If we do many many searches, how to minimize total search time?

If node x is a more frequent search than y, we want x to have the lower depth.
If some nodes are searched for much more frequently, best tree could have depth $\Omega(n)$. 
Optimal Binary Search Tree

Given:
1) A sorted array \( A[1..n] \) of keys for the nodes.
2) An array \( f[1..n] \) of access frequencies.

We search for \( A[i] \) a total of \( f[i] \) times.

Goal: Build best static BST to minimize total search time.
For a given BST $T$, let $v_1, v_2, \ldots, v_n$ be its nodes in sorted order so $v_i$ stores $A[i]$, 

$$\text{Cost}(T, f[1..n]) := \sum_{\omega = 1}^{n} \max(\text{ancestors of } v_\omega \text{ in } T, \text{nodule's depth} + 1) \quad (\text{root has one ancestor: itself})$$
binary search tree:

\[ \text{root} \]

\[ \text{right subtree} \]

\[ v_r \text{, root node} \]

If \( \omega = r \), all ancestors of \( v_\omega \) except \( v_r \) are in left subtree.

\[
\text{Cost} (T, f[I..n]) = \sum_{\omega=1}^{n} f[I_\omega] \cdot 1
\]

\[
+ \sum_{\omega=1}^{n} f[I_\omega] \cdot \left( \# \text{ancestors in left}(T) \right)
\]

\[
+ \sum_{\omega=1}^{n} f[I_\omega] \cdot \left( \# \text{ancestors in right}(T) \right)
\]
\[
\text{Cost}(T_j \cup C_{1..n}) = \sum_{i=1}^{n} \text{Cost}(\text{left}(T_j \cup C_{1..n-1})) + \text{Cost}(\text{right}(T_j \cup C_{r+1..n}))
\]

\[
\text{Cost}(T_j \cup C_{1..0}) = 0
\]
Opt\(\text{Cost}(i, k)\): optimal cost for BST over A[i..k].

\[
\text{Opt}\text{Cost}(i, k) =
\begin{cases}
0 & \text{if } i > k \\
\min_{j \in I(i)} \left( \sum_{j=1}^{k} f(j) + \text{Opt}\text{Cost}(i, r-1) + \text{Opt}\text{Cost}(r+1, k) \right) & \text{otherwise}
\end{cases}
\]

Final goal: Compute Opt\(\text{Cost}(1, n)\).
\[ F(i, k) : = \begin{cases} 0 & \text{if } i > k \\ F(i, k-1) + f(k) \text{ otherwise} \end{cases} \]

Goal: Fill array \( F[1..n] \). \( O(n^2) \) subproblems in \( O(1) \) time each.

\[
\begin{array}{ccc}
\text{INITF}(f[1..n]): & \text{INITF}(f[1..n]): & \text{INITF}(f[1..n]): \\
\text{for } i \leftarrow 1 \text{ to } n & \text{for } i \leftarrow 1 \text{ to } n & \text{for } i \leftarrow 1 \text{ to } n \\
0 & 0 & 0 \\
F[i, i-1] & F[i, i-1] & F[i, i-1] \\
\text{for } k \leftarrow i \text{ to } n & \text{for } k \leftarrow i \text{ to } n & \text{for } k \leftarrow i \text{ to } n \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{cases} 0 & \text{if } i > k \\ \text{Opt}\text{Cost}(i, k) \text{ otherwise} \end{cases}
\]
Memoization:

subproblems: \( 1 \leq i \leq n+1 \)
\[ 0 \leq k \leq n \]

memo data structure:

\[ \text{OptCost}[1..n+1, 0..n] \]
$O(n^3)$ time

$O(n^2)$ subproblems

$O(n)$ per problem

$O(n^2)$ with more work!

[Frickson D.3]
Independent Set of a graph: set of vertices that have no shared edges

Max ind set: given a graph, find max size ind. set

REALLY HARD
(for arbitrary input graphs)
So, say we're given a tree $T$.

Root it...

Do we include the root?

If no, find max ind sets in each subtree independently.

If yes... find sets in grandchild subtrees!
MIS(v): size of MIS in subtree rooted at v
w\rightarrow v: "w is a child of v"

Next time: how to memoize!