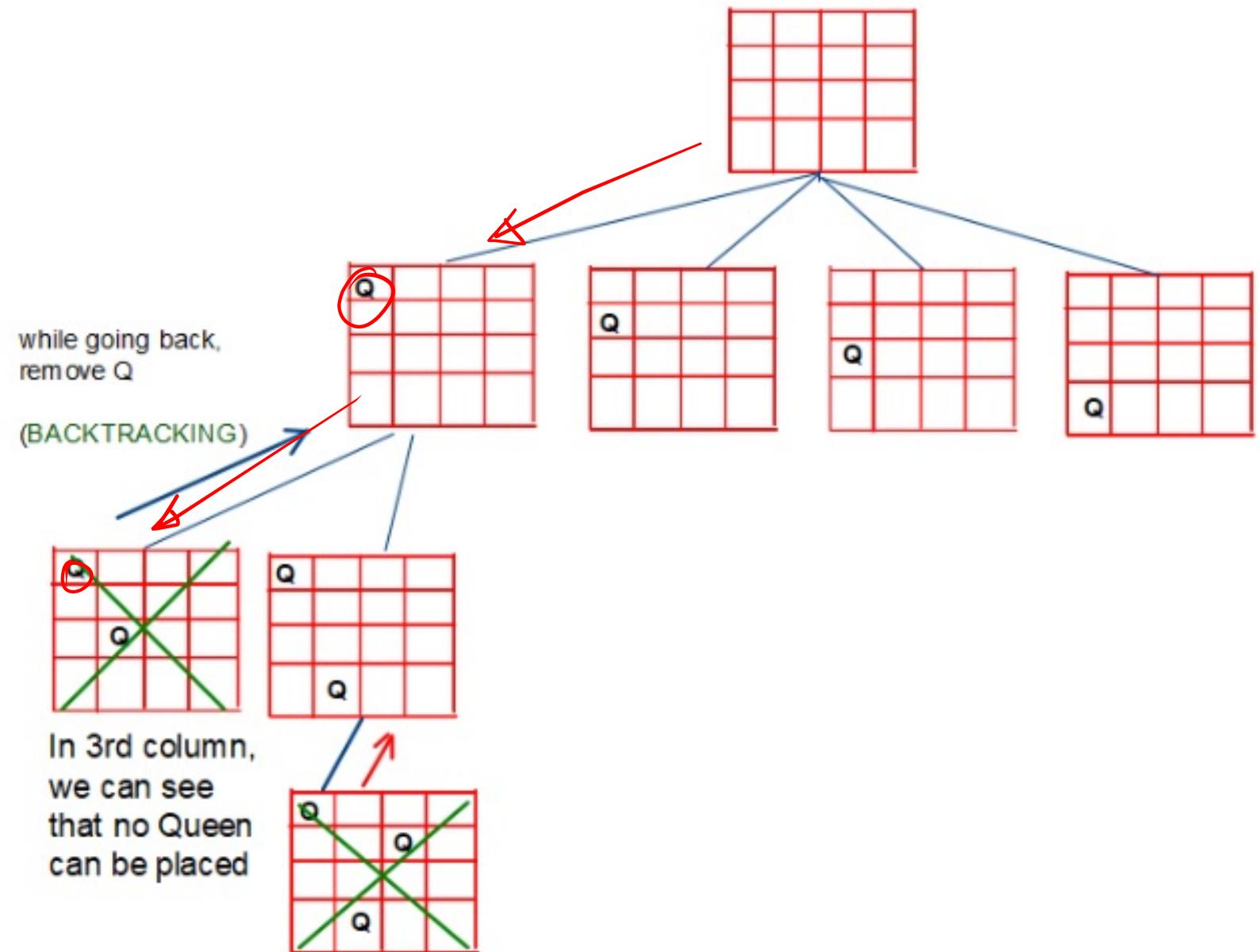
3×3

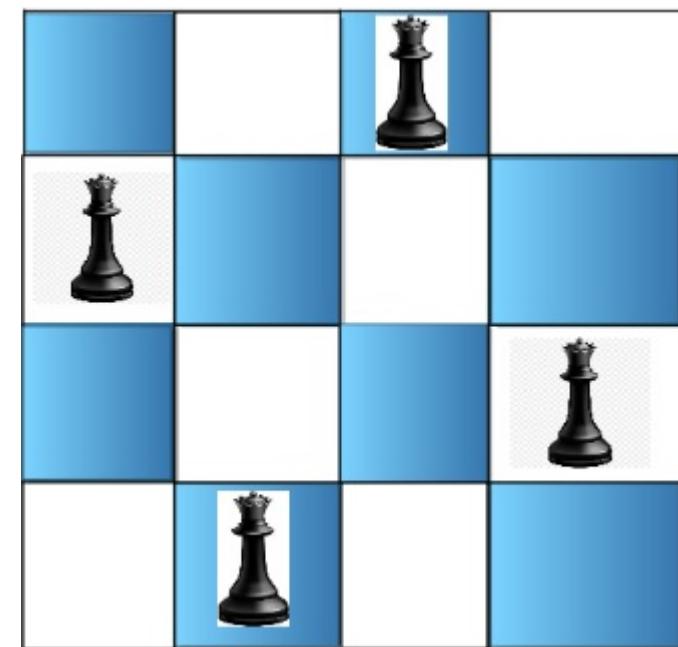
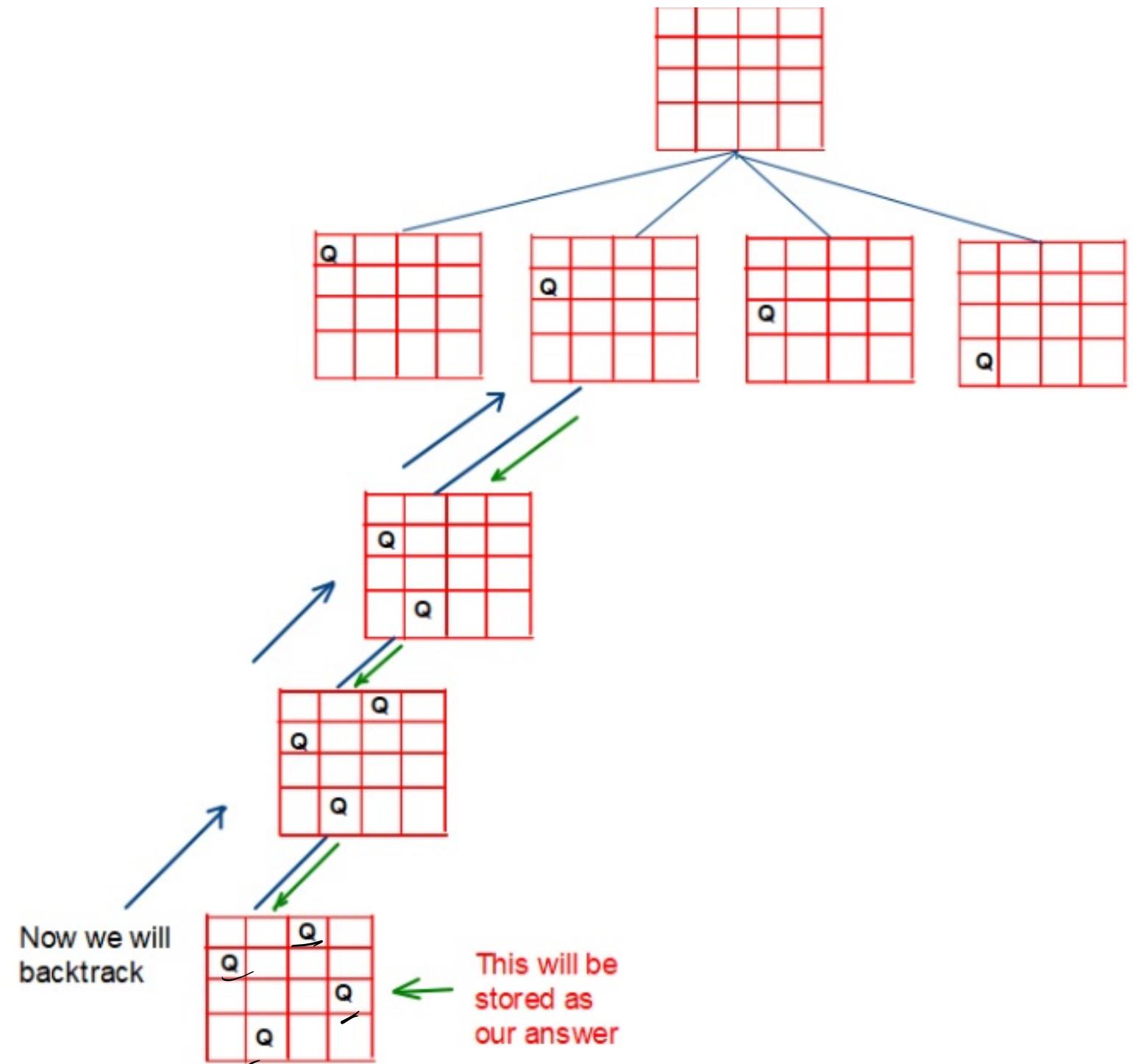
$\text{ans} = 0$

