Gauss → 92 solutions to 8 Queens
“methodisches Tatonieren”

Place Queens \((Q, r)\):
Print all possible ways to
Place queens in rows \(r\) thru \(n\)
given locations \(Q[1..r-1]\) of queens
in rows \(1..r-1\).
\textbf{PLACEQUEENS}(Q[1..n], r):
if \( r = n + 1 \)
\hspace{1cm} \textbf{print} Q[1..n]
else
\hspace{1cm} for \( j \leftarrow 1 \) to \( n \)
\hspace{2cm} legal \leftarrow \text{TRUE}
\hspace{2cm} for \( i \leftarrow 1 \) to \( r - 1 \)
\hspace{3cm} \textbf{if} (Q[i] = j) \textbf{or} (Q[i] = j + r - i) \textbf{or} (Q[i] = j - r + i)
\hspace{3cm} legal \leftarrow \text{FALSE}
\hspace{2cm} \textbf{if} legal
\hspace{3cm} Q[r] \leftarrow j
\hspace{1cm} \text{PLACEQUEENS}(Q[1..n], r + 1) \quad \text{(Recursion!)}

\textbf{Figure 2.2.} Gauss and Laquière's backtracking algorithm for the \( n \) queens problem.
A game state = positions of all pieces + who goes next

A game state is good iff
- current player has already won, or
- there is a move leaves opponent with a bad game state.
**PLAYAnyGAME**({\textit{X, player}}):

if \textit{player} has already won in state \textit{X}

return \textbf{Good}

if \textit{player} has already lost in state \textit{X}

return \textbf{Bad}

for all legal moves \textit{X} \rightarrow \textit{Y}

if \text{PLAYAnyGAME}(\textit{Y, \neg player}) = \textbf{Bad}

return \textbf{Good}

\text{\{X} \rightarrow \textit{Y} \text{ is a good move\}}

return \textbf{BAD}

\text{\{There are no good moves\}}
**Interprets**

**Is Word(w) \( \rightarrow \) T/F**

This algorithm repeats subproblems.

Memoize \( \rightarrow \) remember results of each subproblem
**SPLITTABLE(A[1..n]):**

if $n = 0$
    return True

for $i \leftarrow 1$ to $n$
    if IsWORD(A[1..i])
        if SPLITTABLE(A[i+1..n])
            return True

return False

**SPLITTABLE(i):**

if $i > n$
    return True

for $j \leftarrow i$ to $n$
    if IsWORD(i, j)
        if SPLITTABLE(j+1)
            return True

return False

$T(n) = O(n^2) + \sum_{i=1}^{n-1} T(n-i)$

= $O(2^n)$

---

— What do you need to remember about the past?
— What problem are solving to make future decisions?

Recurse!