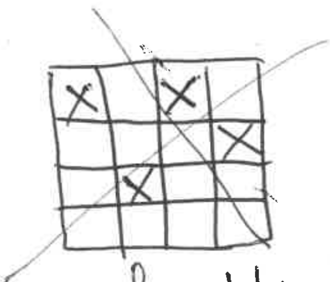
Backtracking algorithms

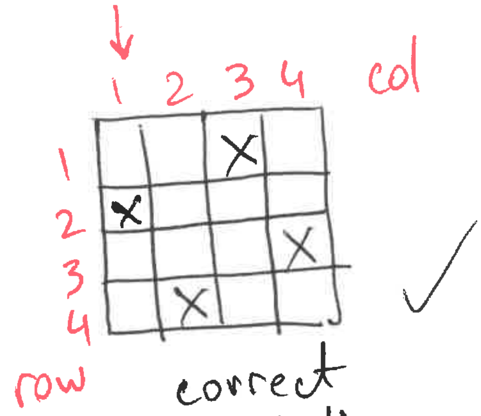
2.3.2017

n queens problem – place n queens on an n×n chess board, such that no 2 queens attack each other by being on the same row, same column, or same diagonal.

$n=4$



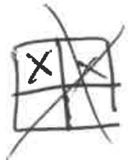
unfeasible solution



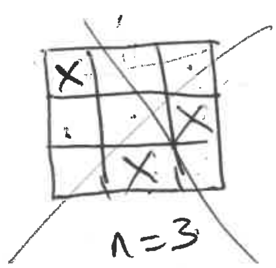
correct solution

$n=1$, solution is trivial

$n=2$ and $n=3$ do not have a solution



$n=2$



$n=3$

$n \geq 4$, there is always a feasible solution

solution:

(comp 1, comp 2, - - - - , comp n)

comp 1: place 1st queen in the first col

comp 2: place 2nd queen in the second column

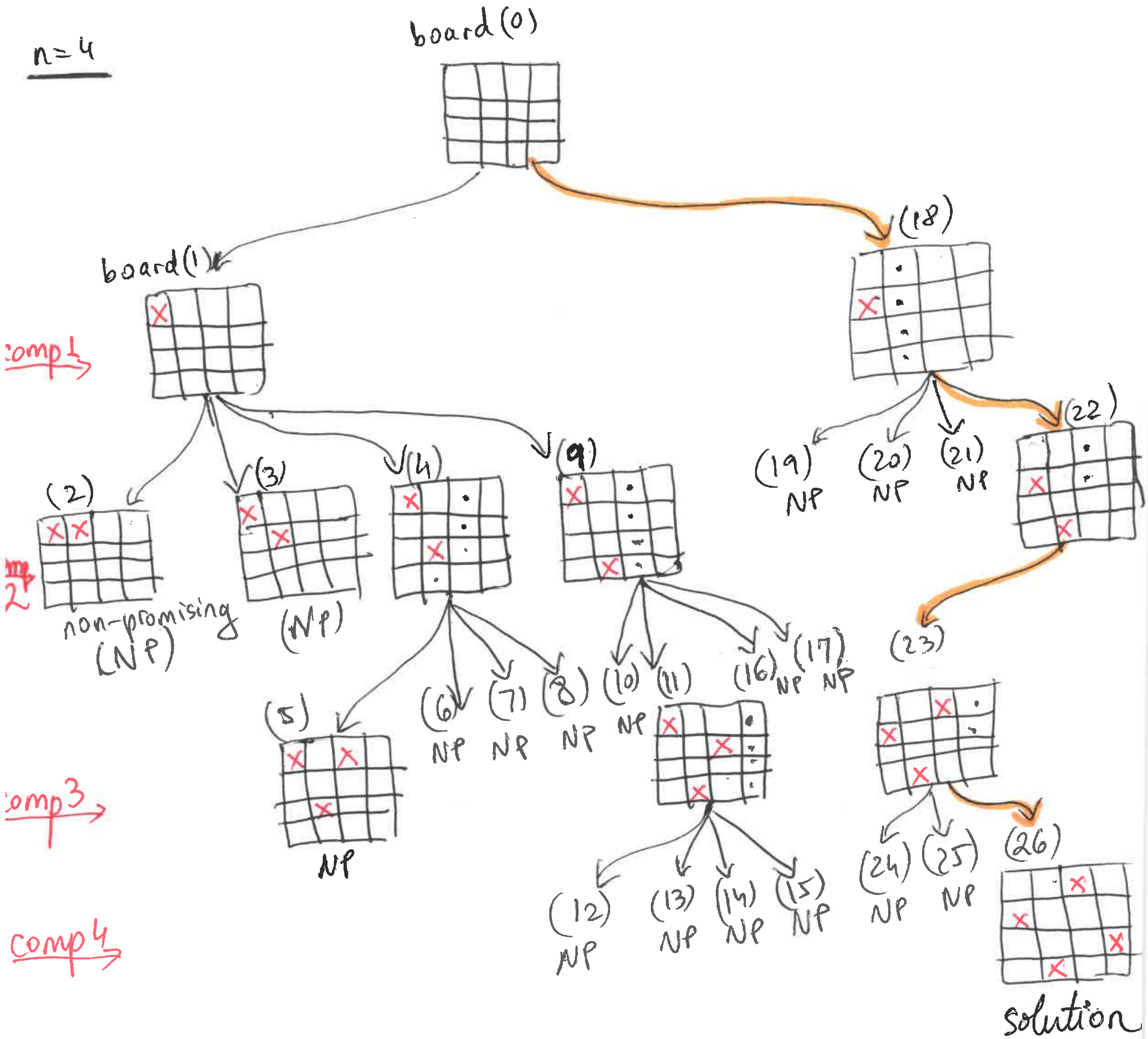
comp n: place nth queen in the nth column board (i) – chess board after i attempts