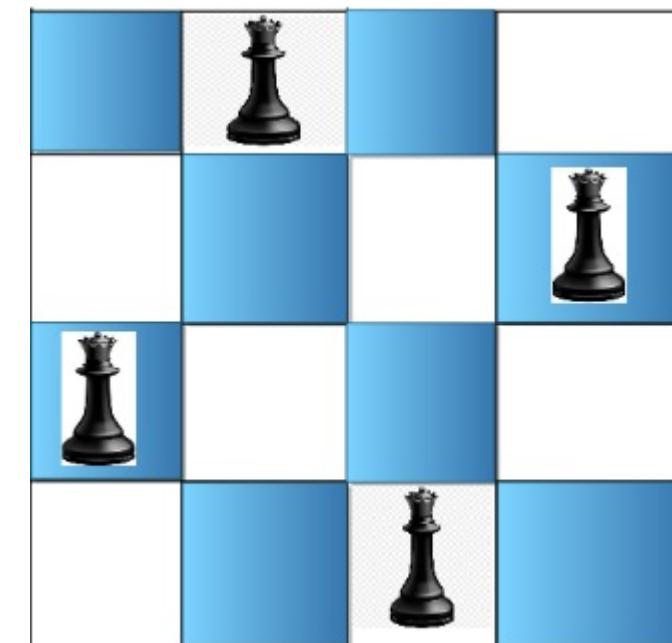
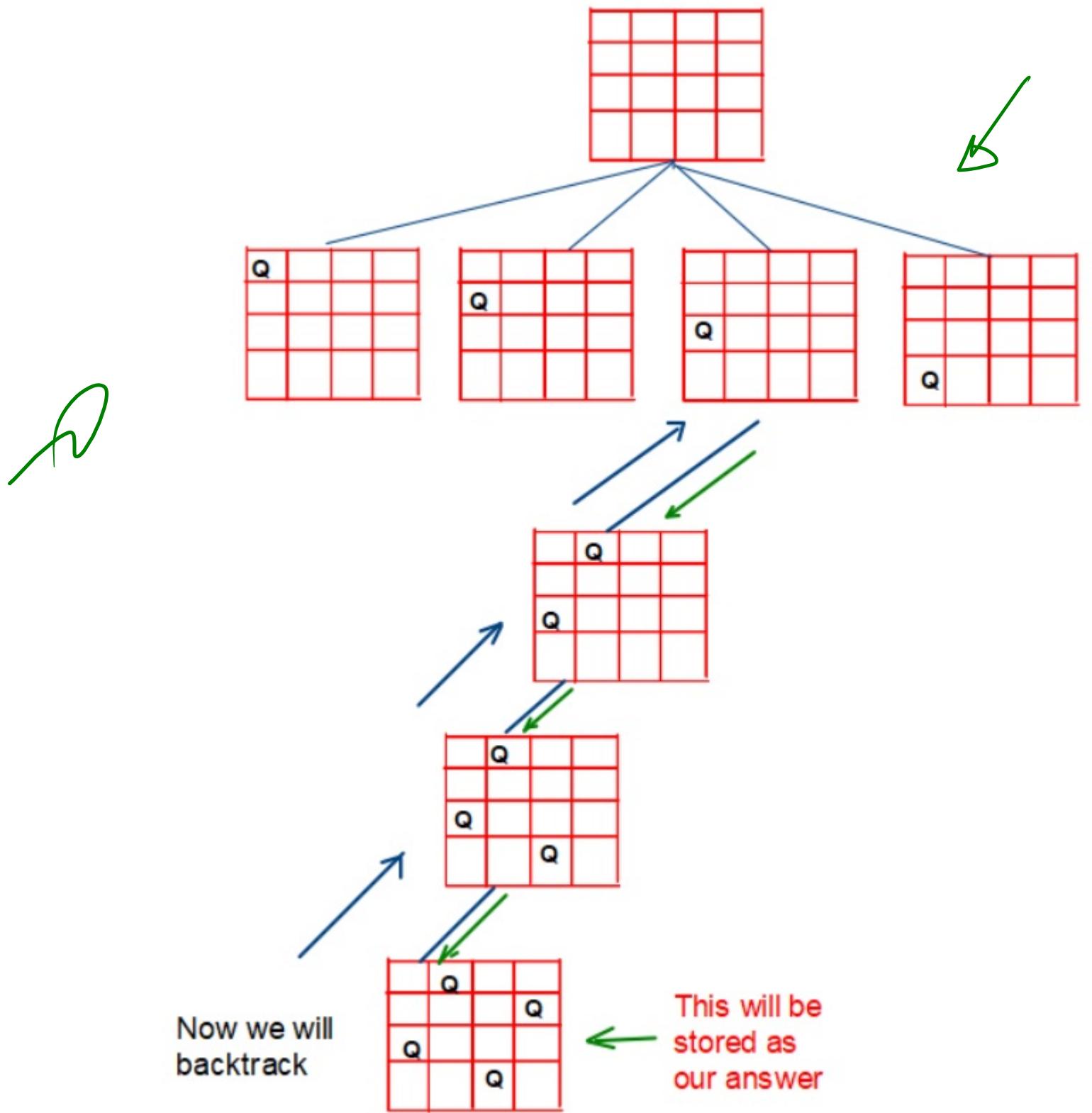


