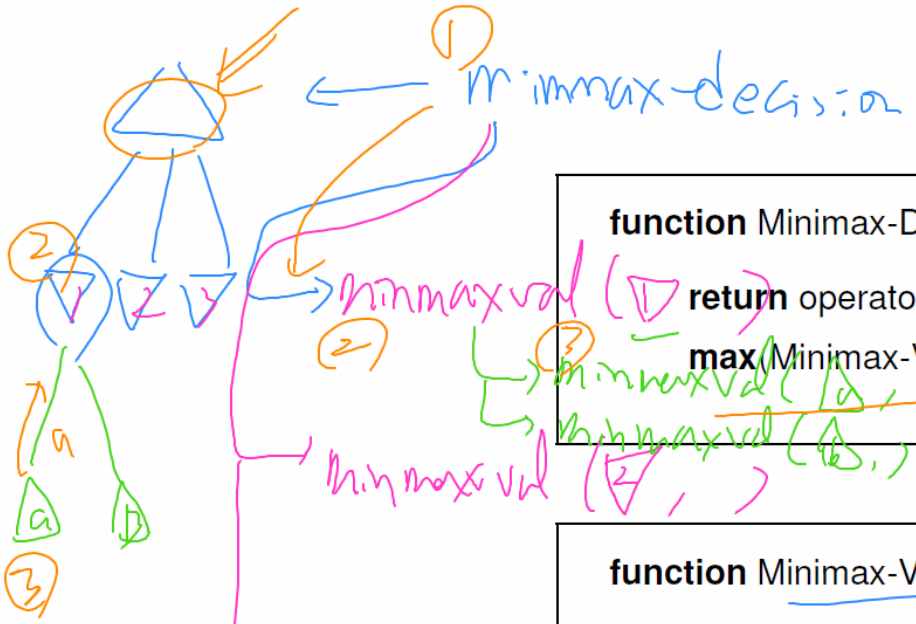


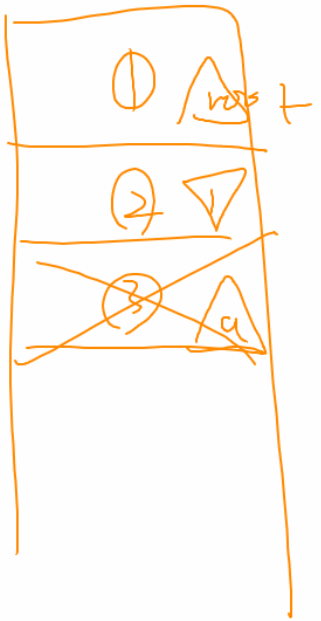
Minimax Decision

function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)
else if Max's move **then**
→ **return** max of successors' Minimax-Value
else
→ **return** min of successors' Minimax-Value



Minimax Decision

function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

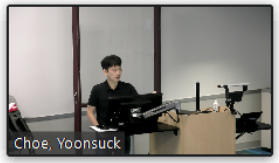
if Goal(state), **return** Utility(state)

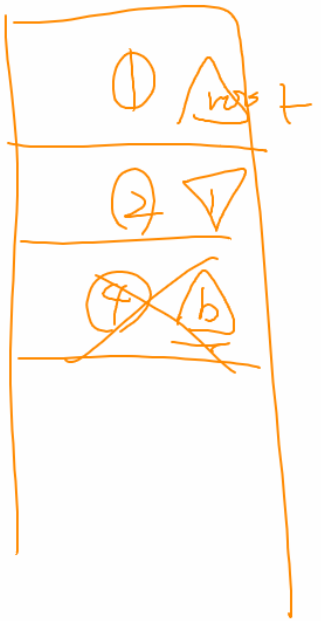
else if Max's move **then**

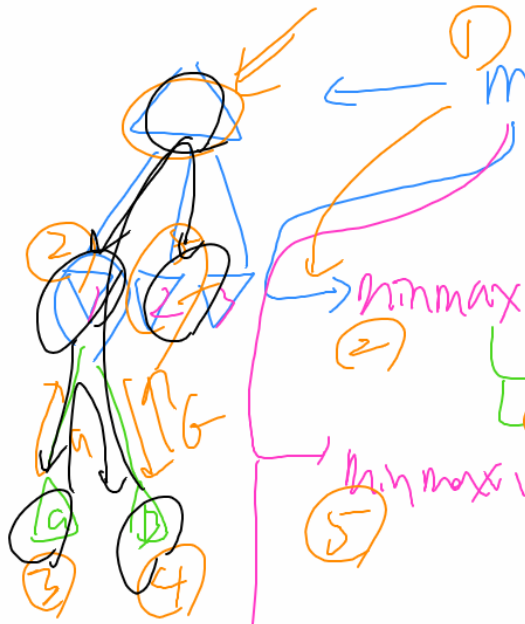
→ **return** max of successors' Minimax-Value

else

→ **return** min of successors' Minimax-Value







Minimax Decision

function Minimax-Decision (game) **returns** operator

→ **return** operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

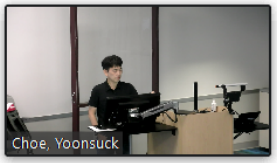
if Goal(state), **return** Utility(state)

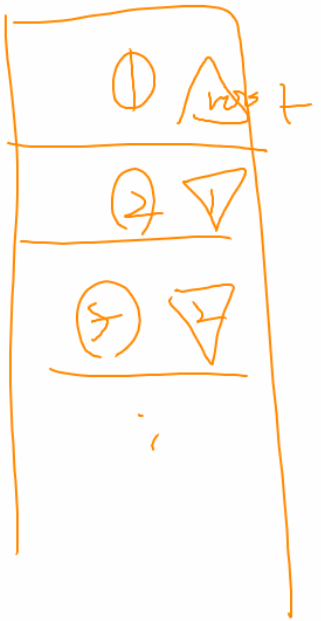
else if Max's move **then**

→ **return** max of successors' Minimax-Value

else

→ **return** min of successors' Minimax-Value





So here and then to here, then to here, so that's basically a depth, first exploration due to the way that this whole recursive thing is implemented

Minimax Decision

function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)

else if Max's move **then**

→ **return** max of successors' Minimax-Value

else

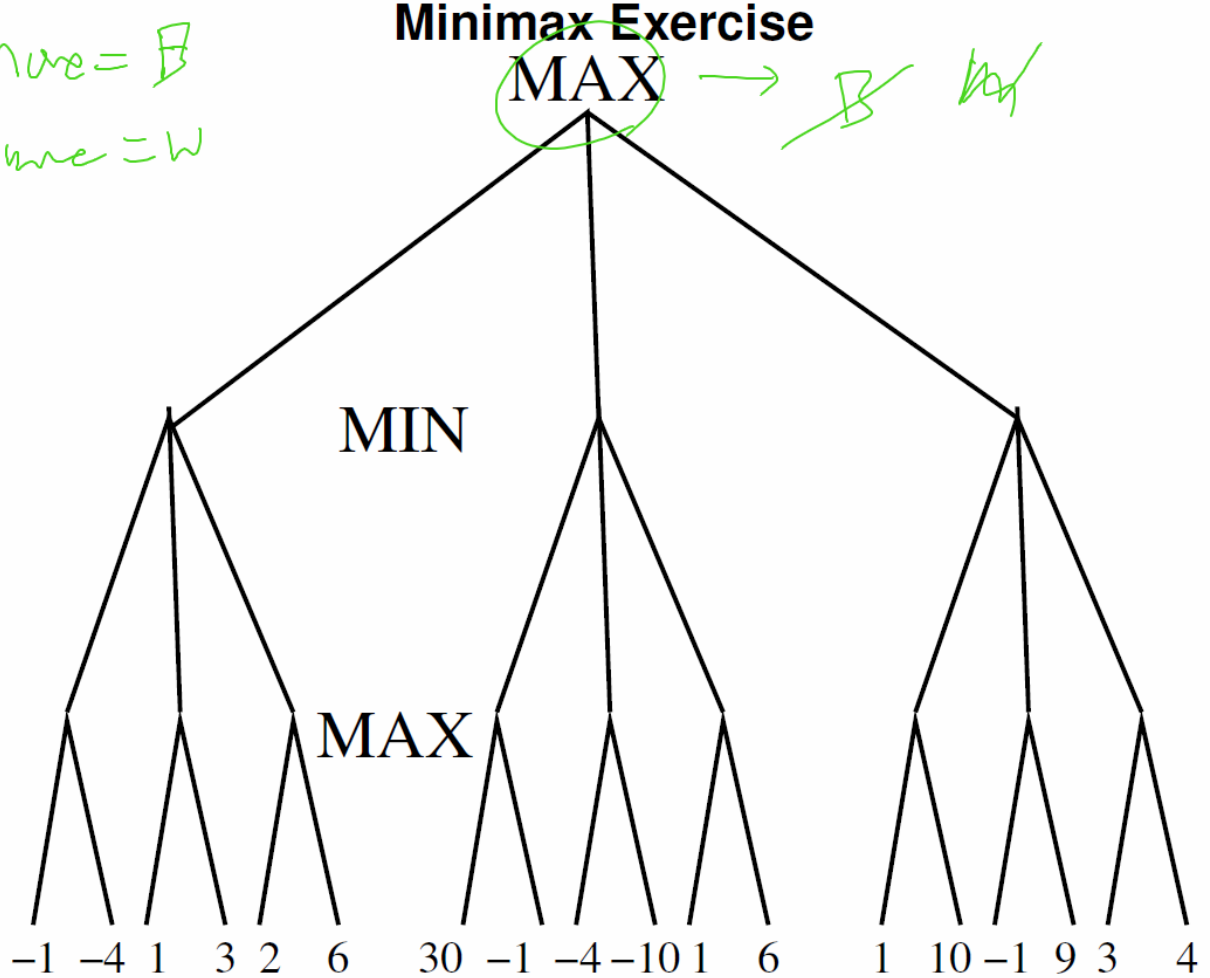
→ **return** min of successors' Minimax-Value

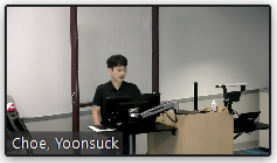
of the successors and return the min value. This is where the distinction between situations like that and like that