board (0) – initial chessboard, no queens placed

strategy

- place queens column by column, left-to-right
- for each column, try placing the queen row by row, top-bottom

n=4



rn-queens (k, n)

- tries to place a queen in column k
- assumes queens have been already properly placed in cols 1, 2, .., k-1

if a queen is properly placed in col k, then recursively call rn-queens (k+1, n)

otherwise, if it fails to place a queen in col k, then it backtracks one level, trying a new location for the queen in col k-1

array row [1..n]

row [k] – indicates the row in which the queen in col k is placed

example

	1	2	3	4
1			X	
2	X			
3				X
4		X		

row

1	2	3	4
2	4	1	3

solution: (row[1], row[2], row[3], row[4])

position-ok (k, n)

- returns true if the queen in col k does not conflict with the queens in cols 1, 2, .., k-1
- returns false otherwise

- conflict for queens on the same row

array row_used [1..n]

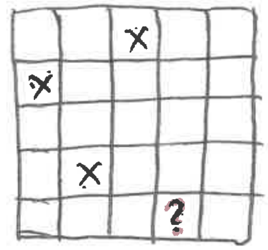
row-used [r] – true if a queen occupies row r; false otherwise example

let $K=4$

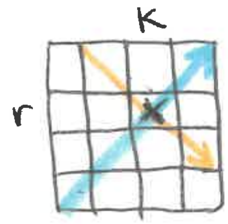
	1	2	3	4	5
row	2	4	1	5	

	1	2	3	4	5
row_used	T	T	F	T	F

$K=4$



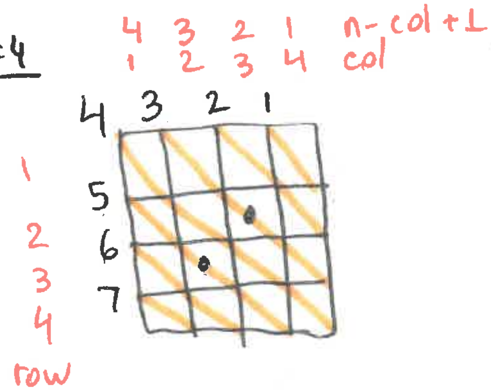
- conflict for queens on the same diagonal