6

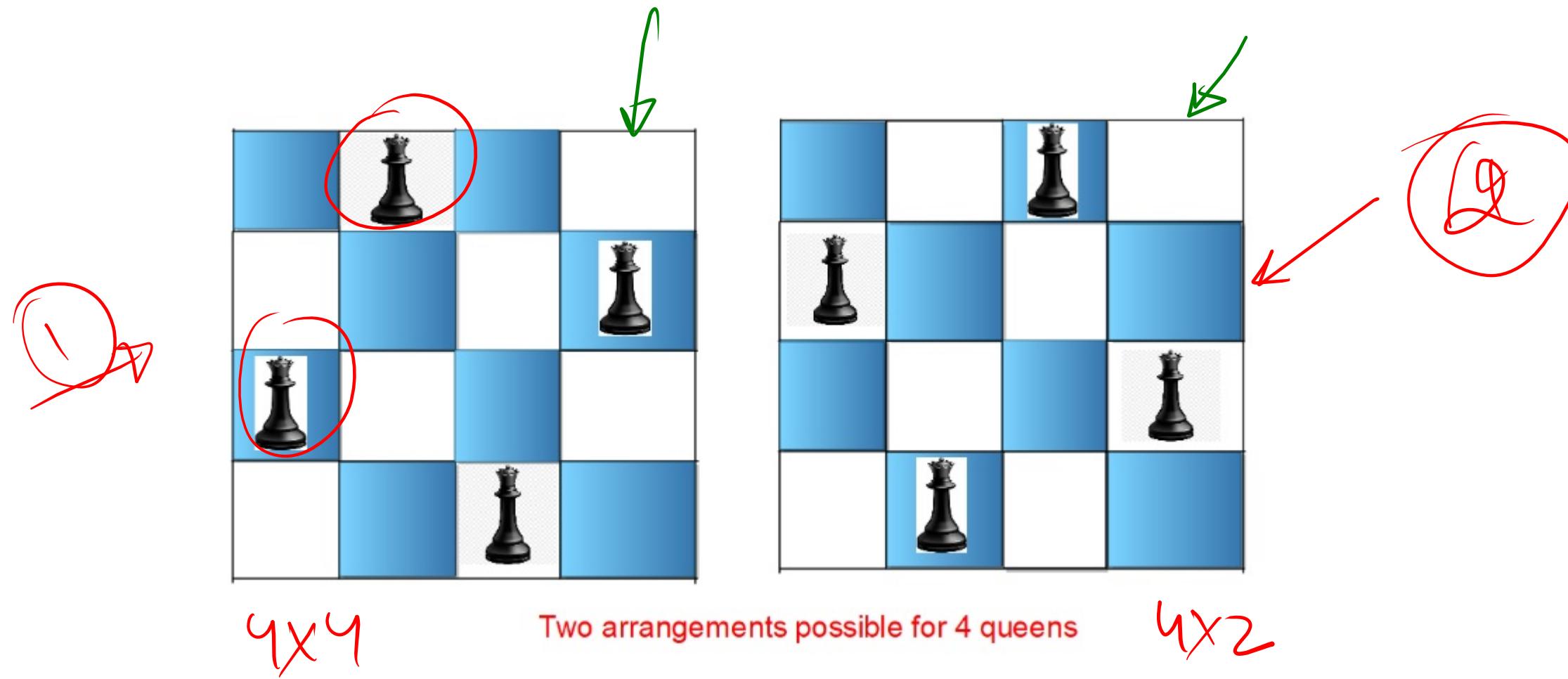


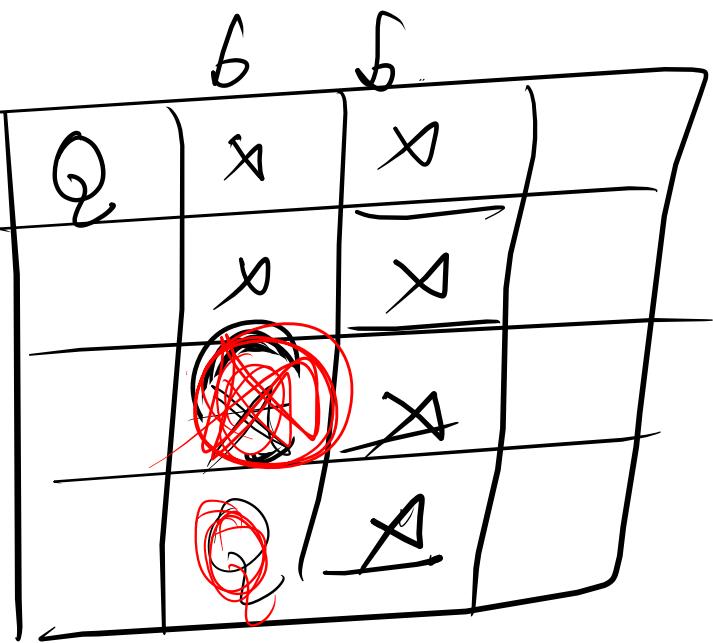
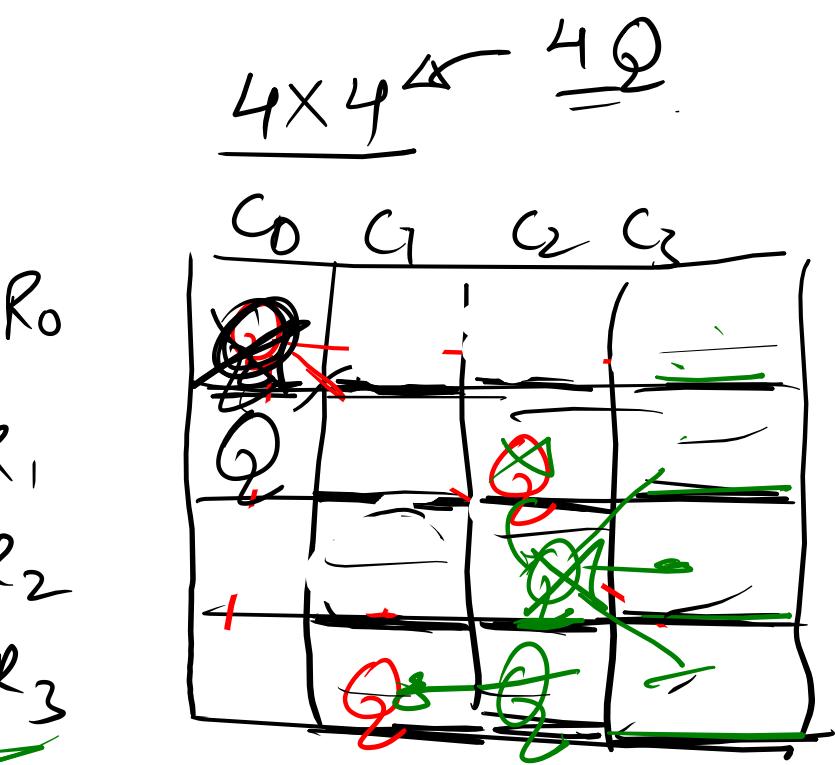
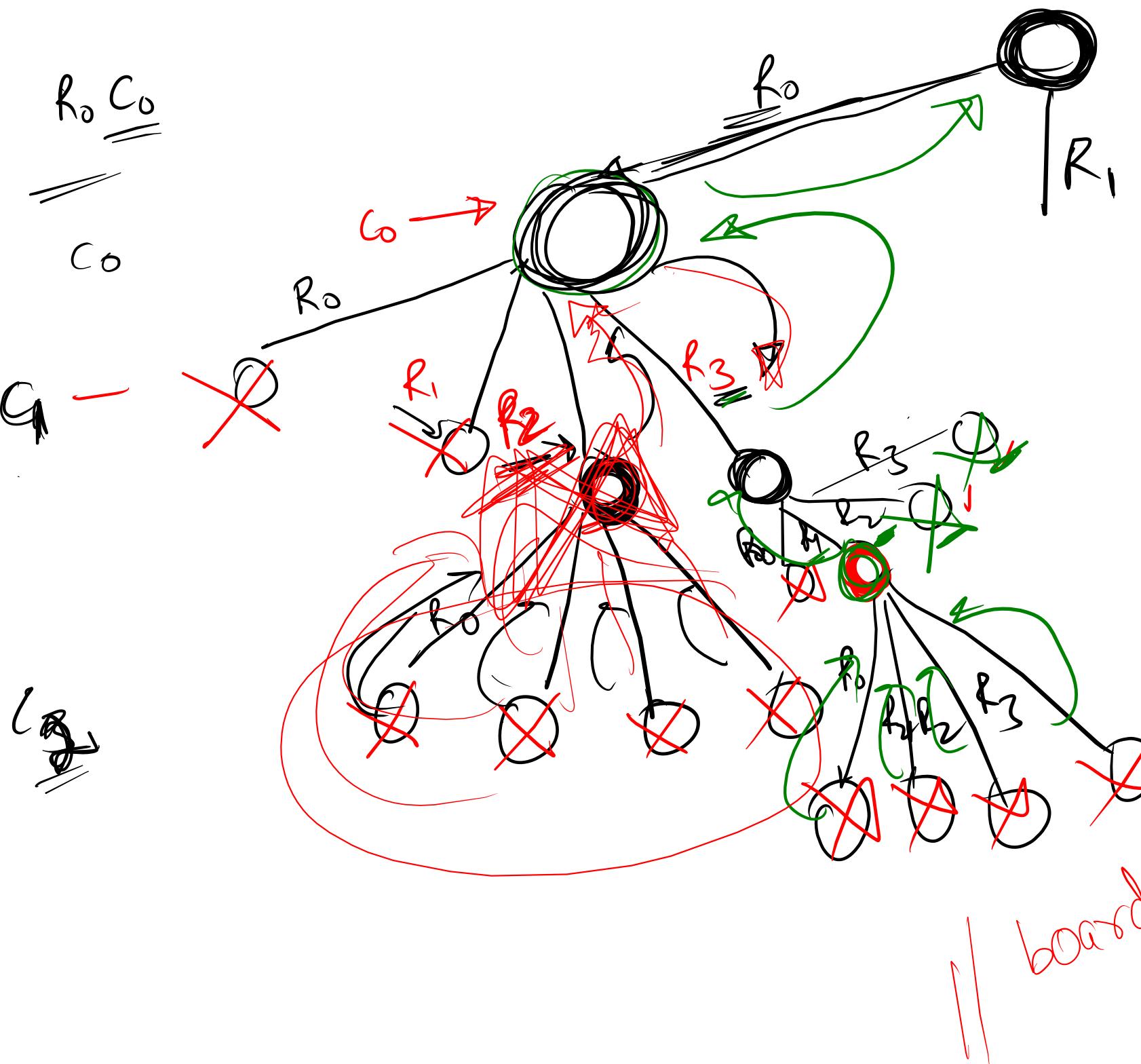






~~4x4~~ → 4 Queen





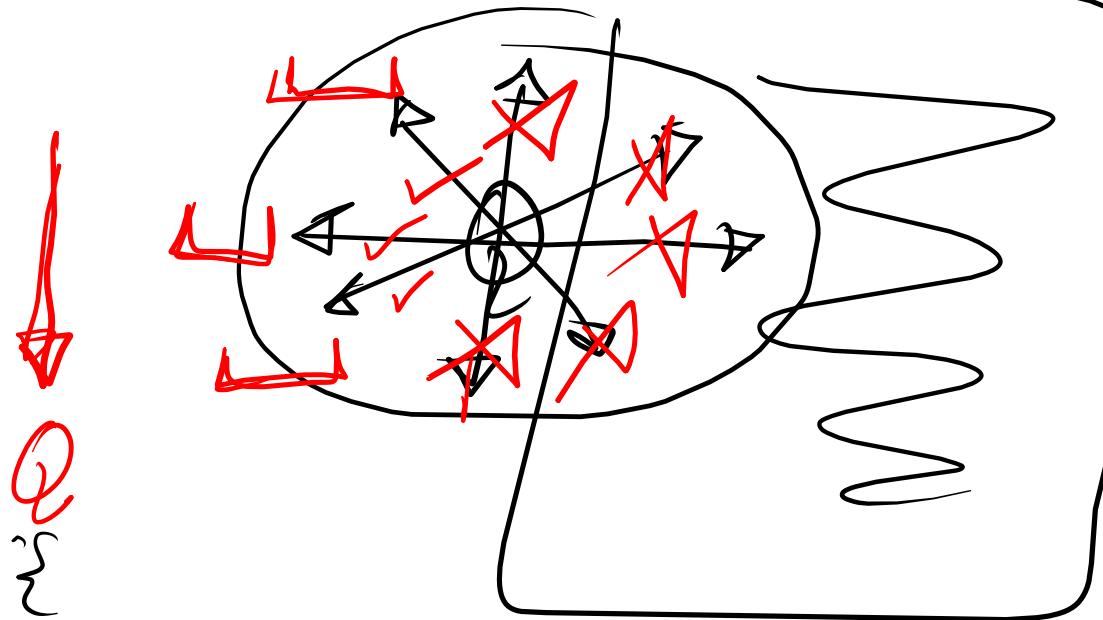
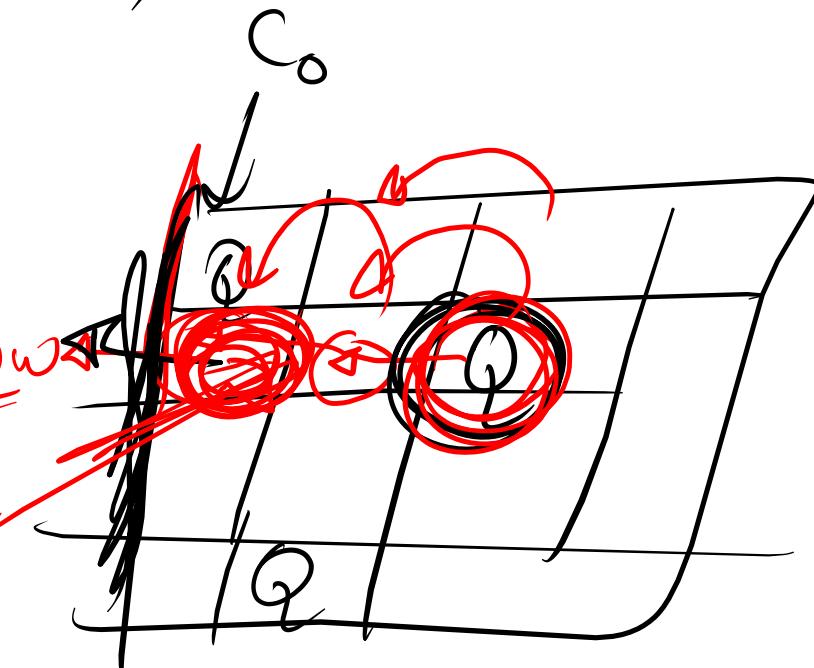
* isSafe (board, row, col)