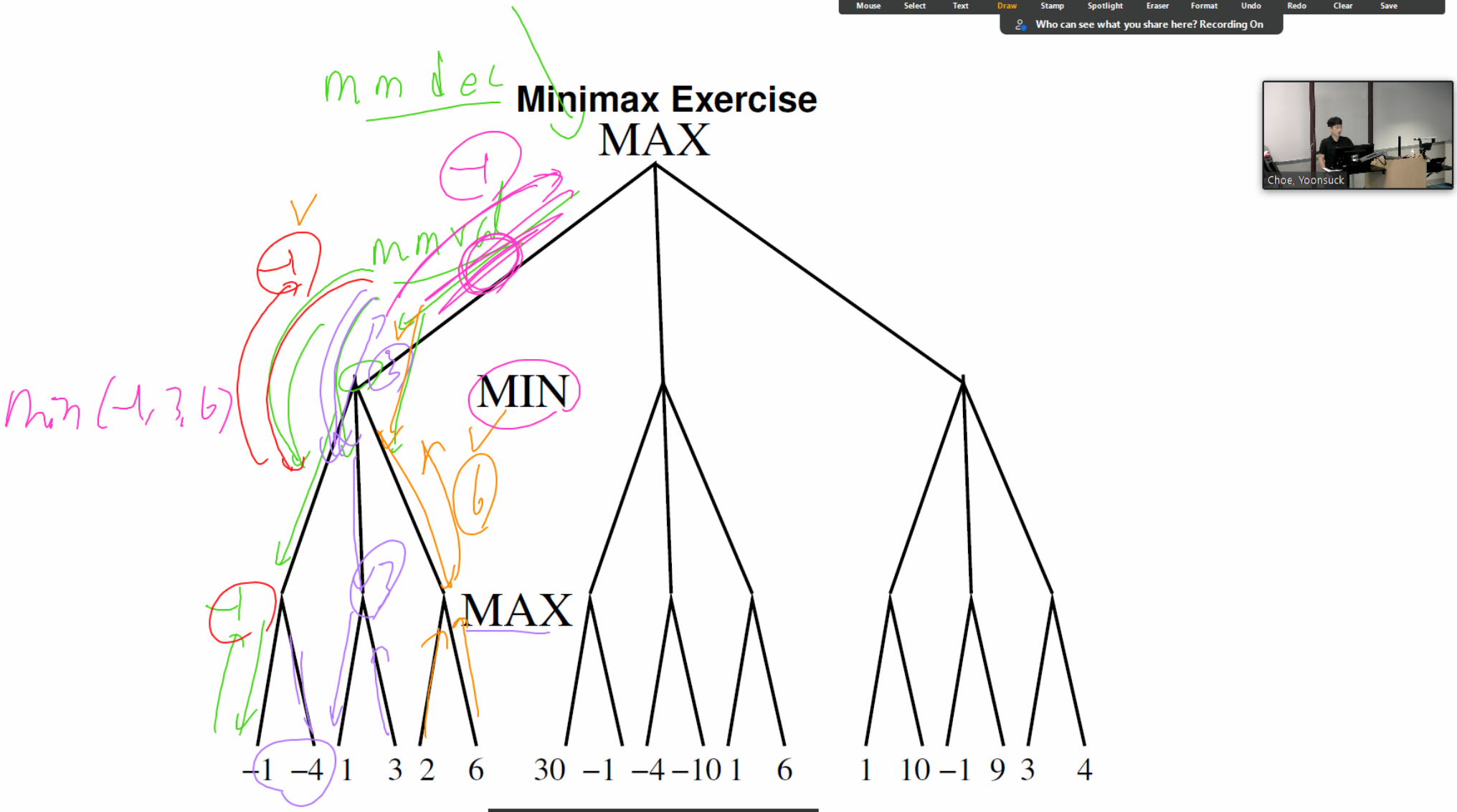
F. Move = B

E. me = W

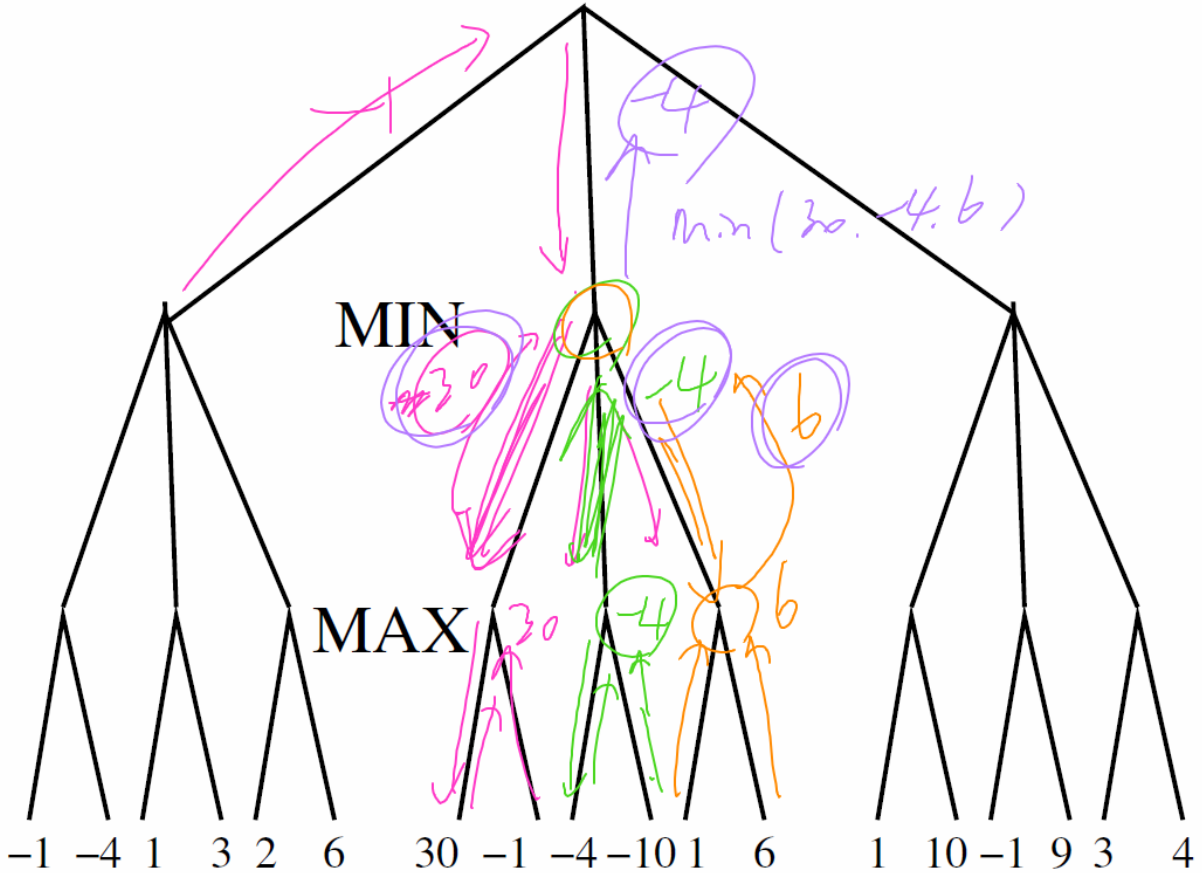
Minimax Exercise

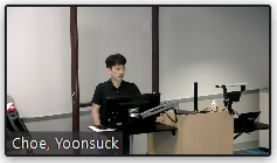






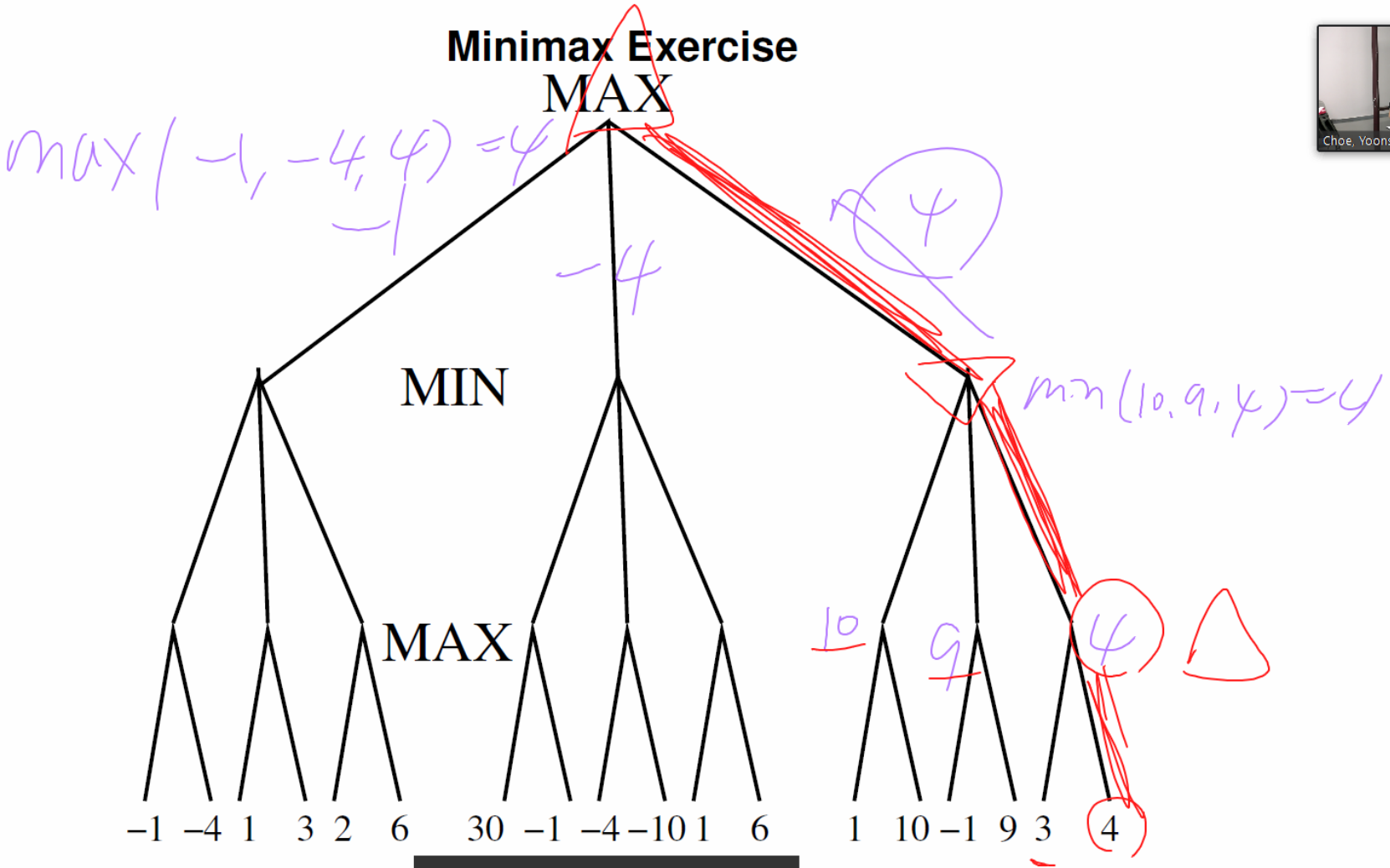
Minimax Exercise

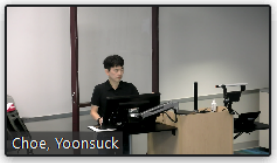




Oh, 30, as 4 and 6 b minus 4,

Minimax Exercise



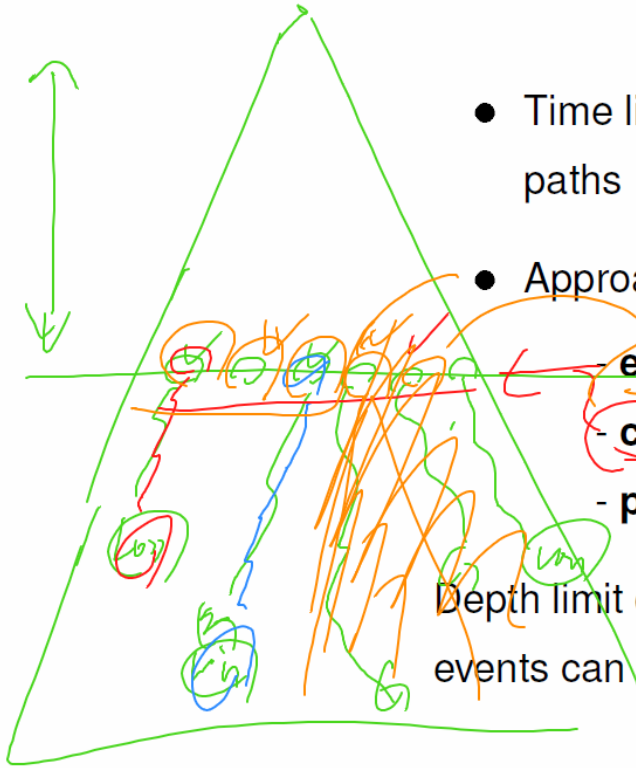


You'll pick 4, so that's the solution using this very simple recursive

Minimax Exercise

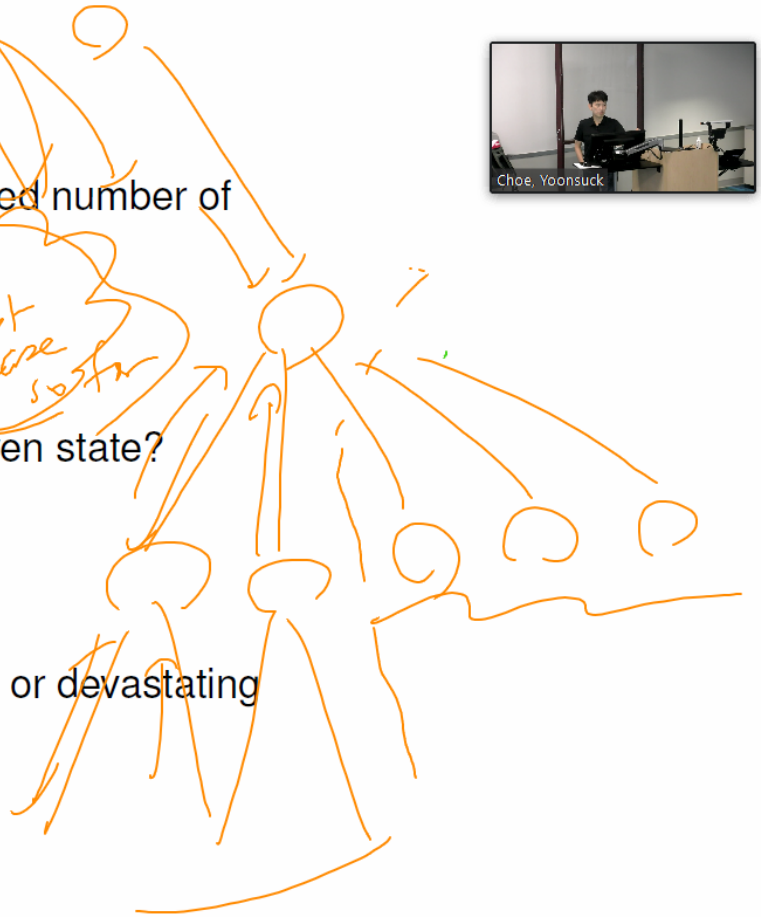
appointment.

