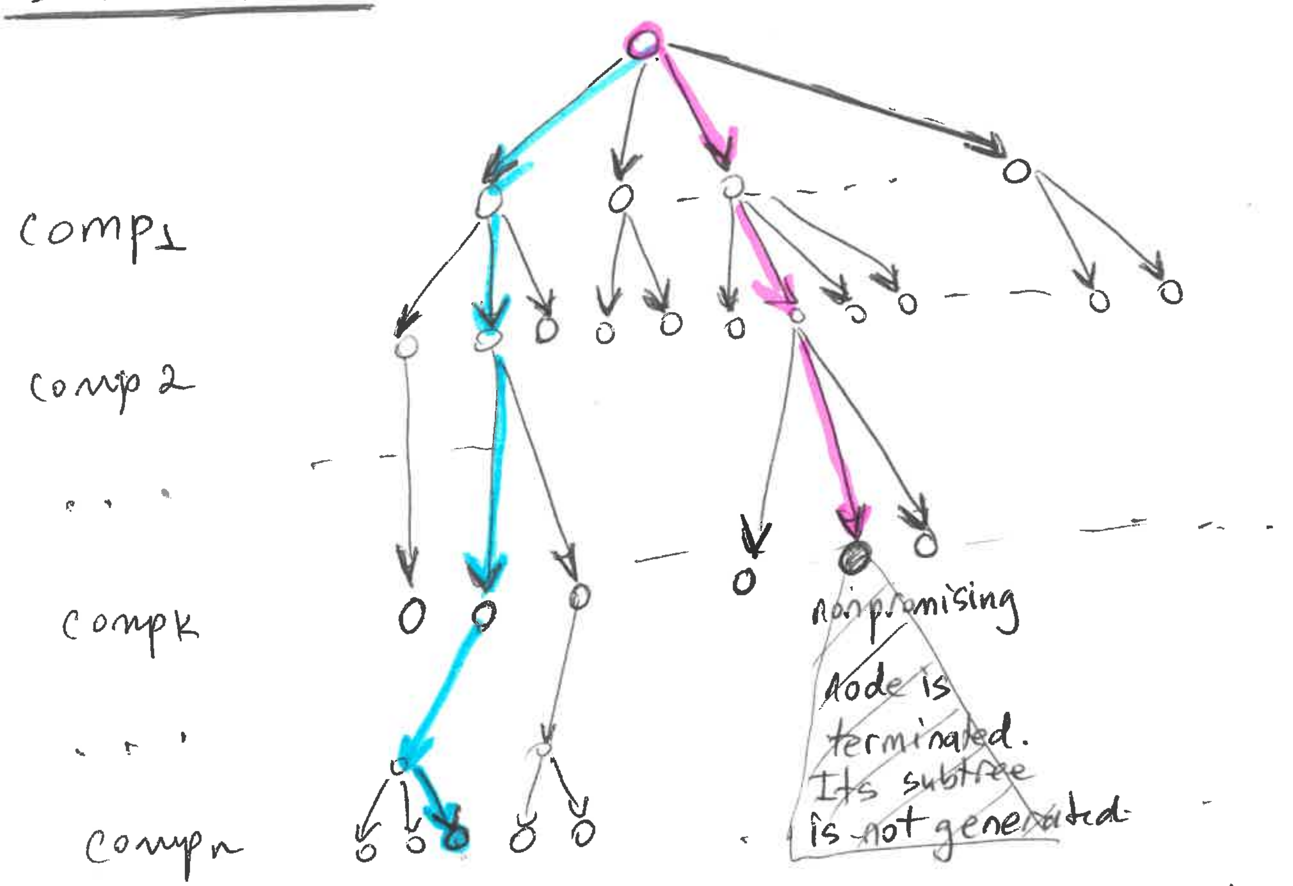
Backtracking algorithms

solution = $(\text{component}_1, \text{component}_2, \dots, \text{component}_k, \dots, \text{component}_n)$

$\xrightarrow{\hspace{6cm}}$
partial solution

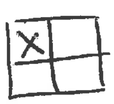
Search Tree



n-queens problem: place n queens on an $n \times n$ board such that no two queens attack each other by being on the same row, column or diagonal.

$n=1$, solution is trivial 

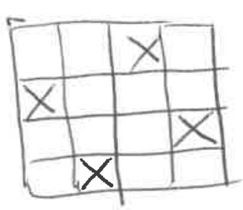
For $n=2$ and $n=3$, the problem does NOT have a solution.


$n=2$


$n=3$

For $n \geq 4$, the problem always has a solution.