① left side

while (col >= 0) {

 if (board[row][col] = 'Q') {
 return false;
 col --; *fix*

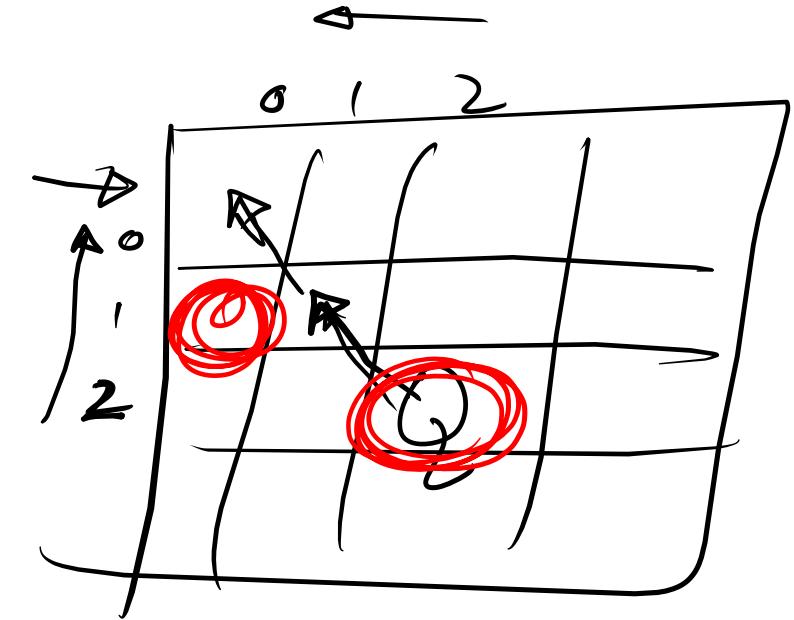
 }
}



```

2           ↗
    // checking left diagonal
    while (row >= 0 & col >= 0) {
        if (board[row][col] == 'Q') {
            return false;
        }
        row--;
        col--;
    }
    return true;

```

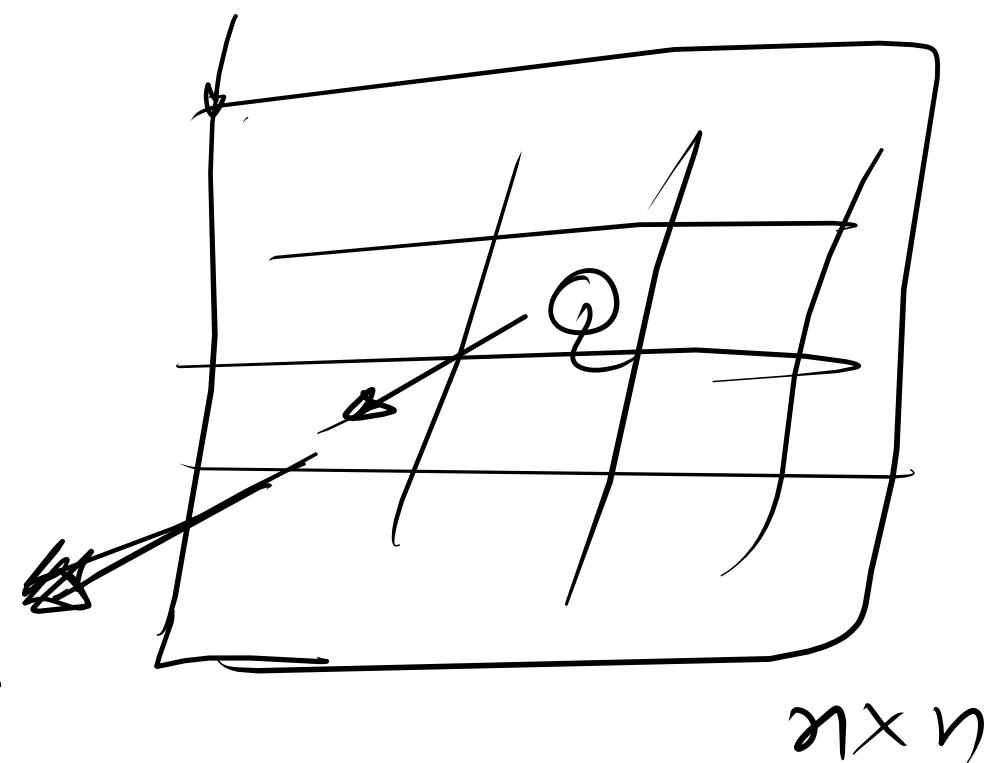


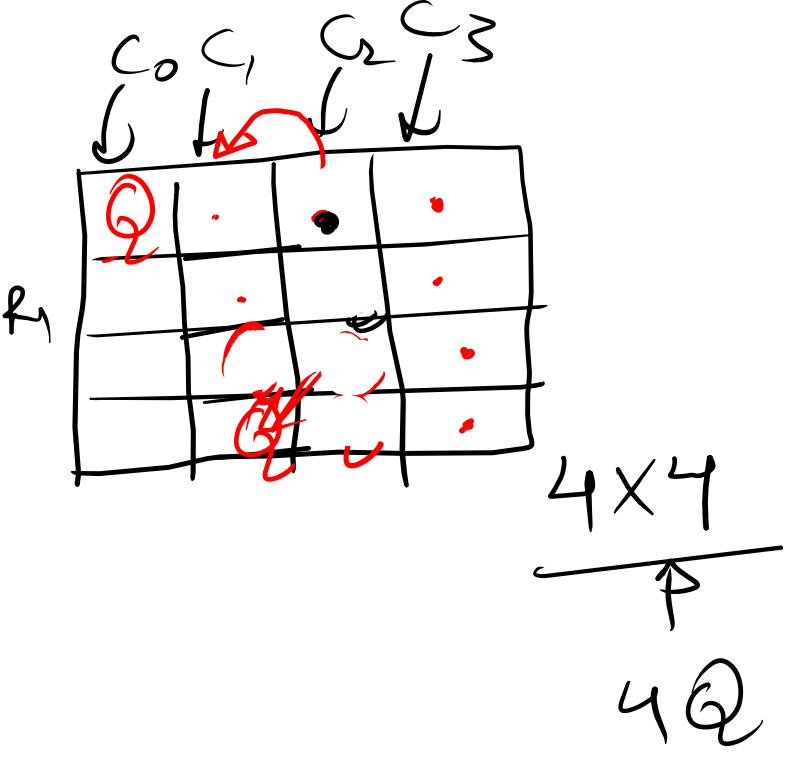
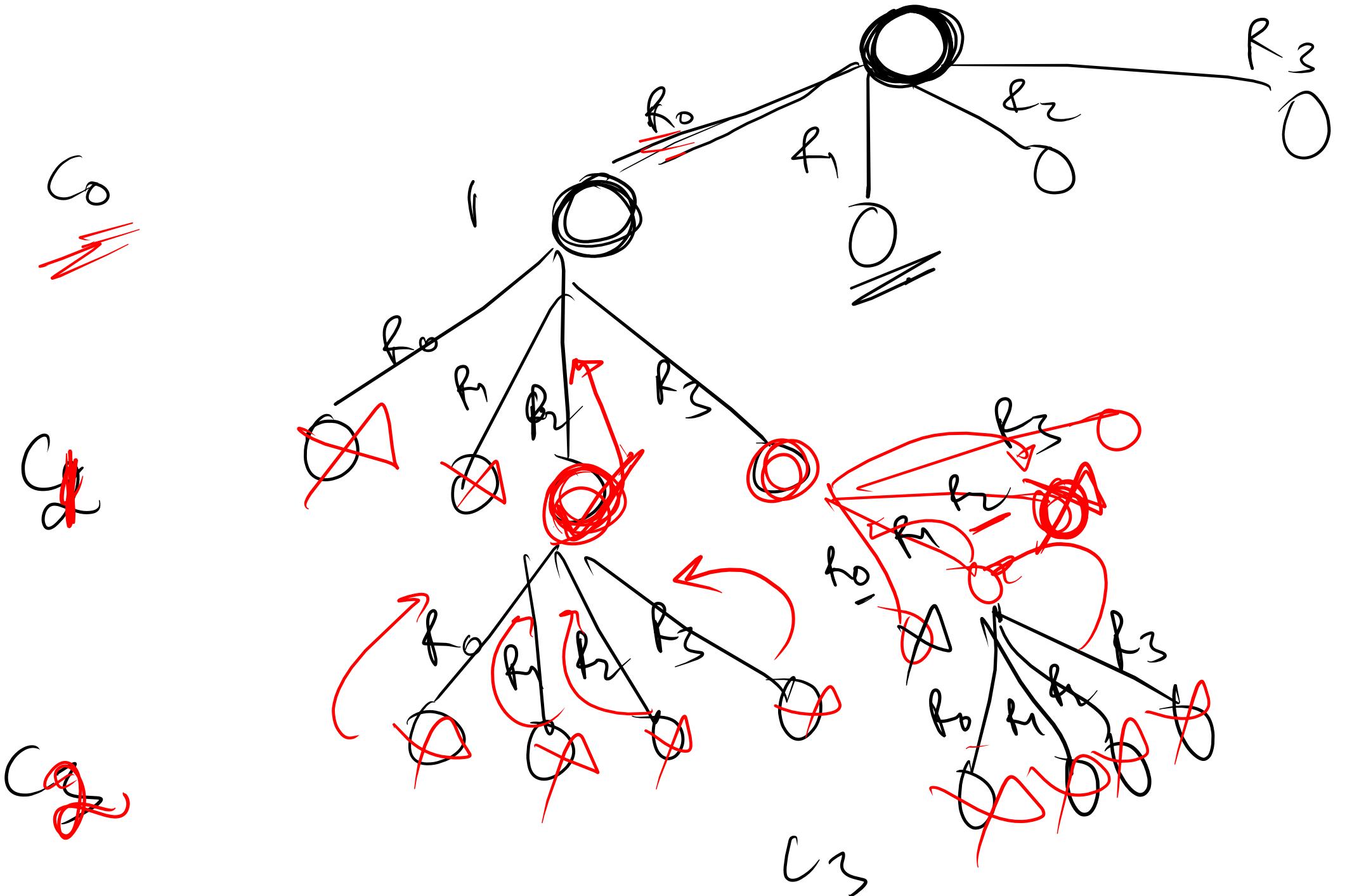
③

