HWS out — due next Tue

• Arithmetic takes time

```python
result = 1
for i in 1 to n:
    result = result + result
```

• Recursion sometimes requires generalization.

Median: $\varnothing \overset{m}{\rightarrow}\varnothing$
methodisches Tattonieren

brute force

"recursion"
\begin{algorithm}
\textbf{PLACEQUEENS}($Q[1..n], r$):
\hspace{1em} \text{if } r = n + 1 \\
\hspace{2em} \text{print } Q[1..n] \\
\hspace{1em} \text{else} \\
\hspace{2em} \text{for } j \leftarrow 1 \text{ to } n \\
\hspace{3em} \text{legal } \leftarrow \text{TRUE} \\
\hspace{3em} \text{for } i \leftarrow 1 \text{ to } r - 1 \\
\hspace{4em} \text{if } (Q[i] = j) \text{ or } (Q[i] = j + r - i) \text{ or } (Q[i] = j - r + i) \\
\hspace{4em} \text{legal } \leftarrow \text{FALSE} \\
\hspace{3em} \text{if legal} \\
\hspace{4em} Q[r] \leftarrow j \\
\hspace{2em} \text{PLACEQUEENS}($Q[1..n], r + 1$) \quad (\text{Recursion!})
\end{algorithm}

\textbf{Figure 2.2.} Gauss and Laqui\`ere's backtracking algorithm for the $n$ queens problem.
**PLAYAnyGAME(\(X, \text{player}\)):**

- if \(\text{player}\) has already won in state \(X\)
  
  return \text{Good}

- if \(\text{player}\) has already lost in state \(X\)
  
  return \text{Bad}

- for all legal moves \(X \rightarrow Y\)
  
  if \(\text{PLAYAnyGAME}(Y, \neg \text{player}) = \text{BAD}\)
  
  return \text{Good}  \quad \text{\(\langle X \rightarrow Y \text{ is a good move} \rangle\)}

- return \text{BAD}  \quad \text{\(\langle \text{There are no good moves} \rangle\)}
Input: string $A(1..n)$

Fixed subroutine $\text{IsWord}(w) \rightarrow \text{true}$ if $w$ is a "word"
$\rightarrow \text{false}$

Question: Can we partition $A$ into a sequence of words?

What problem are you really solving?
What do we need to remember about past decisions?
What are the recursive subproblems?
Is there a simple encoding of subproblems?
**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

---

\[
T(n) \leq O(n) + \sum_{i=1}^{n-1} T(i)
\]

\[
= O(2^n)
\]