Resource Limits



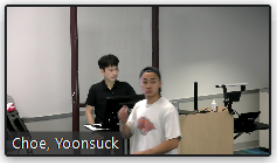
- Time limit: as in Chess → can only evaluate a fixed number of paths
- Approaches:
  - **evaluation function** : how desirable is a given state?
  - **cutoff test** : depth limit
  - **pruning**

Depth limit can result in the **horizon effect**: interesting or devastating events can be just over the horizon!



And suppose that a certain point you decide that whatever you do it's not going to be the the best case scenario so far, which is information that

Minimax Decision



function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

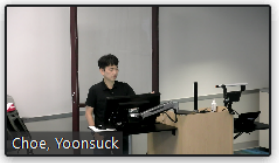
if Goal(state), **return** Utility(state)
else if Max's move **then**
→ **return** max of successors' Minimax-Value
else
→ **return** min of successors' Minimax-Value

Resource Limits

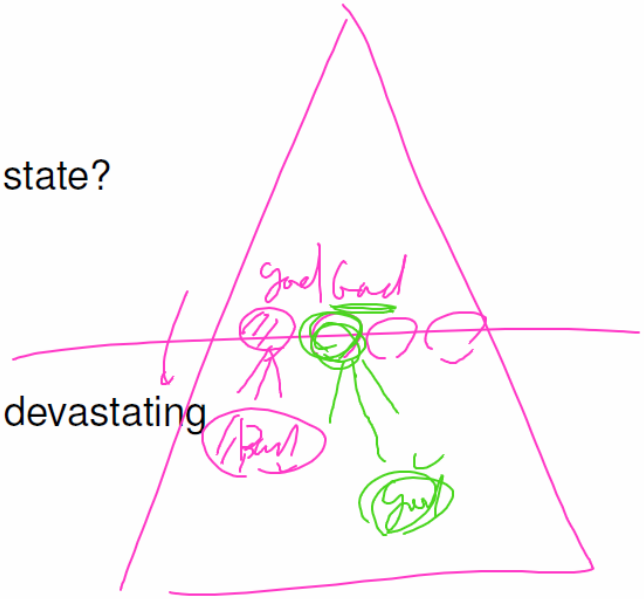
- Time limit: as in Chess → can only evaluate a fixed number of paths
- Approaches:
  - **evaluation function** : how desirable is a given state?
  - **cutoff test** : depth limit
  - **pruning**

Depth limit can result in the **horizon effect**: interesting or devastating events can be just over the horizon!

Resource Limits



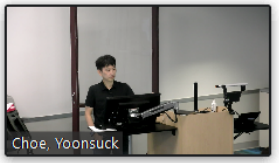
- Time limit: as in Chess $\longrightarrow$ can only evaluate a fixed number of paths
- Approaches:
  - **evaluation function** : how desirable is a given state?
  - **cutoff test** : depth limit
  - **pruning**



Depth limit can result in the **horizon effect**: interesting or devastating events can be just over the horizon!

It looks pretty bad. Okay, but at at a later point can really turn out to be

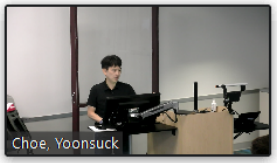
Evaluation Functions



For chess, usually a **linear** weighted sum of feature values:

- $\mathrm{Eval}(s) = \sum_i w_i f_i(s)$
- $f_i(s)$ = (number of white piece X) - (number of black piece X)
- other features: degree of control over the center area
- exact values do not matter: the **order** of Minimax-Value of the successors matter.

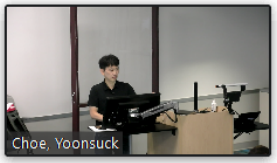
Evaluation Functions



For chess, usually a **linear** weighted sum of feature values:

- $\mathrm{Eval}(s) = \sum_i w_i f_i(s)$
- $f_i(s)$ = (number of white piece X) - (number of black piece X)
- other features: degree of control over the center area
- exact values do not matter: the **order** of Minimax-Value of the successors matter.

Evaluation Functions



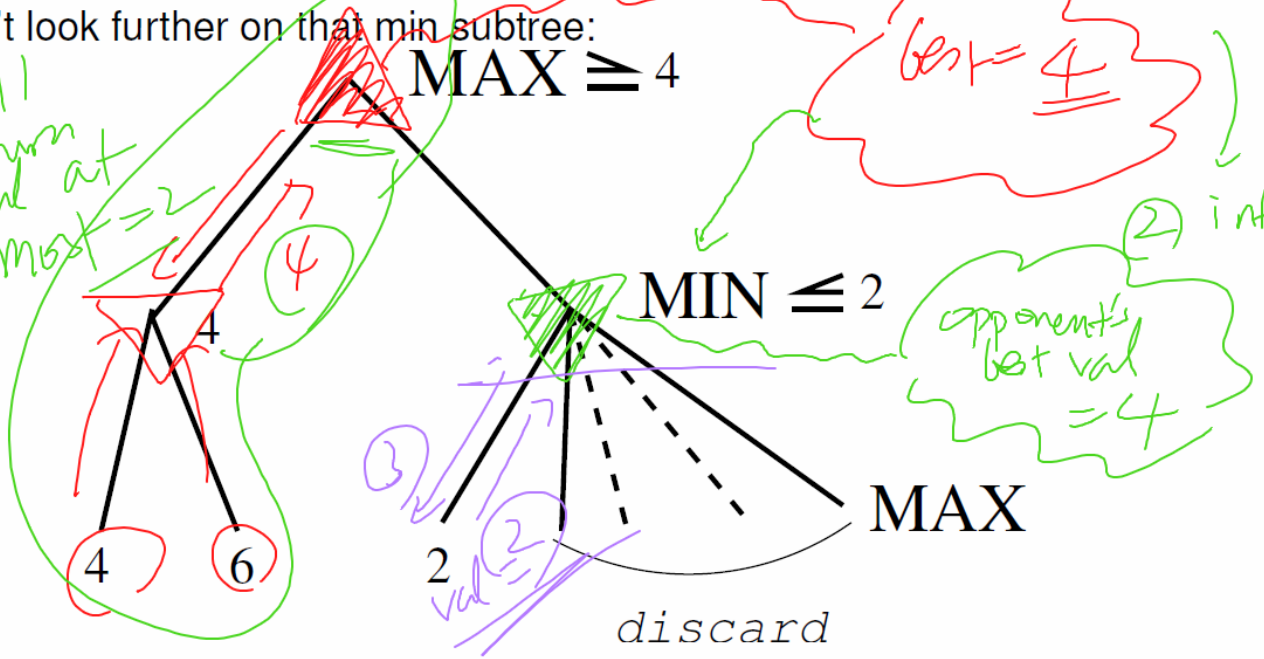
For chess, usually a **linear** weighted sum of feature values:

- $\mathrm{Eval}(s) = \sum_i w_i f_i(s)$ = 10, 20, 30 ✓ = 5, 200, 3000 ✓
- $f_i(s)$ = (number of white piece X) - (number of black piece X)
- other features: degree of control over the center area
- exact values do not matter: the **order** of Minimax-Value of the successors matter.

so you lower it. If the ordering is the same, then for this particular case

α Cuts: α=best MAX so far

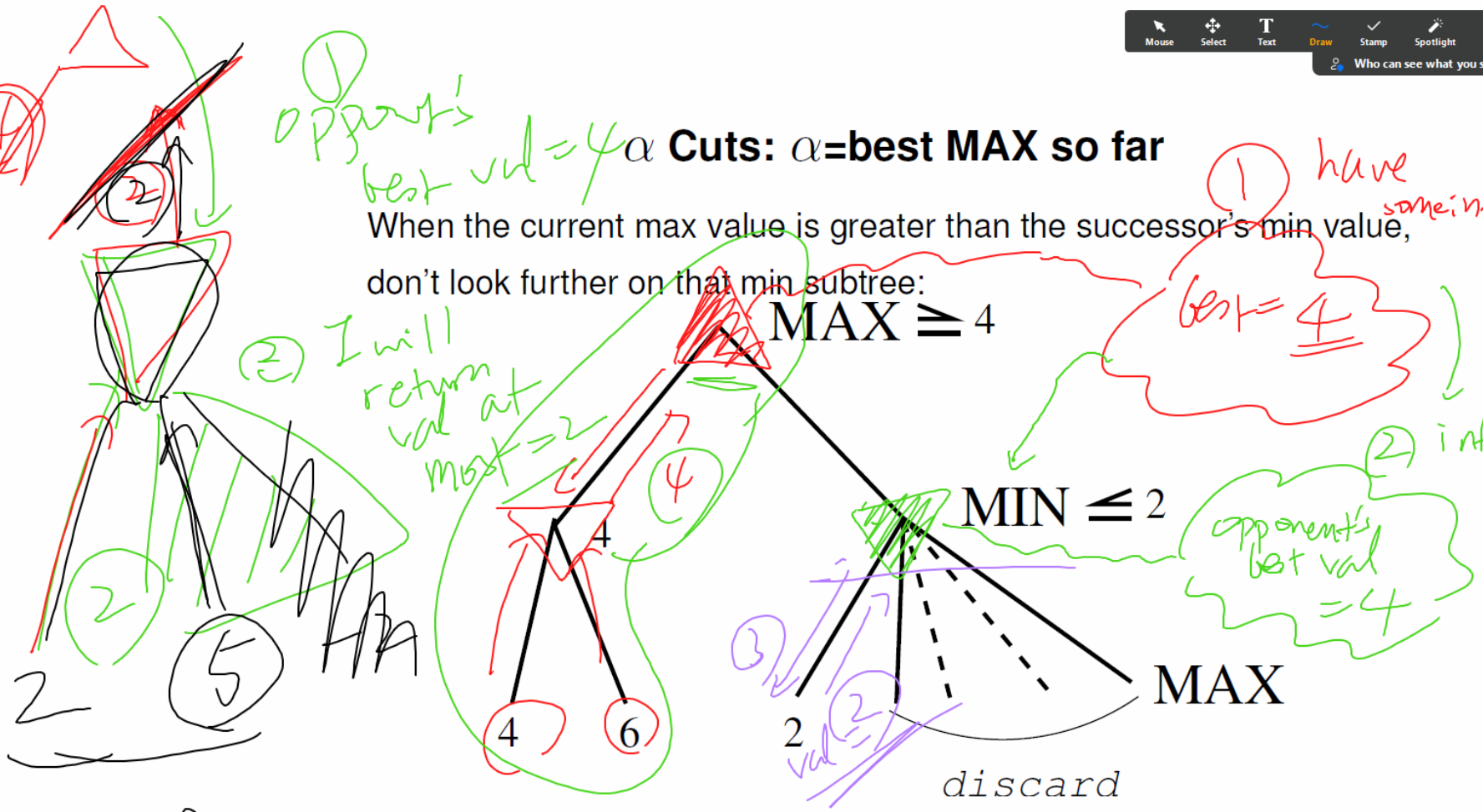
When the current max value is greater than the successor's min value, don't look further on that min subtree:



Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

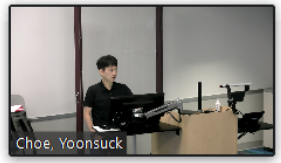
α Cuts: α=best MAX so far

When the current max value is greater than the successor's min value, don't look further on that min subtree:

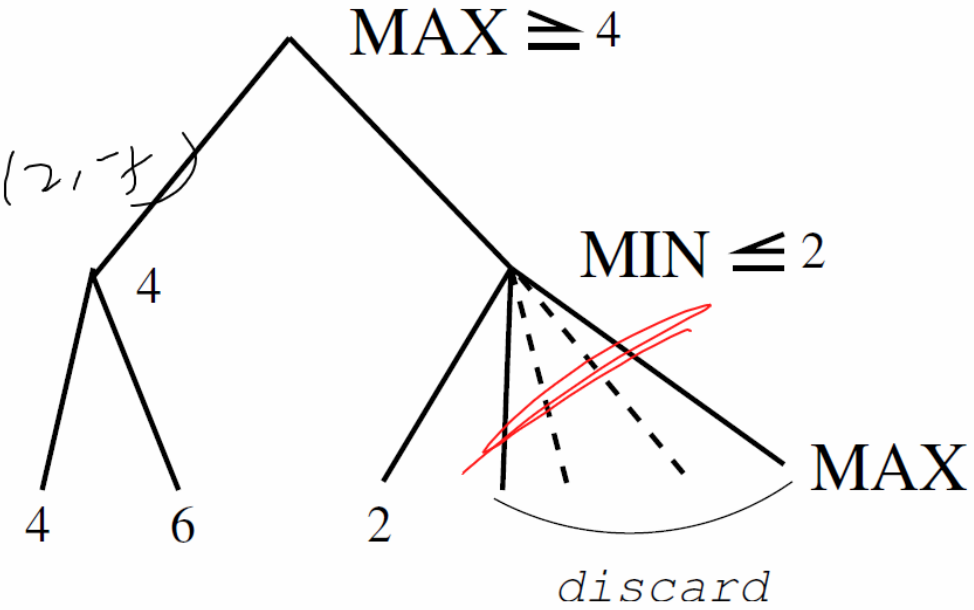


Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

α Cuts: α=best MAX so far



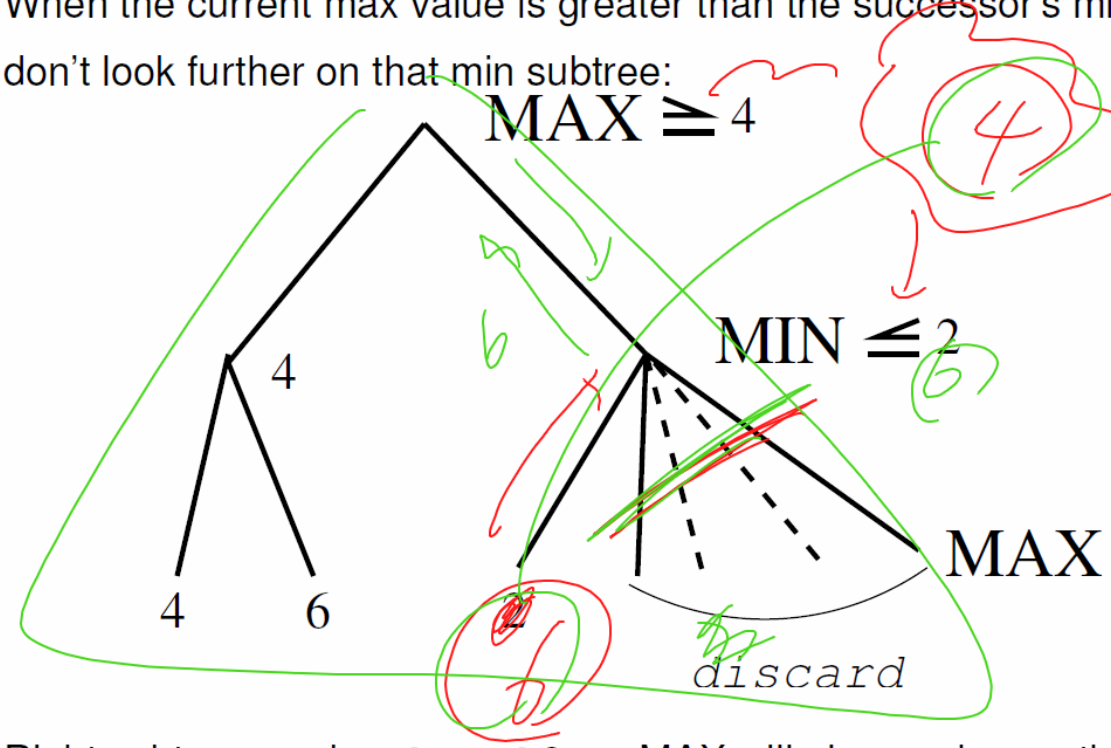
When the current max value is greater than the successor's min value, don't look further on that min subtree:

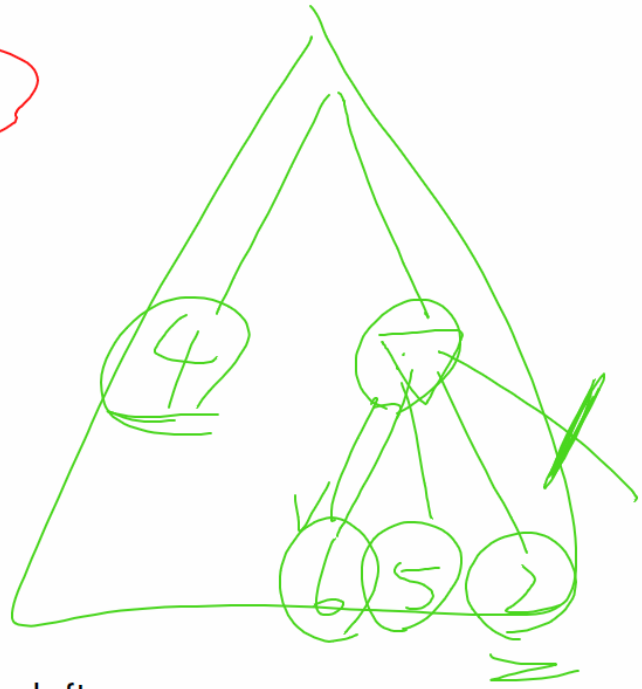


Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

α Cuts: α=best MAX so far

When the current max value is greater than the successor's min value, don't look further on that min subtree:

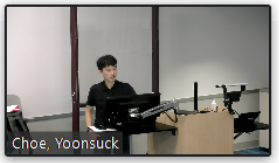




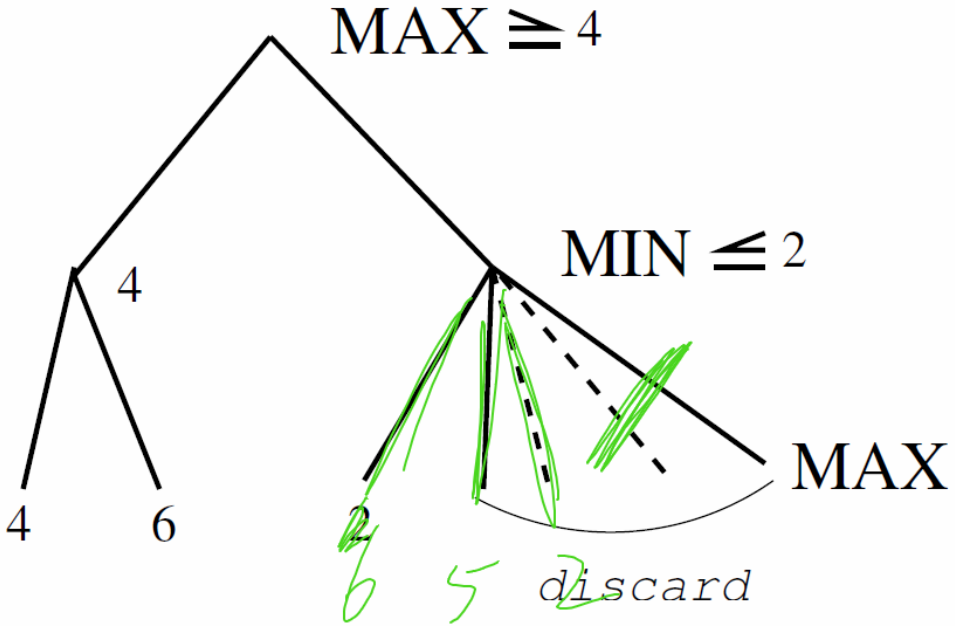
Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

Then you have to keep on going. Then, as soon as you reach that which is worse than that, then you can cut

α Cuts: α=best MAX so far



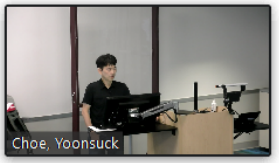
When the current max value is greater than the successor's min value, don't look further on that min subtree:

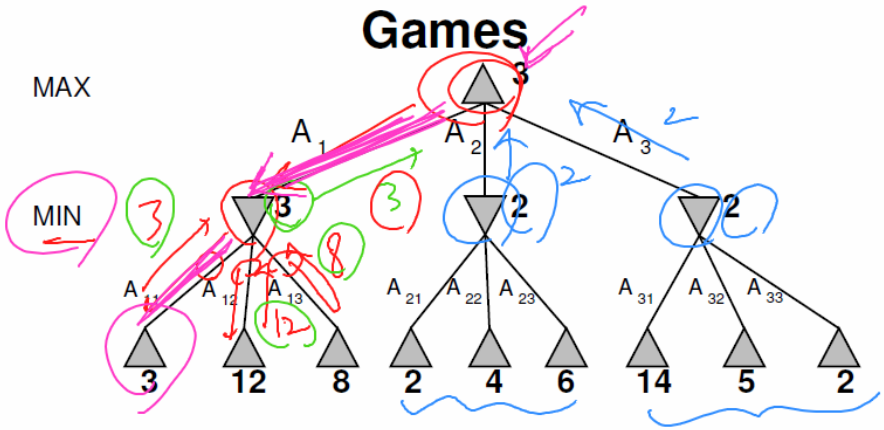


Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

So to summarize. If if this was 6 and 5, and then you have 2, then only after absorbing these 3, then you can cut

Minimax: Strategy for Two-Person Perfect Info Games

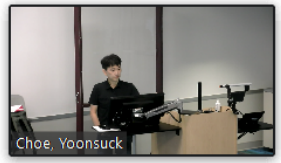




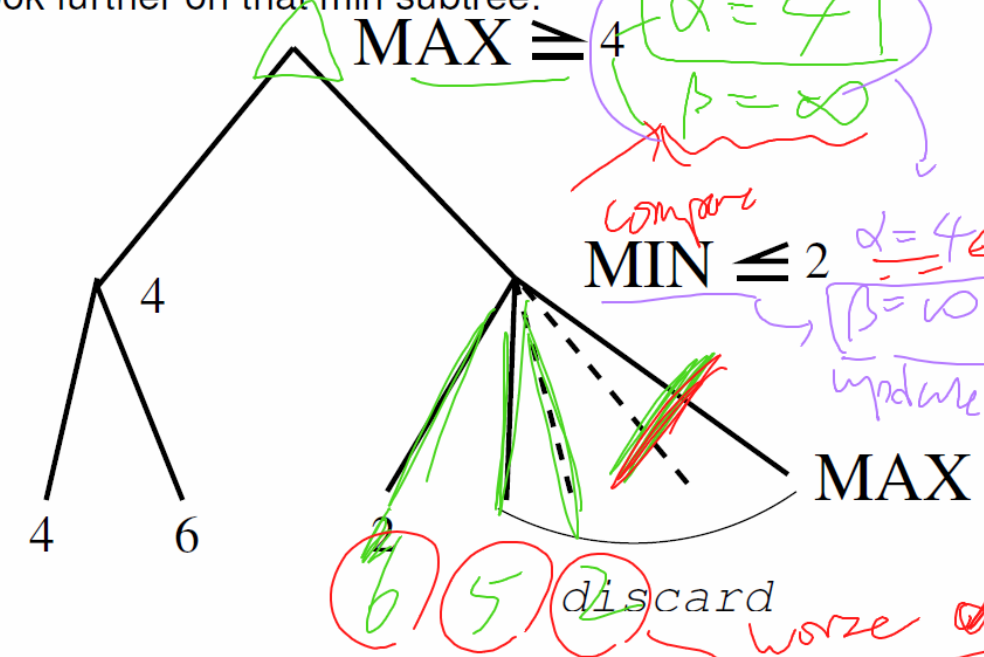
- generate the whole tree, and apply util function to the leaves
- go back upward assigning utility value to each node
- at MIN node, assign **min(successors' utility)**
- at MAX node, assign **max(successors' utility)**
- **assumption**: the opponent acts optimally

