(00:30)
**The Minimax Principle** (Ch 2)

Game Tree Evaluation (2.1)
- n round deterministic 2-player game
- Players take turns making a 1-bit choice
- At end winner is some function of choices made

We can represent all the possible states and outcomes of such a game with a Game Tree.

Where each node represents a game state (root = start, leaves = end) and contains a value 0 or 1 to represent that there exists a winning strategy for the first or second player, where if they make the correct choice every time they will always be able to reach some leaf node-end state where they win.

Given the leaf nodes of any game tree it is possible to solve for every node up to the root node recursively, by looking at which player would have the ability to choose at each level.

There's always a winning strategy for one of the two players at every state. This can be shown inductively using a tree with 1 root node and 2 leaf nodes as a base case and then by doing the following.
Given a game tree of height \( n \) whose two subtrees of height \( n-1 \) both have winning strategies for one of the two players. If either subtree has a winning strategy for the first player

The first player will choose to go down a winning subtree and have a winning strategy. Else

The second player has a winning strategy regardless of the first player’s decision.

Label node 1 if player whose turn it is has winning strategy. Label node 0 otherwise. Label \( (\text{node}) = \text{NAND} \text{(labels(children))} \)

---

(07:45)

**NAND-Tree Evaluation**

Height \( n \) binary tree with labels at leaves. This tree can be thought of as a circuit with NANDs at the non-leaf nodes for the purpose of evaluating the output of the root.

Query Complexity

Algorithm queries for values at leaves. 

Cost \( \text{(algorithm)} = \# \text{ queries.} \)

---

(12:25)

**Deterministic Algorithms**

Requires \( 2^n = 4^{n/2} \) queries on leaf nodes.

When evaluating a node’s first child keep the node value as 1 to signify that we don’t yet know what the outcome is for the NAND node. This makes the algorithm correctly evaluate both children recursively.

Inductively we force the algorithm to evaluate every single leaf. \( 2^{n-1} \) for each child.

---

(15:00)

**NP Algorithms**

Have hints(someone telling you the winning strategy, but not the adversaries choices) about which leaves to query, but need to prove the answer Only \( 2^{n/2} \) queries needed to evaluate part of game tree that comes from the winning strategy for the first player, because their first decision will immediately cut out the other \( 2^{n/2} \) leaf nodes from possibility.

---

(17:30)

**Randomized Algorithm**
Pick a random child of root and recursively evaluate
If child evaluates to 0
   Done, return is a winning strategy
Else
   Evaluate other child and return based on that.

\[ W_n = E[\text{queries for tree of height } n, \text{ in which root is winning}] \]
\[ L_n = E[\text{queries for tree of height } n, \text{ in which root is losing}] \]

\[ W_n = 2W_{n-1} \]
\[ L_n \leq L_{n-1} + (1/2)W_{n-1} \]

\[ L_n = 2W_{n-1} \leq 2L_{n-2} + W_{n-2} \]
\[ W_n \leq L_{n-1} + (1/2)W_{n-1} \leq 2W_{n-2} + (1/2)L_{n-2} + (1/4)W_{n-2} \]

Something\(n \leq 3\)Something\(n-2\)

Therefore, \(E[\text{queries}] \leq 3^{n/2}\)

---

(26:00)

**Lower Bounds**

Game between
Algorithm - Trying to figure out answer with as few queries as possible.
Input A - Wants to force Algorithm to need as many queries as possible.

For deterministic algorithm, adversary just abuses mechanics of algorithm to maximize queries with a designed bad input.
Randomness helps here because instead of needing just 1 bad input, the adversary now needs inputs that are bad on average for the randomization.

**ROCK PAPER SCISSORS** - Matrix Games

<table>
<thead>
<tr>
<th></th>
<th>Rock</th>
<th>Paper</th>
<th>Scissors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rock</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Paper</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>Scissors</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
A zero sum game, both players have diametrically opposed goals. One player can only win by beating the other.

Deterministic strategies are bad, because there is always an input that completely beats it.

Randomized algorithms can be used to always guarantee you a chance of success.
A randomized algorithm is simply a probability distribution over which deterministic algorithm to play. (rock, paper, scissors) Assuming adversary can't know your randomness they can no longer reliably beat you.

(36:20)
**Matrix M**

Randomized algorithm $\leftrightarrow$ probability vector $P$

$P$ tells us probability of playing each rock, paper or scissors.
Payoff against input $i = \sum p_i \cdot M_{i,j} = p^T Me_j$

(37:35)
**Minimax Strategy** - Finding best worst case outcome

$Sup = \text{supremum}$
$Inf = \text{infimum}$
$Sup_p \cdot \text{Min}_j p^T Me_j = Sup_p \cdot Inf_q \cdot p^T M_q$

Algorithm picks strategy $p$, Adversary picks strategy $q$. Goal of algorithm is to pick a $p$ such that the best possible $q$ would be as good for the algorithm as possible.

(40:00)
**Thrm Von Neumann (2.2)**

$p$ and $q$ are probability vectors.
$Sup_p \cdot Inf_q \cdot p^T M_q = Inf_q \cdot Sup_p \cdot p^T M_q = Inf_q \cdot Max_{e_i} \cdot e_i^T M_q$

Moral: We can prove optimal lower bounds by finding an input distribution such that ANY deterministic algorithm has bad average case analysis.

To prove lower bounds don't find bad inputs, find bad input distributions.

(46:00)
NAND Tree Evaluation

Pick $P = (\sqrt{5} - 1) / 2$.

Each leaf independently equals 1 with probability $P$.

$P^2 = 1 - p$ implies all nodes in the tree are 1 with probability $p$.

Prop 2.7

If leaves are iid independent, then the best deterministic algorithm is a depth first pruning algorithm.

$W(n) = W(n-1) + P \cdot W(n-1) = (1 + P)^n = (1.616...)^n$ which is the golden ratio.

Random solution evaluates out to $(1.41...)^n$, so by using the random algorithm we can on average beat out the deterministic algorithm.