// check left bottom diagonal

```
while (row < n && col >= 0) {  
    if (b[row][col] == 'Q') return false;  
    col--;  
    row++;  
}
```

return true;





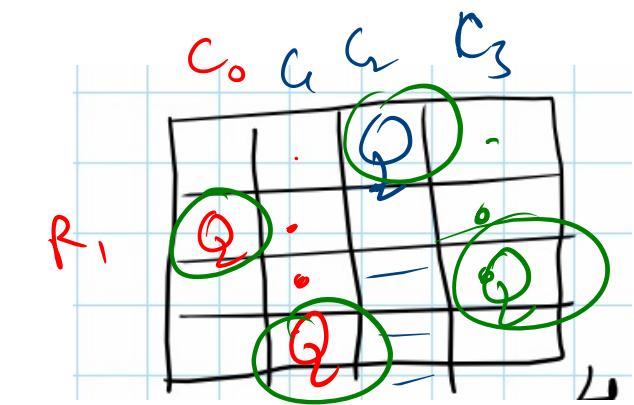
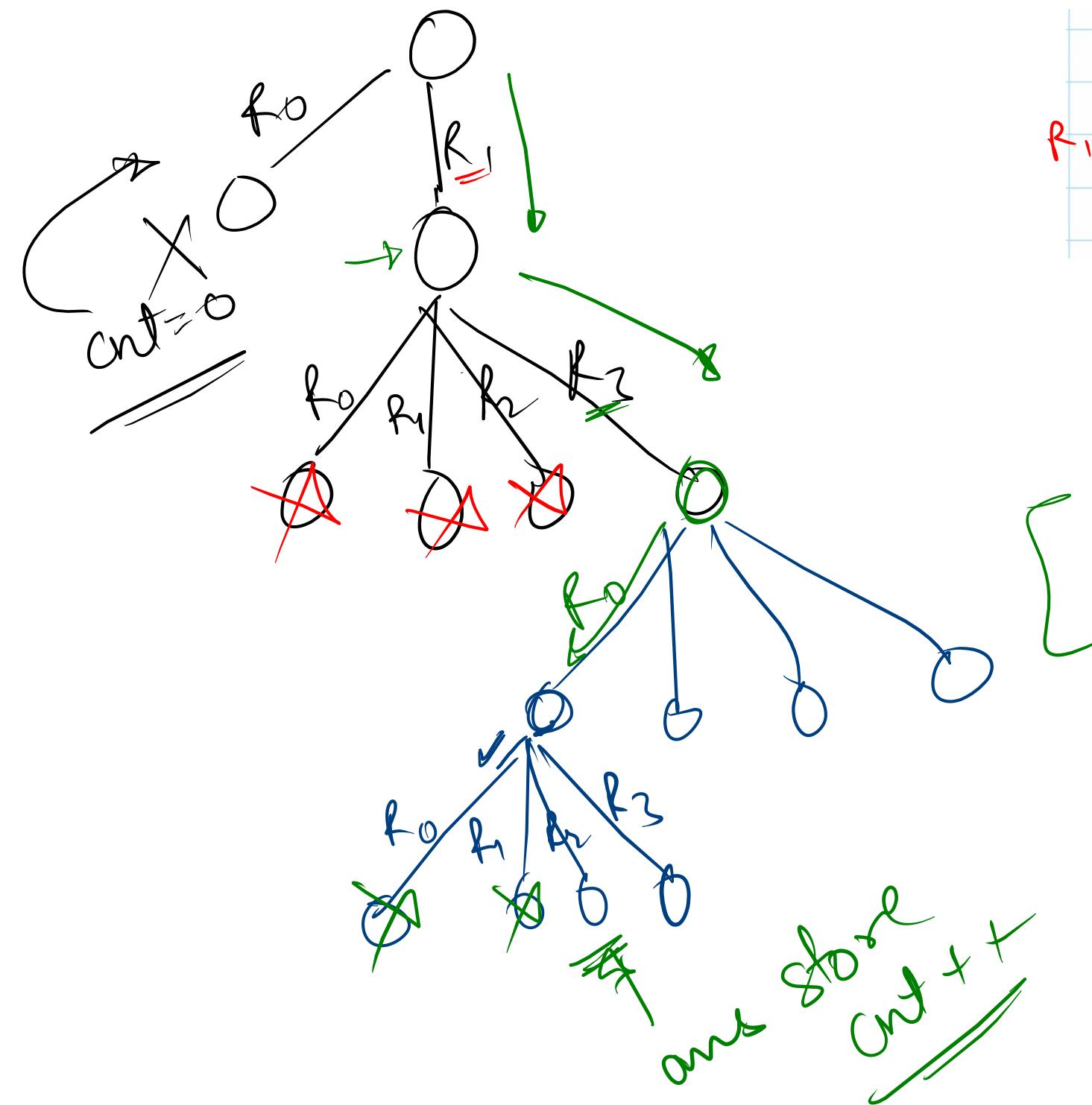
C_0