Node

α Cuts: α=best MAX so far



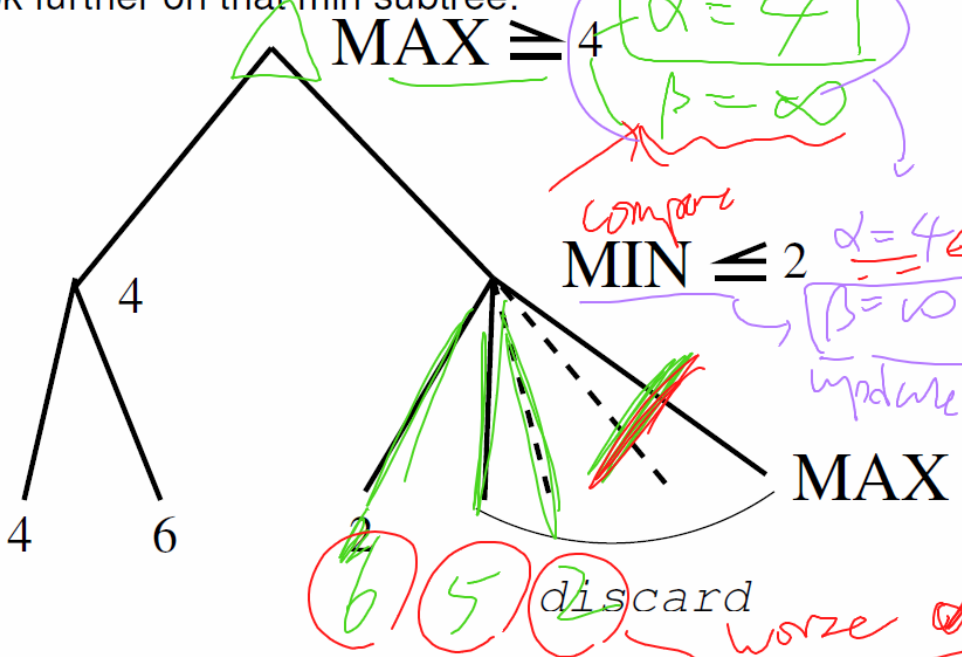
When the current max value is greater than the successor's min value, don't look further on that min subtree:



Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

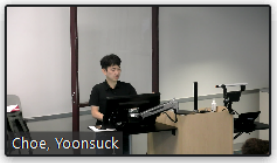
α Cuts: α=best MAX so far

When the current max value is greater than the successor's min value, don't look further on that min subtree:

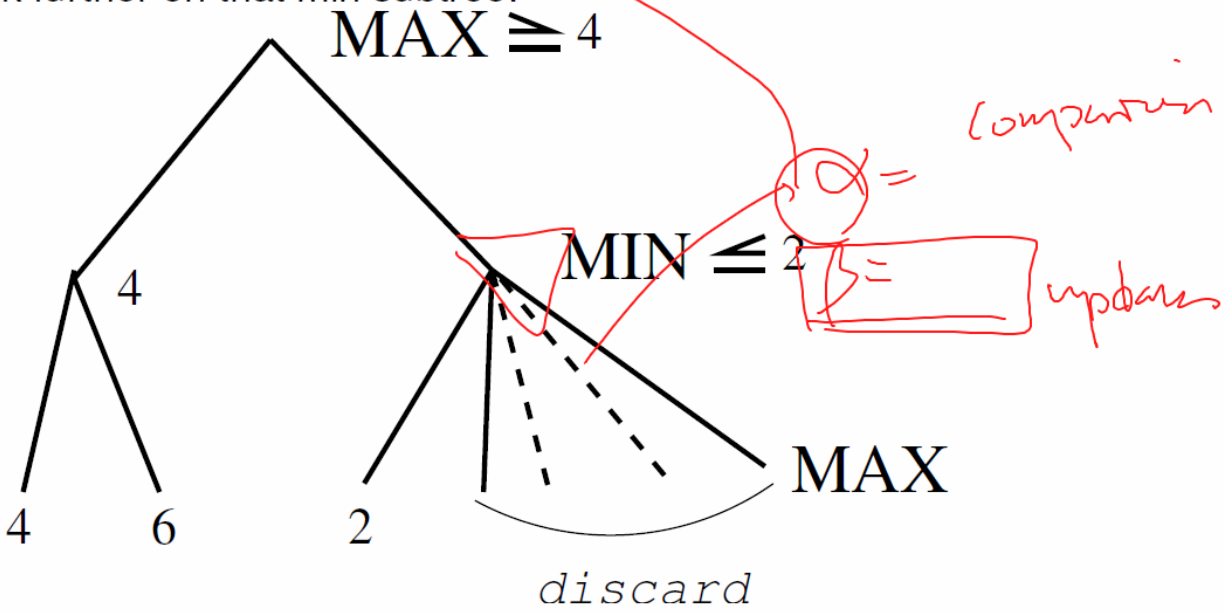


Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

α Cuts: α=best MAX so far

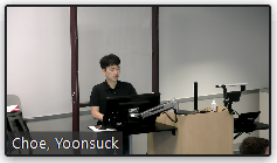


When the current max value is greater than the successor's min value, don't look further on that min subtree:

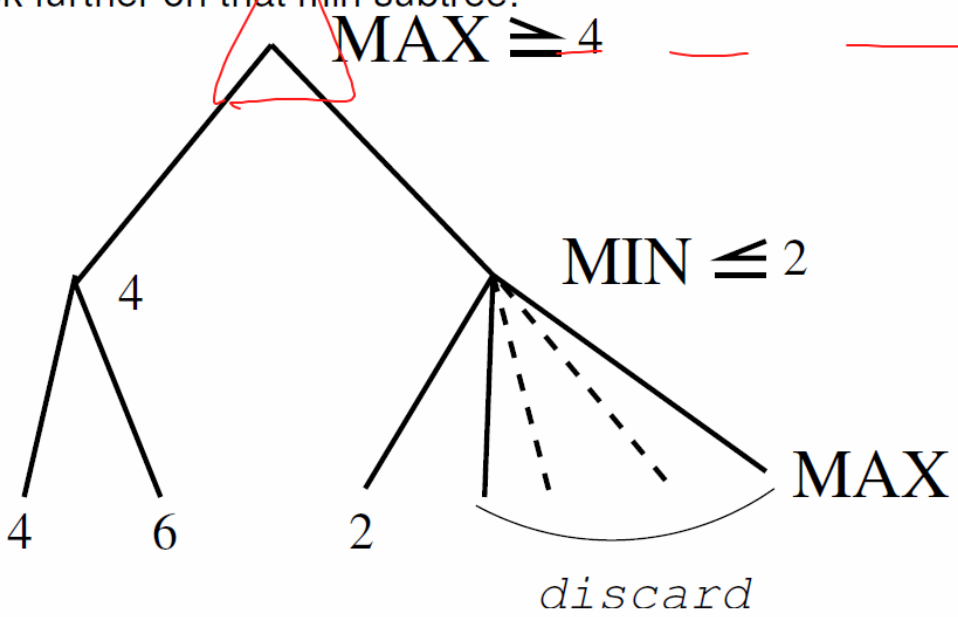


Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

α Cuts: α=best MAX so far



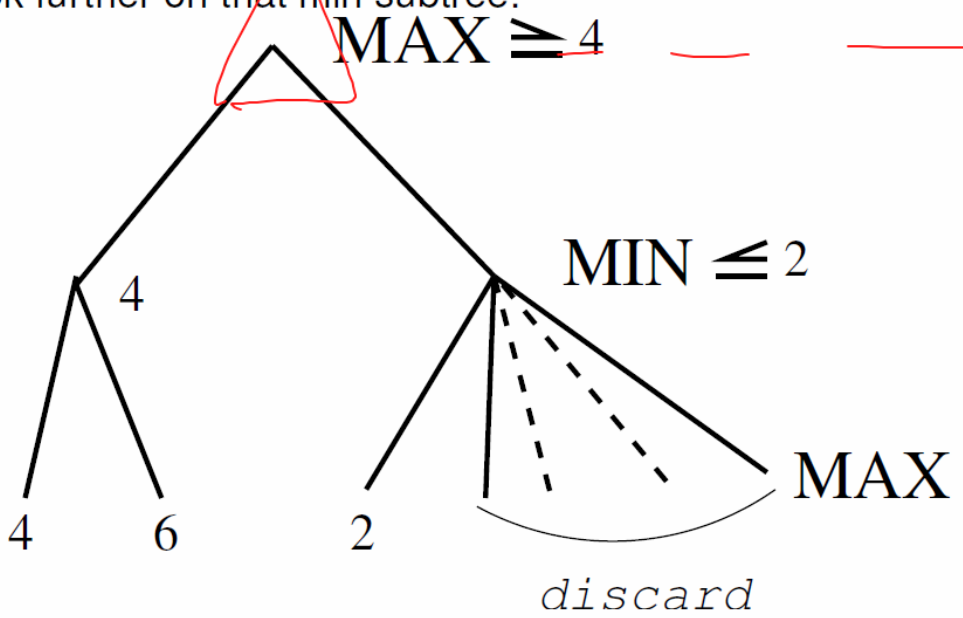
When the current max value is greater than the successor's min value, don't look further on that min subtree:



Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

α Cuts: α=best MAX so far

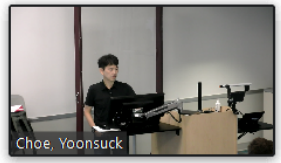
When the current max value is greater than the successor's min value, don't look further on that min subtree:



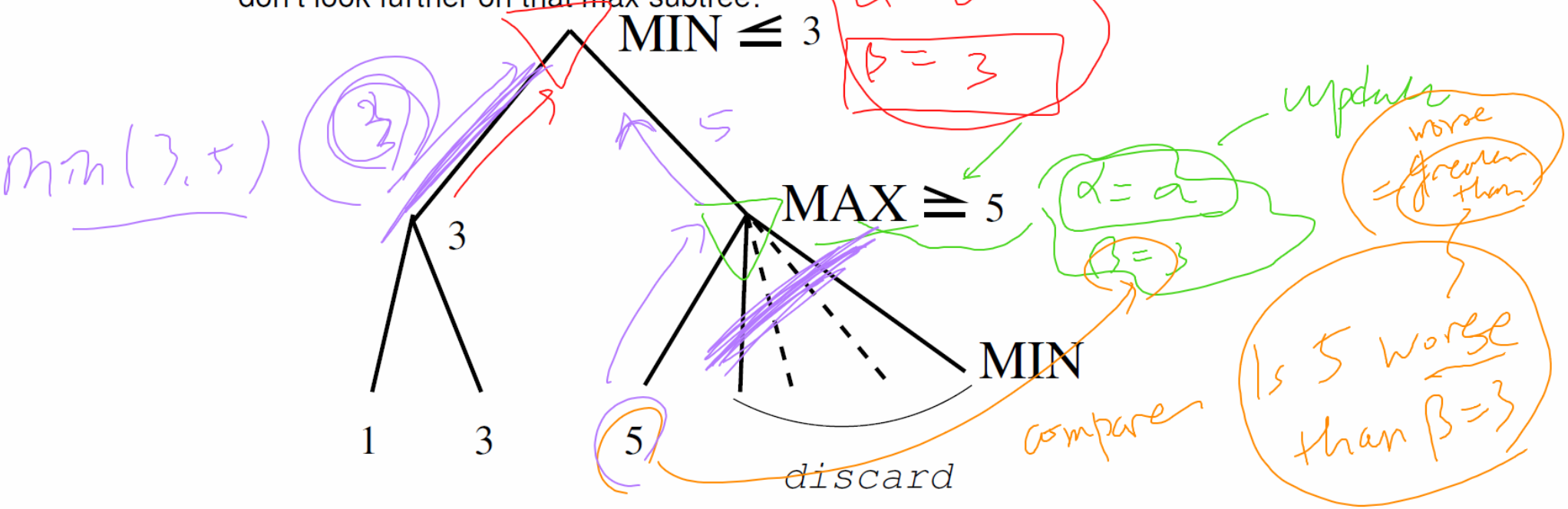
Right subtree can be **at most** 2, so MAX will always choose the left path regardless of what appears next.