$n=4$



solution = ($comp_1$, $comp_2$, $comp_3$ - - - - $comp_n$)

place $queen_1$ on col_1 — place $queen_2$ on col_2 — place $queen_3$ on col_3 — place $queen_n$ on col_n

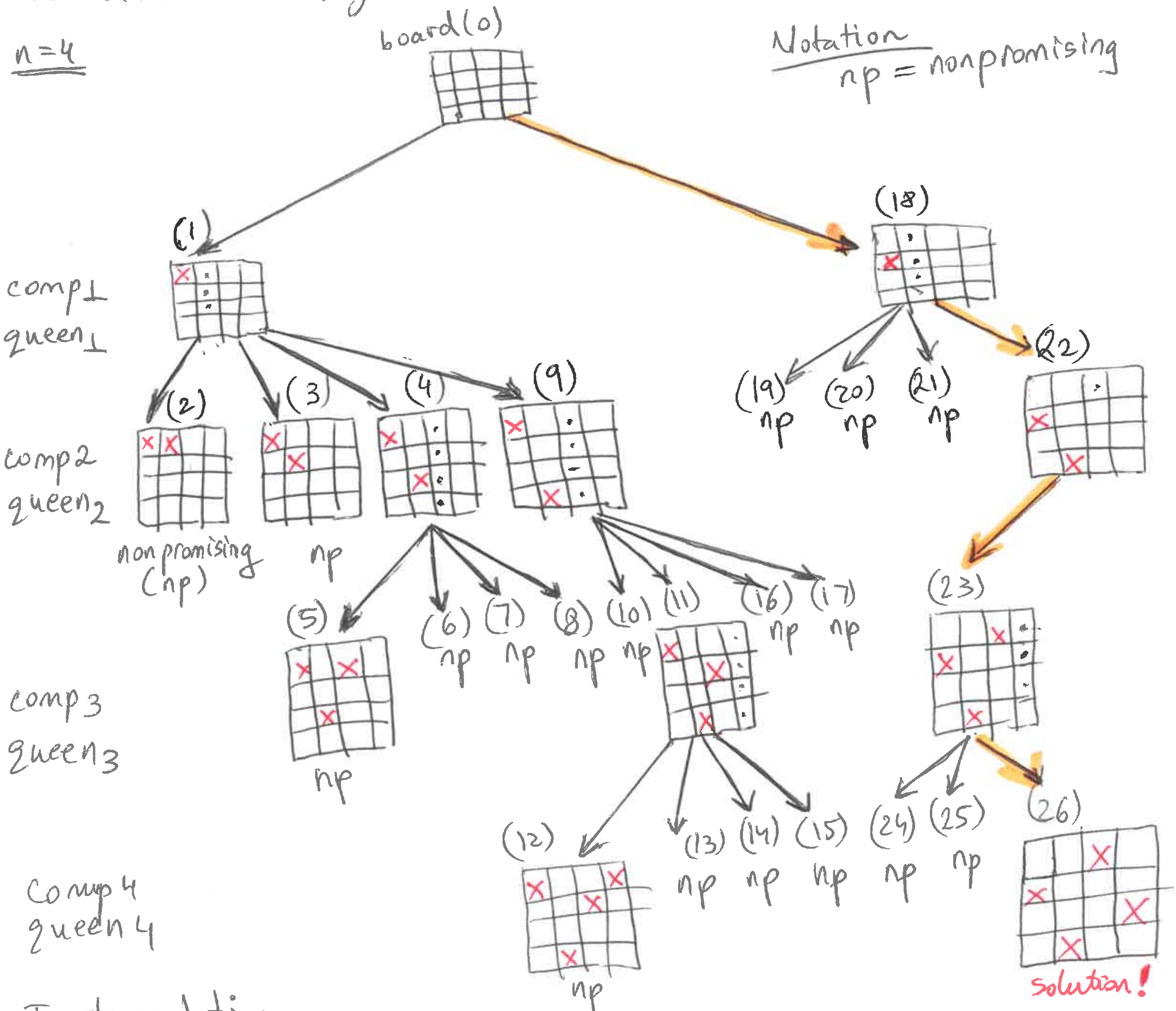
strategy

- place queens successively in columns, left to right
- for each column, work from top to bottom

board (0) – empty board
board (i) – board generated after i attempts to place queens

<u>n=4</u>

<u>Notation</u>
np = nonpromising



<u>Implementation</u>

n_queens(n)
rn_queens(k, n)
position_OK(k, n)

rn_queens(k, n)