=

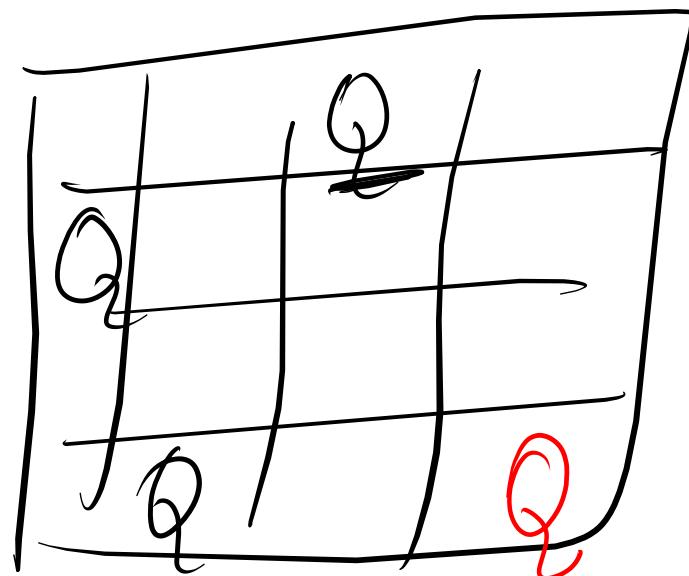
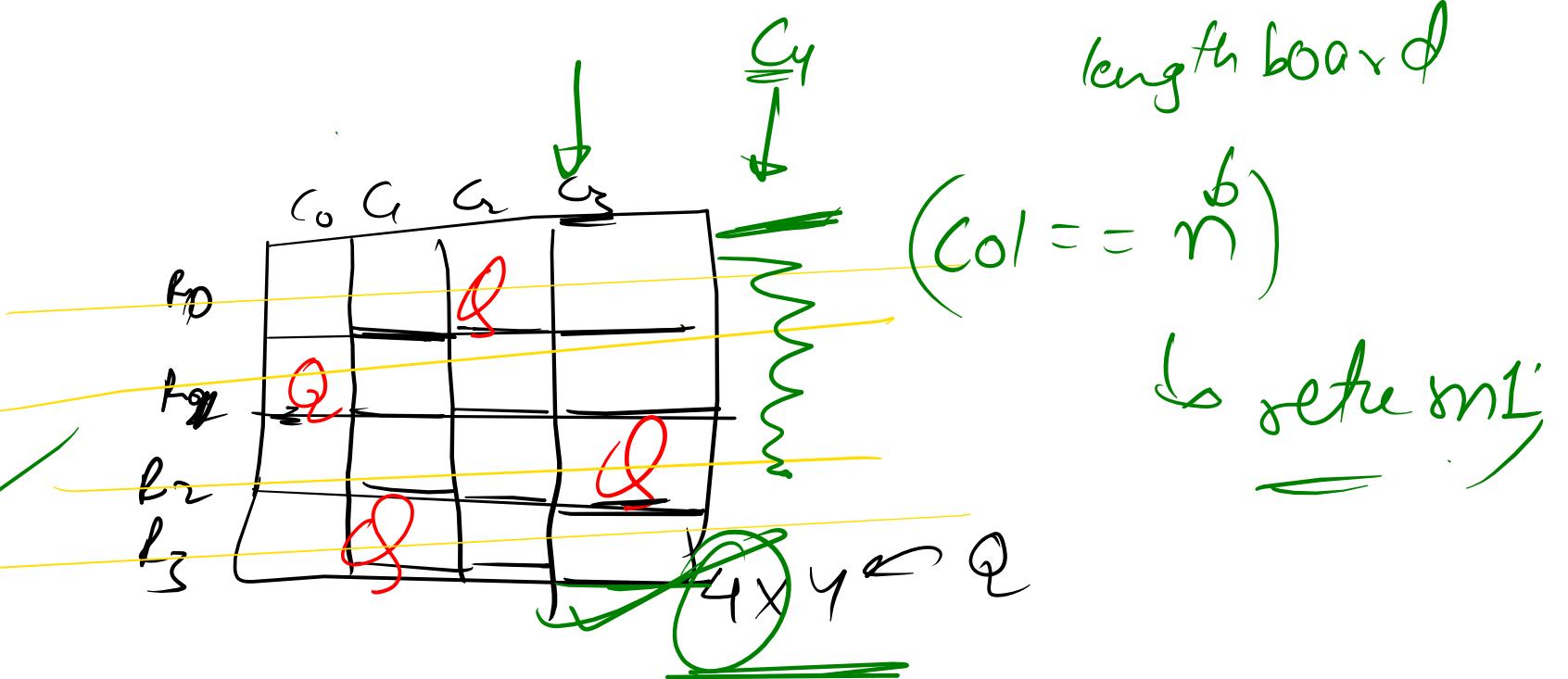
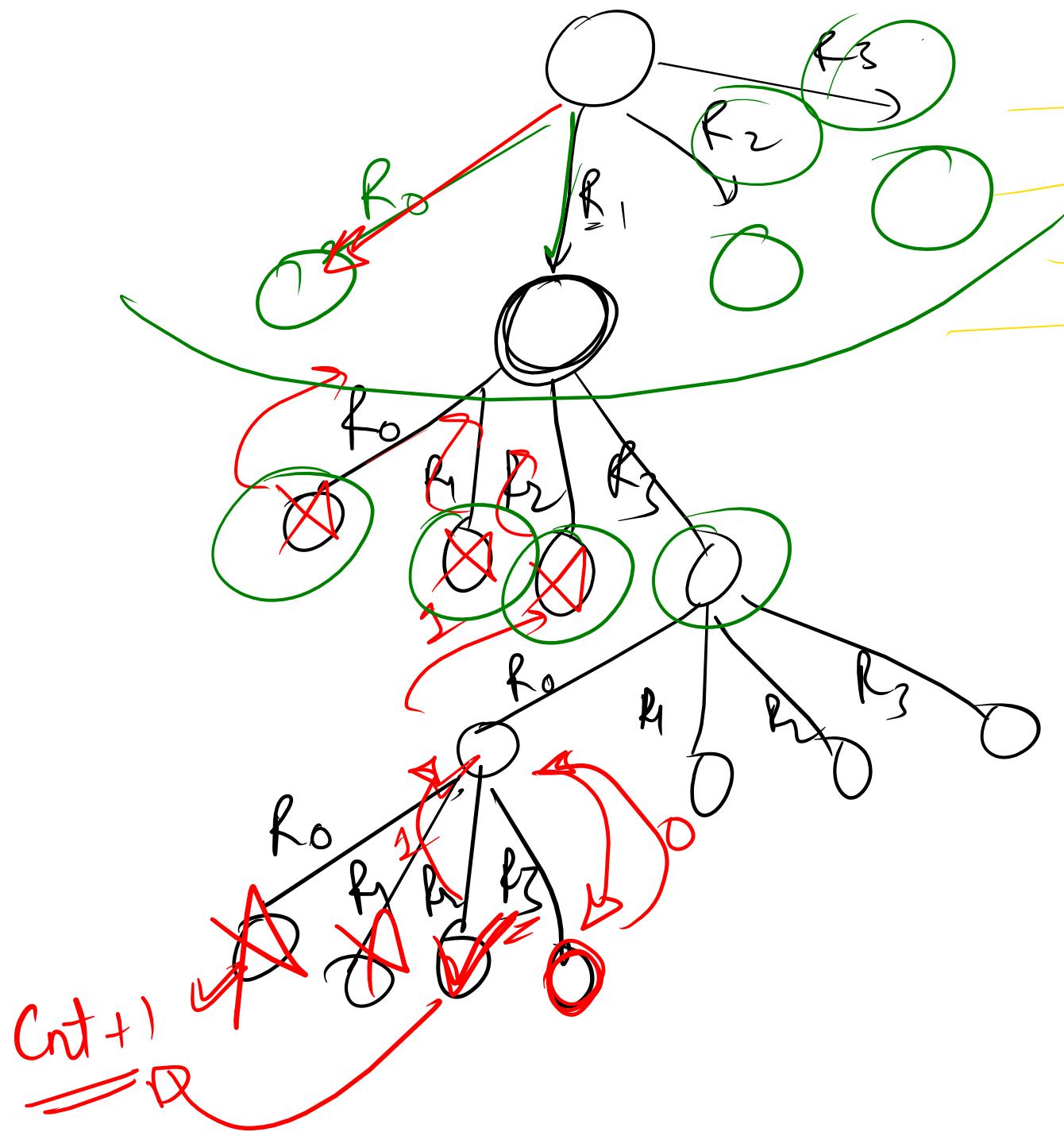
G