row column
(r, K)

ddiag - downward diagonal in direction ↘

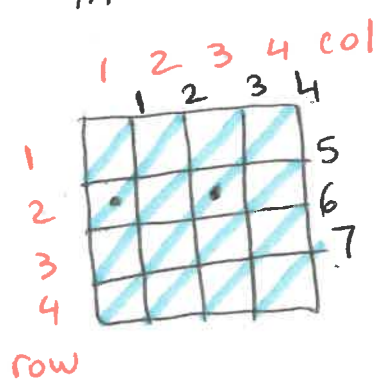
$n=4$



array ddiag_used [1..2n-1]

ddiag_used[d] → true if there is a queen on the ddiag d
→ false otherwise

udiag - upward diagonal in direction ↗



array udiag_used [1..2n-1]

udiag_used [d] → true if there is a queen on the udiag d
→ false otherwise

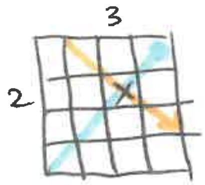
- if a queen is placed in (r, K) then the following diagonals are occupied:

$$\begin{cases} \text{udiag } r+K-1 \\ \text{ddiag } r+n-K \end{cases}$$

$r+(n-K\cancel{+1})\cancel{-1} = r+n-K$

example

$n=4$



- a queen placed in (2,3) is in udiag 4 ← $2+3-1=4$
and in ddiag 3 ← $2+4-3=3$

position_OK(k,n)

```
return !(row_used [row[k]] || ddiag_used [row[k]+n-k] ||
         udiag_used [row[k]+k-1])
```

n_queens(n)

```
for i=1 to n
    row_used [i] = false
for i=1 to 2n-1
    ddiag_used [i] = false
    udiag_used [i] = false
rn_queens (1, n)
```

rn_queens(k,n)

```
for row[k] = 1 to n
    if position_OK(k,n) == true
        row_used [row[k]] = true
        ddiag_used [row[k]+n-k] = true
        udiag_used [row[k]+k-1] = true
        if k == n
            print solution: row[1], row[2], ..., row[n]
            //stop here if only one solution is desired
        else // k < n
            rn_queens (k+1, n)
        row_used [row[k]] = false
        ddiag_used [row[k] +n-k] = false
        udiag_used [row[k]+k-1] = false
```

RT analysis

- how many times is rn_queens(k,n) called?

	$k=1$	1 time
	$k=2$	n times
$\underline{\times}\ \underline{\times}\ \underline{?}$	$k=3$	$\le n\cdot(n-1)$ times
$\underline{\times}\ \underline{\times}\ \underline{\times}\ \underline{?}$	$k=4$	$\le n\cdot(n-1)(n-2)$ times
	$\vdots$	
	$k=n$	$\le n\cdot(n-1)(n-2)\cdots 2$ times

- rn_queens() takes $\Theta(n)$ besides the recursive calls

$$RT \le n\left(1 + n + n\cdot(n-1) + n(n-1)(n-2) + \dots + n(n-1)(n-2)\cdots 2\right)$$

$$RT \le n\cdot n!\left(\frac{1}{n!} + \frac{n}{n!} + \frac{n(n-1)}{n!} + \frac{n(n-1)(n-2)}{n!} + \dots + \frac{n(n-1)(n-2)\cdots 2}{n!}\right)$$

$$RT \le n\cdot n!\ \left(\underbrace{\frac{1}{n!} + \frac{1}{(n-1)!} + \frac{1}{(n-2)!} + \frac{1}{(n-3)!} + \dots + \frac{1}{1!}}\right)$$

We know that $\boxed{e = \sum_{i=0}^{\infty} \frac{1}{i!}}$, where $e = 2.718\ldots$

$$\frac{1}{1} + \underbrace{\frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!}} = e \qquad (n \to \infty)$$

$$RT \le n\cdot n!\,(e-1)$$

$$\boxed{RT = O(n\cdot n!)}$$