- attempts to place a queen in column k
- assume that queens have been already placed in columns 1, 2, ..., k−1 (non conflicting with each other)
- we distinguish two cases:
  - if a queen is placed successfully in column k (i.e. it does not conflict with queens 1, 2, .., k−1), then recursively call rn-queens(k+1, n)
  - otherwise, if it fails to place a queen in column k, then it backtracks one level, trying a new location for the queen in column k−1

array row[1..n]

row[k] − row where the queen in column k is placed

<u>example</u>

n = 4

	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]

solution: 2, 4, 1, 3

position_OK(k, n)

- return true if the queen in column k does not <u>conflict</u> with the queens in columns 1, 2, ..., k−1
- return false otherwise

- conflict if queens are placed on the same row

array row_used[1..n]

row_used[r] — true if a queen has been placed on row r
— false otherwise

example

let $k=4, n=5$

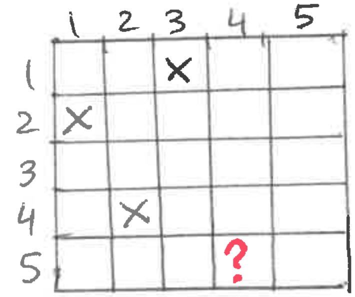
row

1	2	3	4	5
2	4	1	5	

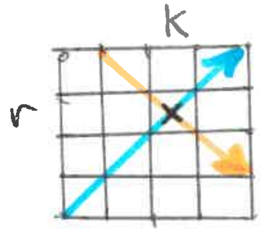
row_used

1	2	3	4	5
T	T	F	T	F

k=4

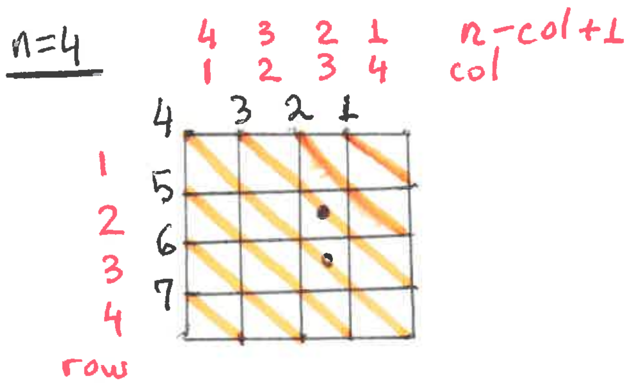


- conflict if queens are placed on the same diagonal



row column
(r, k)

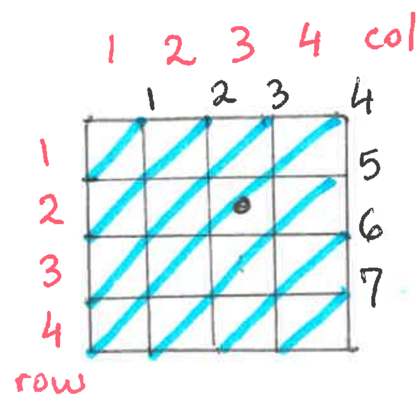
ddiag – downward diagonal with direction ↘



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

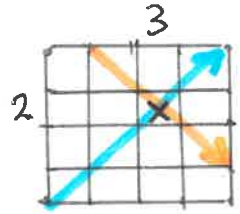
– if a queen is placed at (r, k), then the following diagonals become occupied:

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

Note that $r+(n-k+1)-1 = r+n-k$

example

$n=4$



- a queen placed at (2,3) is in udiag 4 and ddiag 3

udiag $r+k-1 = 2+3-1=4$

ddiag $r+n-k = 2+4-3=3$

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 = 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
x x ?	$k=3$	$\leq n\cdot(n-1)$ times
x x x ?	$k=4$	$\leq n\cdot(n-1)\cdot(n-2)$ times
	...	
	$k=n$	$\leq n\cdot(n-1)\cdot(n-2)\cdots 2$ times

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

$$RT \leq n\cdot\left(1+n+n(n-1)+n(n-1)(n-2)+\ldots+n(n-1)(n-2)\cdots 2\right)=$$

$$=n\cdot n!\left(\frac{1}{n!}+\frac{n}{n!}+\frac{n(n-1)}{n!}+\frac{n(n-1)(n-2)}{n!}+\ldots+\frac{n(n-1)(n-2)\cdots 2}{n!}\right)=$$

$$=n\cdot n!\left(\underbrace{\frac{1}{n!}+\frac{1}{(n-1)!}+\frac{1}{(n-2)!}+\frac{1}{(n-3)!}+\ldots+\frac{1}{1!}}_{e-1}\right)$$

$$\boxed{\sum_{i=0}^{\infty}\frac{1}{i!}=e}$$

$e=2.718\ldots$

$$\frac{1}{1}+\underbrace{\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots}=e$$

$$RT \leq n\cdot n!\cdot(e-1)$$

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