C_2

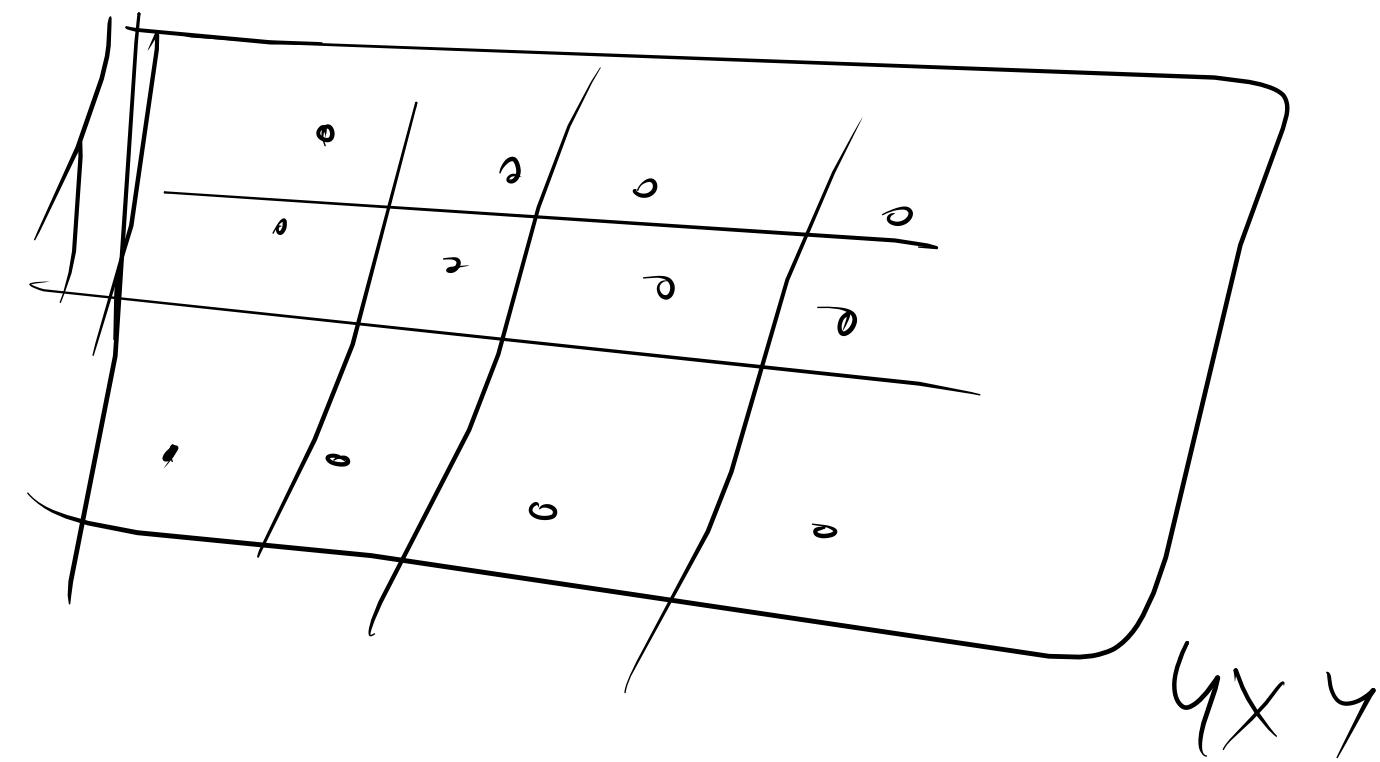
C_3



(f_1, C_0) , (f_3, G) , $(f_0, G)(f_2, C_3)$



chess board



N Queen

```
public static int totalNQueens(int n) {  
    char [][]board = new char[n][n];  
    for(int i = 0 ; i<n ; i++){  
        for(int j = 0 ; j<n ; j++){  
            board[i][j] = '.';  
        }  
    }  
    return solve(0, board);  
}  
  
public static int solve(int col , char[][]board){  
  
    //base case  
    if(col == board.length) return 1 ;  
  
    int cnt = 0 ;  
    //col = 1 ;  
    for(int row = 0 ; row < board.length ; row++){  
        if(isSafe(board, row, col)){  
            board[row][col] = 'Q';  
            cnt += solve(col+1,board);  
            board[row][col] = '.';    // backtracking  
        }  
    }  
    return cnt ;  
}
```

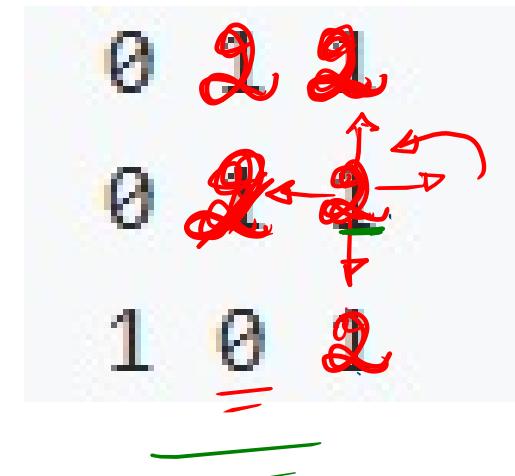
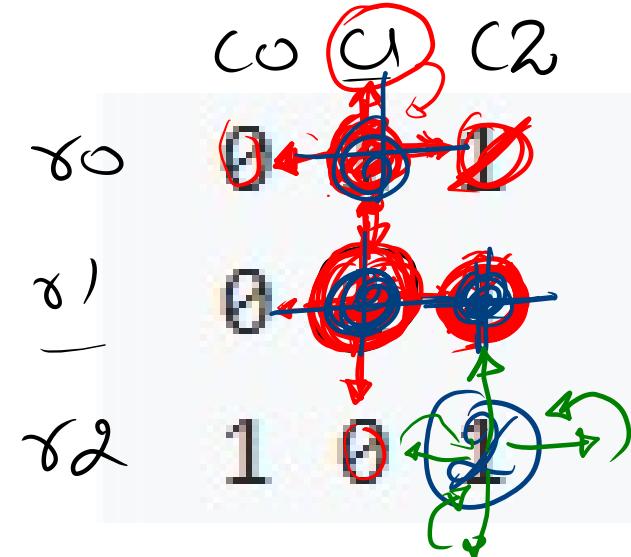
```
public static boolean isSafe(char[][]board , int row , int col){  
  
    int duprow = row ;  
    int dupcol = col ;  
  
    //left isSafe  
    while(col >= 0 ){  
        if(board[row][col] == 'Q') return false;  
        col--;  
    }  
  
    row = duprow ;  
    col = dupcol ;  
    //upper left diagonal  
    while( row >= 0 && col >= 0){  
        if(board[row][col] == 'Q') return false;  
        row--;  
        col--;  
    }  
  
    row = duprow ;  
    col = dupcol ;  
  
    //bottom left diagonal  
    while( row < board.length && col>= 0 ){  
        if(board[row][col] == 'Q') return false;  
        row++;  
        col--;  
    }  
    return true ;  
}
```