Yeah, this is next slide

β Cuts: β=best MIN so far

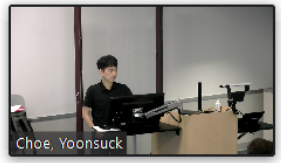


When the current min value is less than the successor's max value, don't look further on that max subtree:

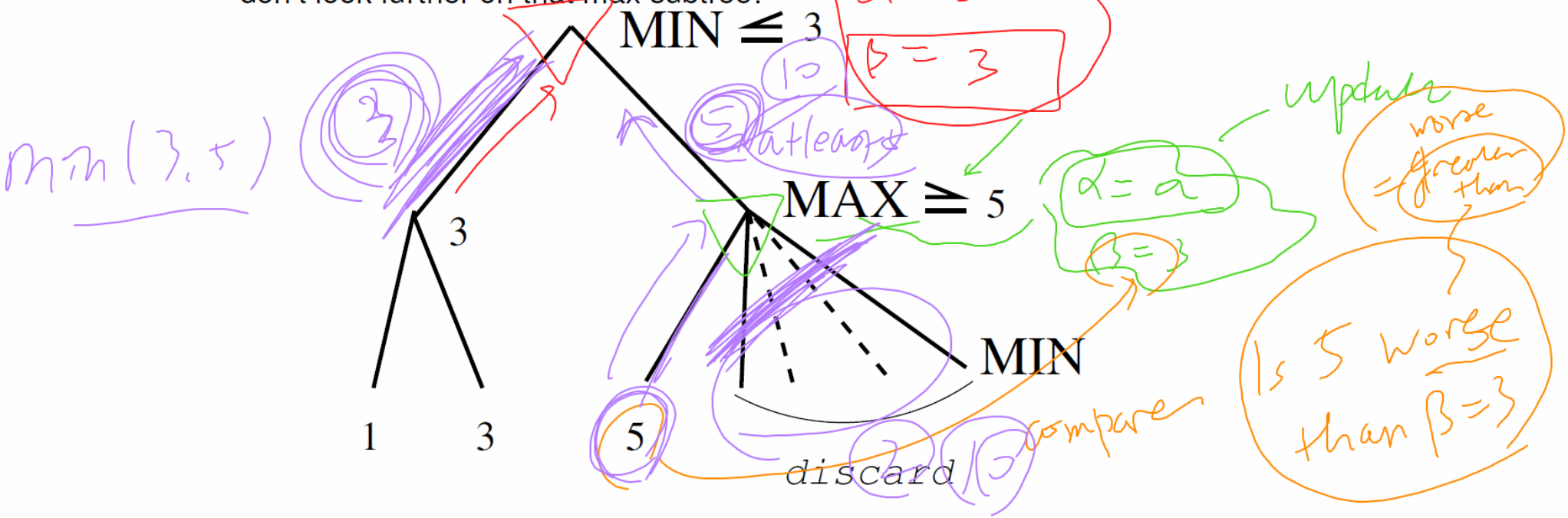


Right subtree can be **at least** 5, so MIN will always choose the left path regardless of what appears next.

β Cuts: β=best MIN so far



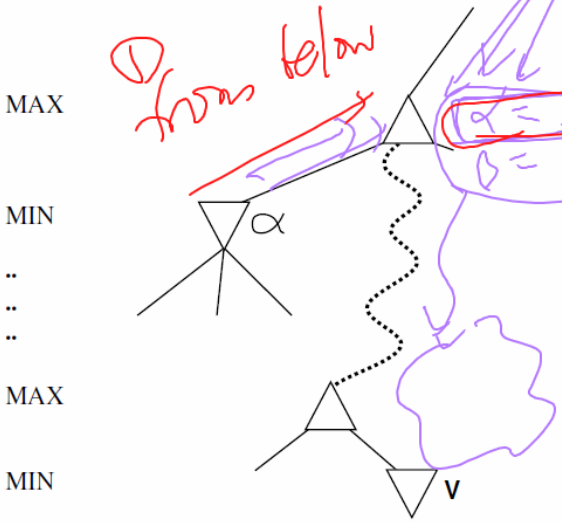
When the current min value is less than the successor's max value, don't look further on that max subtree:



Right subtree can be **at least** 5, so MIN will always choose the left path regardless of what appears next

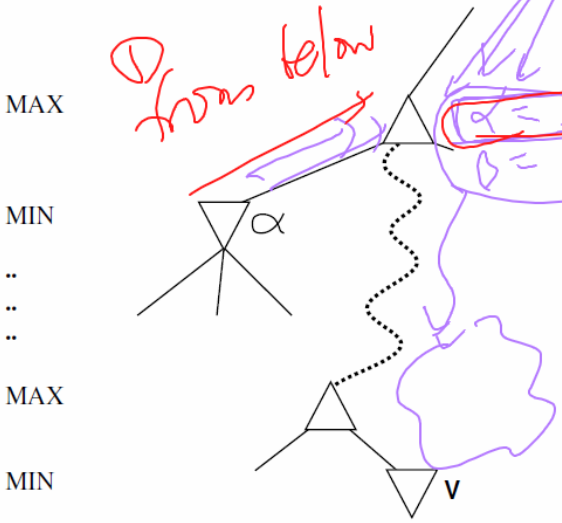
By this point. But time that you encountered this so minimum, 5 value will be returned, and then you mean no component will always just see, I think the other br

$\alpha - \beta$ Pruning



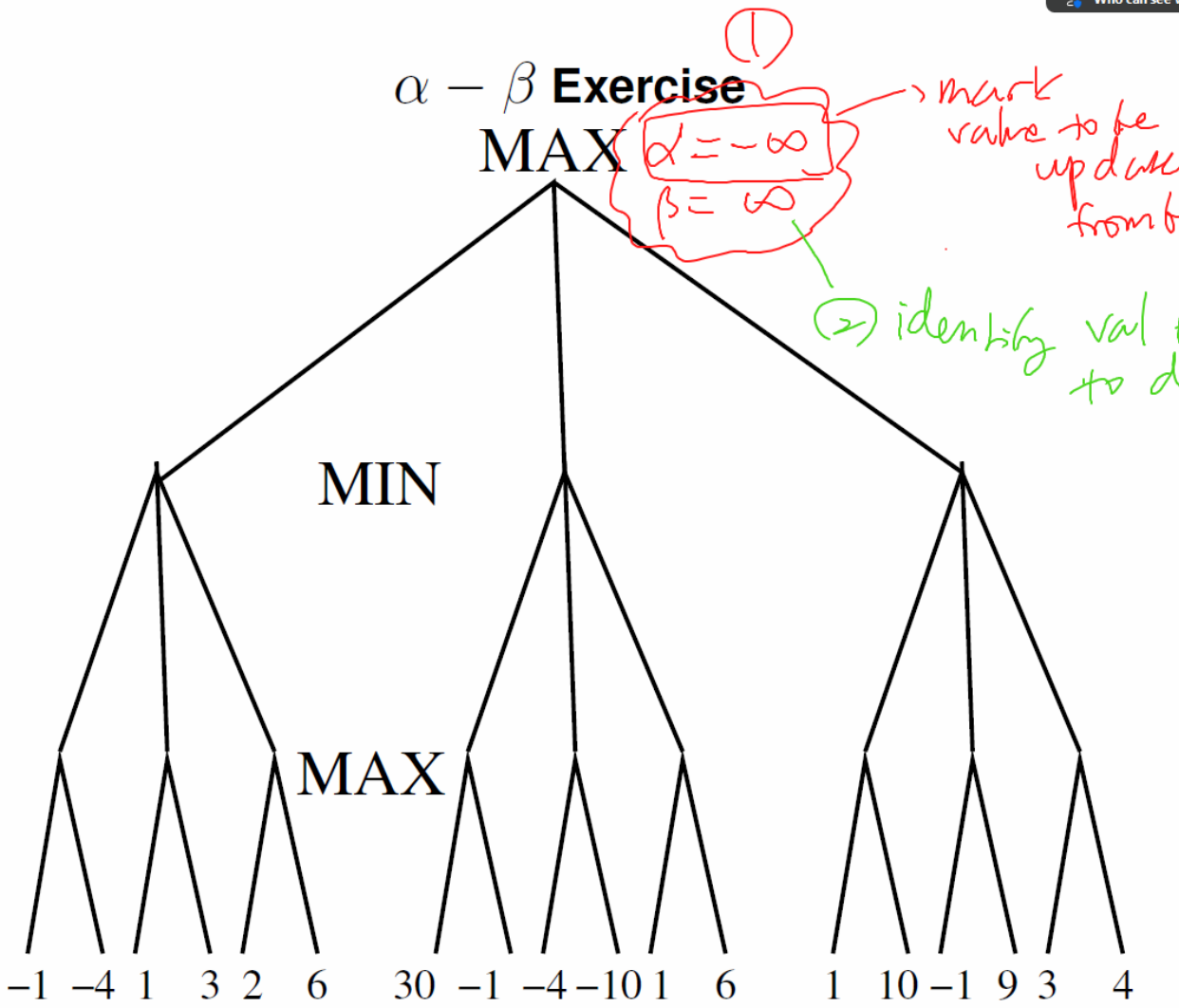
- memory of best MAX value α and best MIN value β
- do not go further on any one that does **worse (!!)** than the remembered α and β

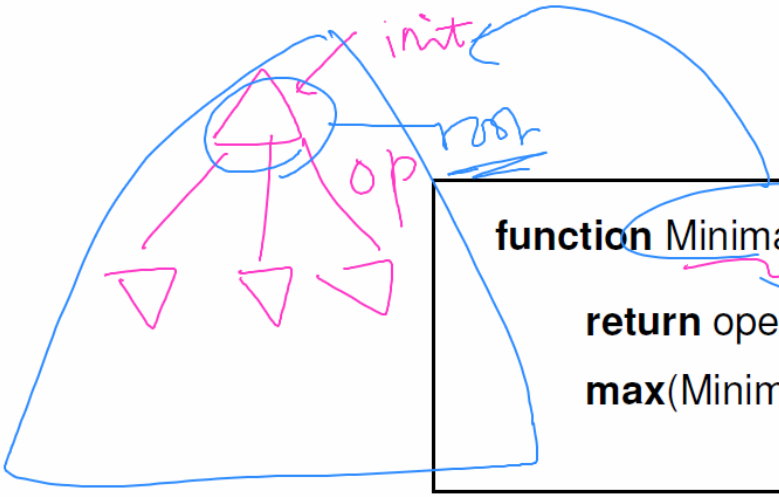
$\alpha - \beta$ Pruning



- memory of best MAX value α and best MIN value β
- do not go further on any one that does **worse (!!)** than the remembered α and β