flood fill

row = 1 col = 1 new color = 2

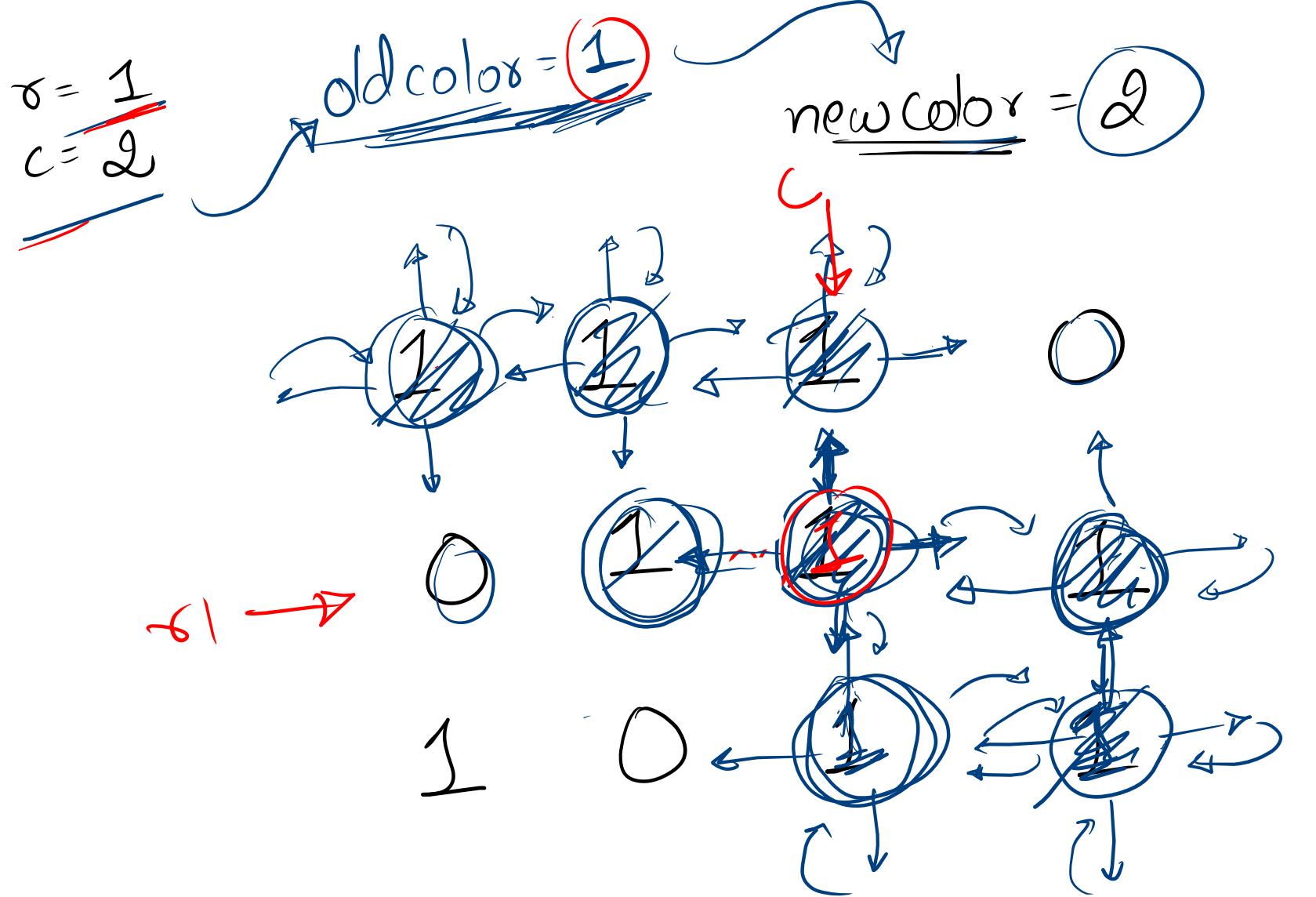
old color = 1 2

0/9



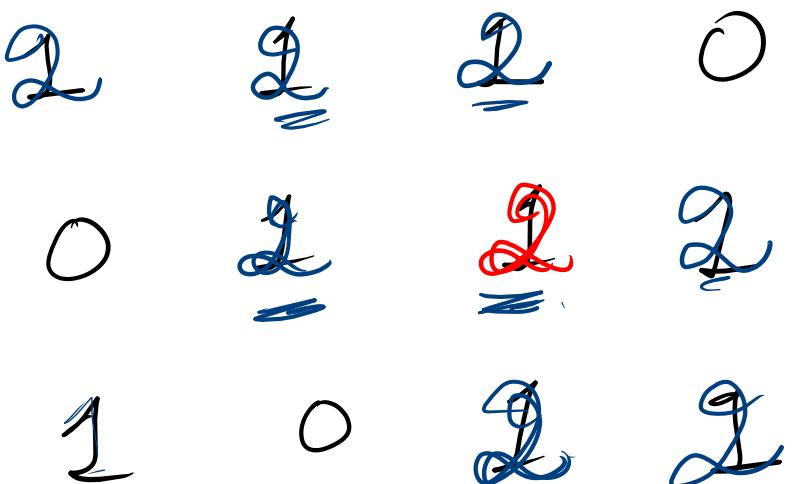
0	2	2
0	2	2
1	0	2

Condition: 4-direction

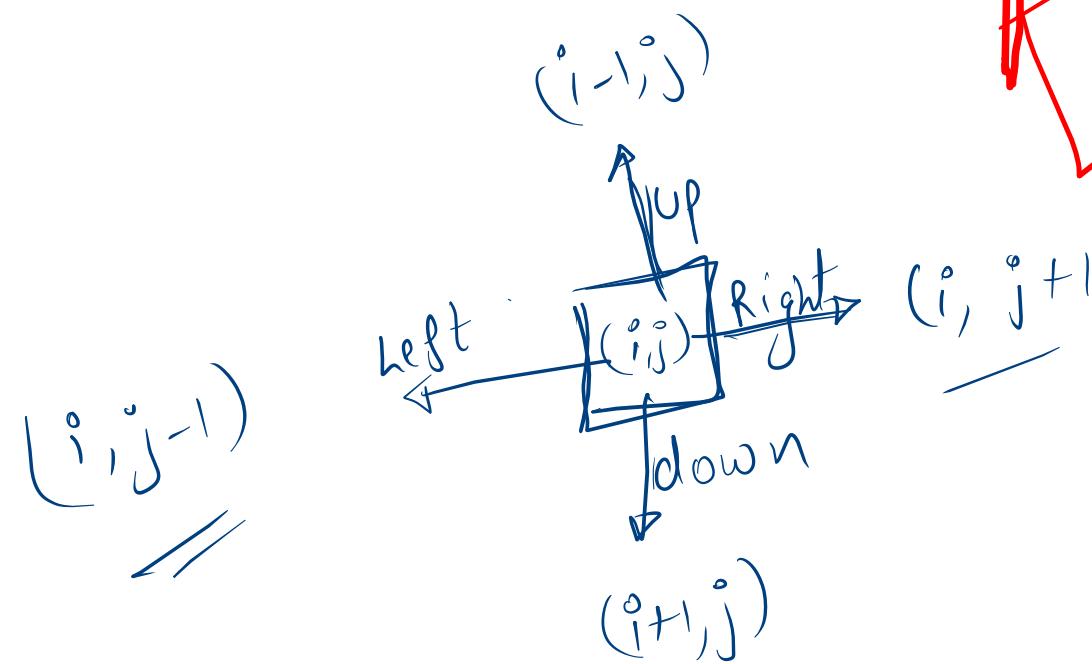


condition: →

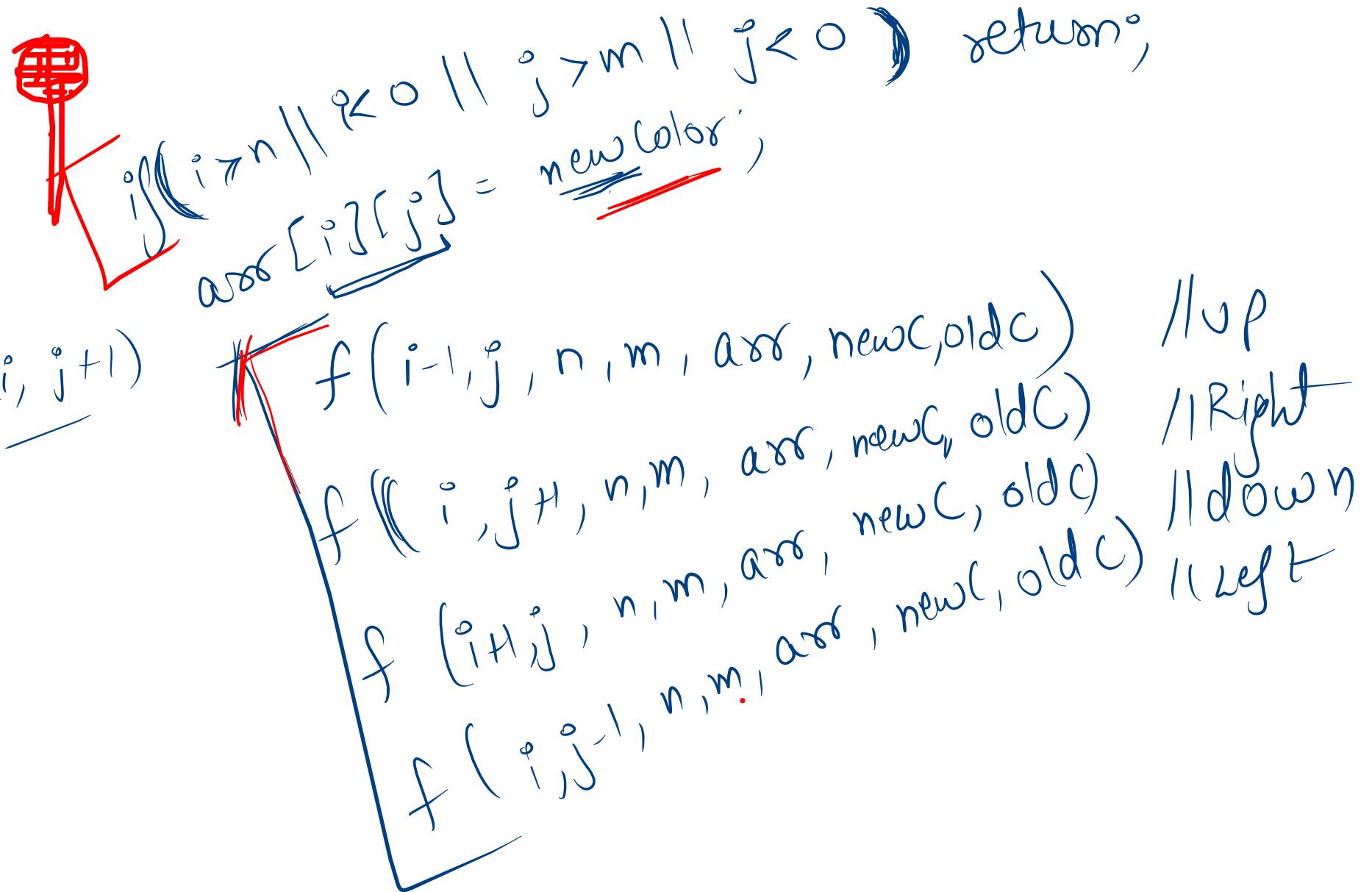
 up
 Right
 down
 left

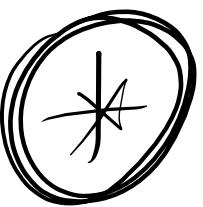


oldcolor = arr[i][j]



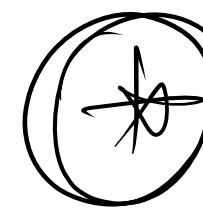
LeetCode - 433
→ solve.



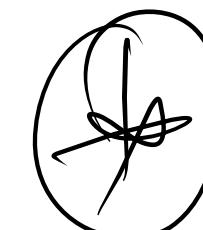


Combination sum

↳ way of Natural no.



Combination sum ↴



flood fill

:
leetcode - 39

:
leetcode - 40

:
leetcode - 733

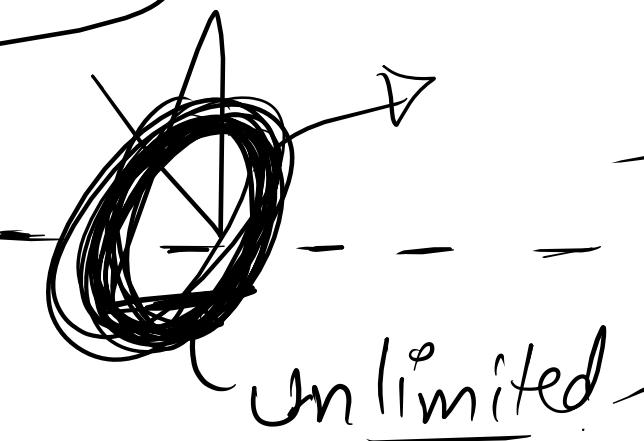
1. Combination Sum 1

Sum > T

$$T = \underline{\underline{7}}$$

arr

[- - - -]



unlimited

(a[], T)

T
NT

Store

[+ +]

subset 1

[- -]

subset 2

[- - -]

- 3

[- - - -]

- 4

(T - arr[])

T
NT

(arr, NT - arr[])

T
NT

$f(\text{arr}, T, \text{sum}, i)$

unlimited

Take

Not take

$f(\text{arr}, T, \cancel{\text{sum} + \text{arr}[i]}, i+1)$

$f(\text{arr}, T, \cancel{\text{sum} - \text{arr}[i]}, i+1)$

if (Target == sum)

ans.add(target); return