So far in this entire search, and based on that, you make a decision to cut





Minimax Decision

function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

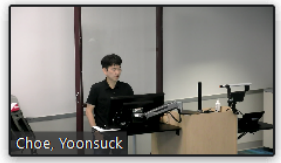
if Goal(state), **return** Utility(state)

else if Max's move **then**

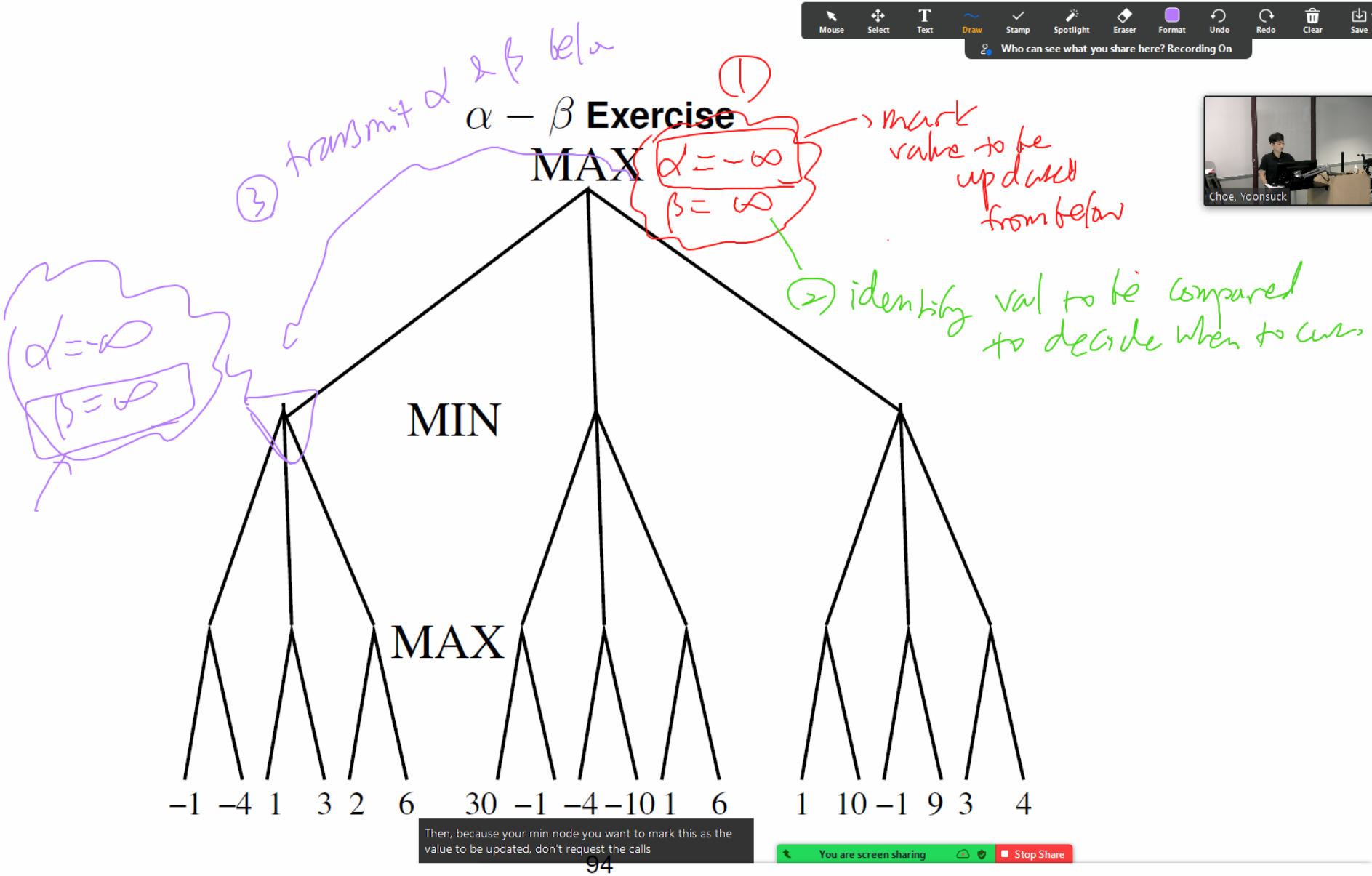
→ **return** max of successors' Minimax-Value

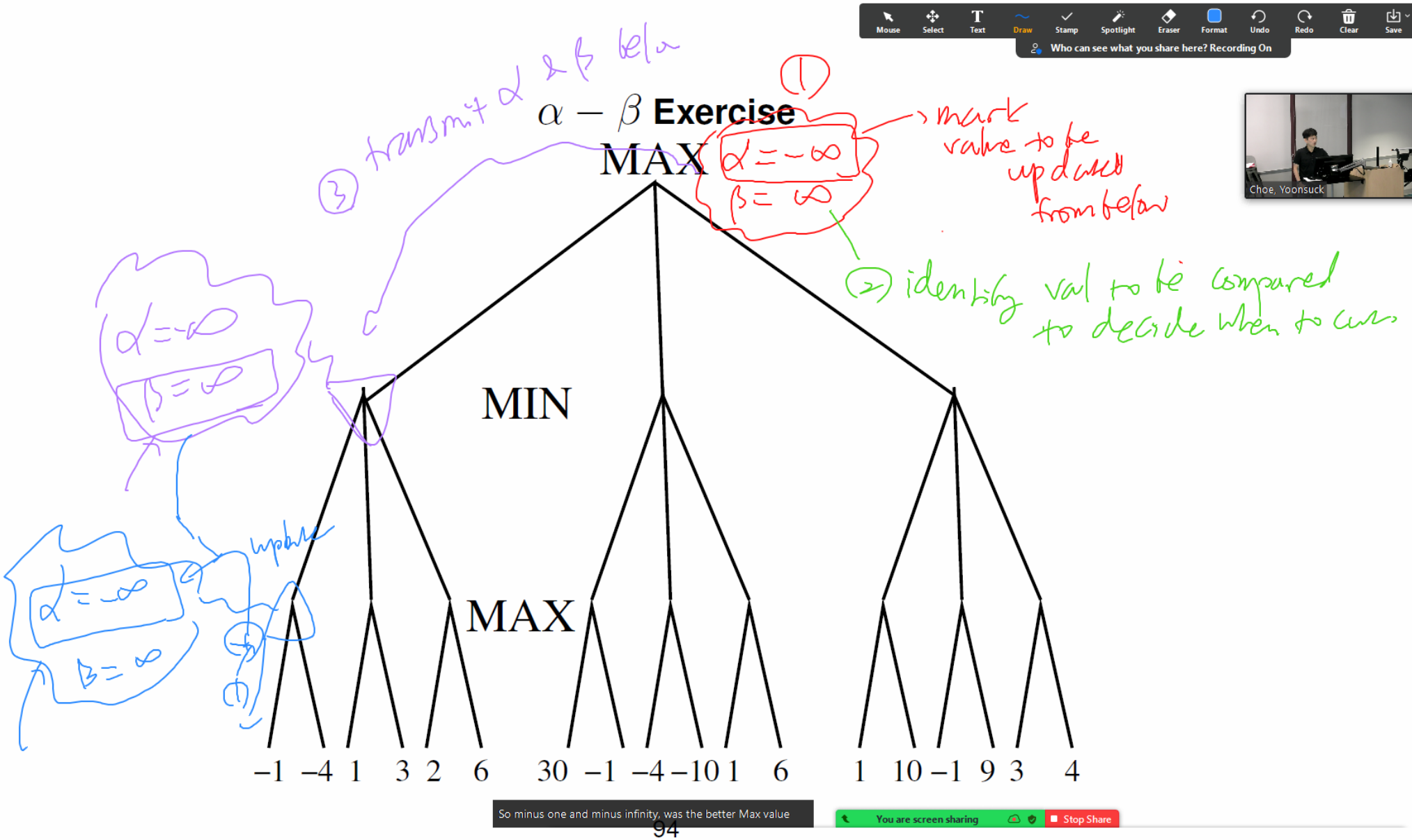
else

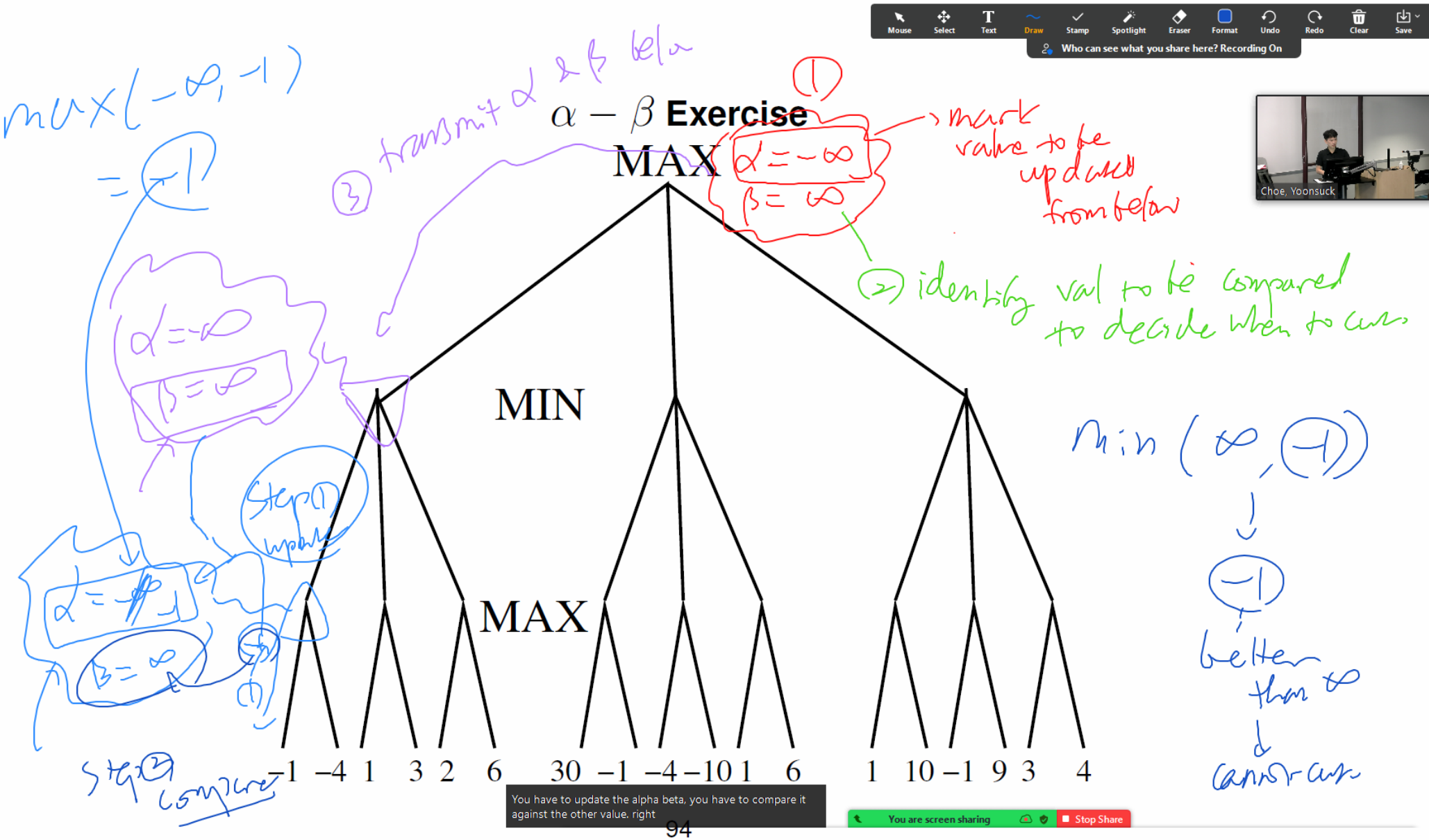
→ **return** min of successors' Minimax-Value



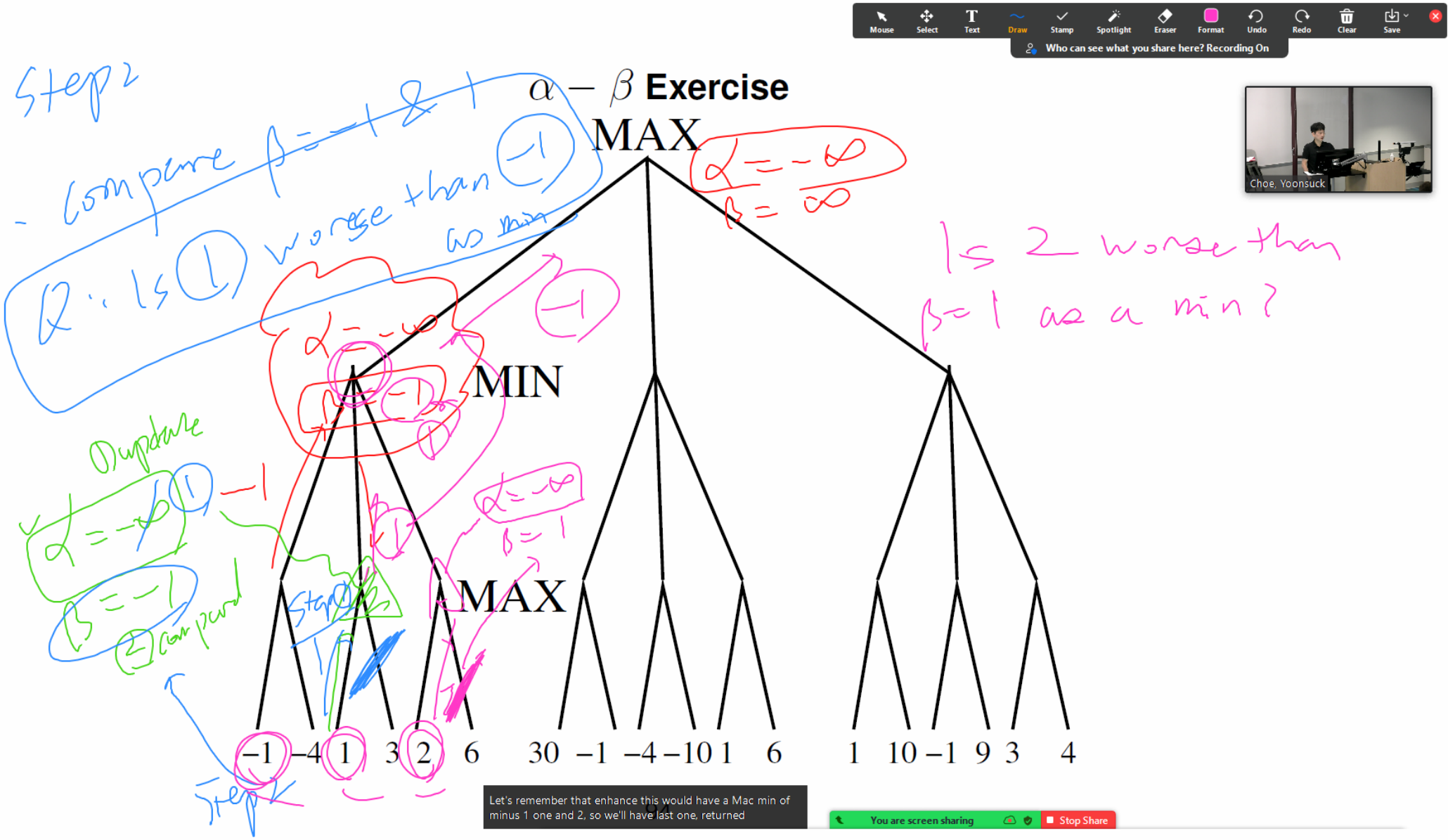
Self, then you'll identify the roots and then find all the children


α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
α = −∞
β = ∞
mark value to be updated from below
identify val to be compared to decide when to cut
transmit α & β below
Choe, Yoonsuck
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Then, because your min node you want to mark this as the value to be updated, don't request the calls
You are screen sharing
Stop Share

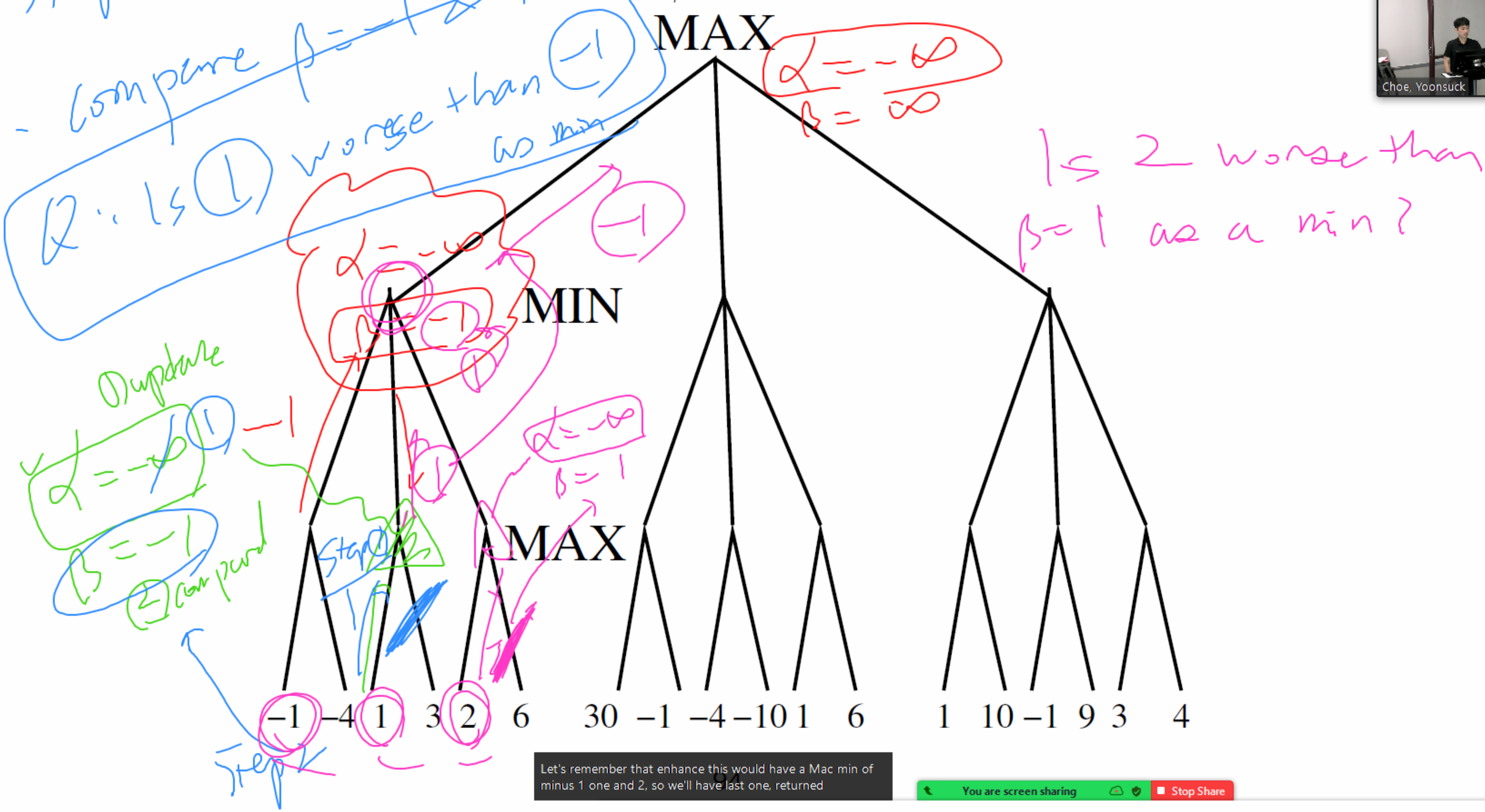

α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
α = −∞
β = ∞
mark value to be updated from below
identify val to be compared to decide when to cut
transmit α & β below
update
So minus one and minus infinity, was the better Max value
You are screen sharing
Stop Share
Choe, Yoonsuck


α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Who can see what you share here? Recording On
Choe, Yoonsuck
You have to update the alpha beta, you have to compare it against the other value. right
You are screen sharing
Stop Share

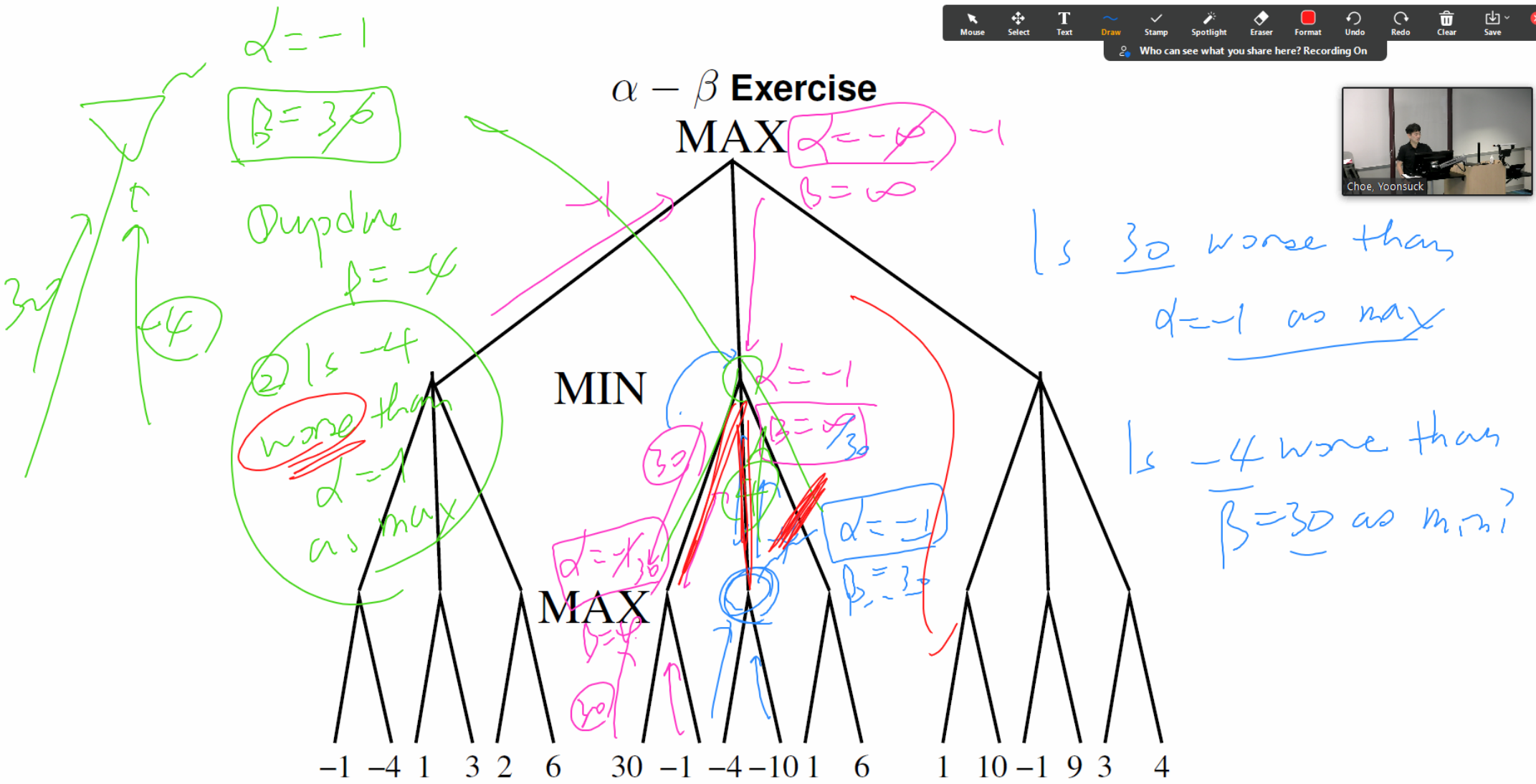

α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Who can see what you share here? Recording On
Choe, Yoonsuck
I say.
You are screen sharing
Stop Share

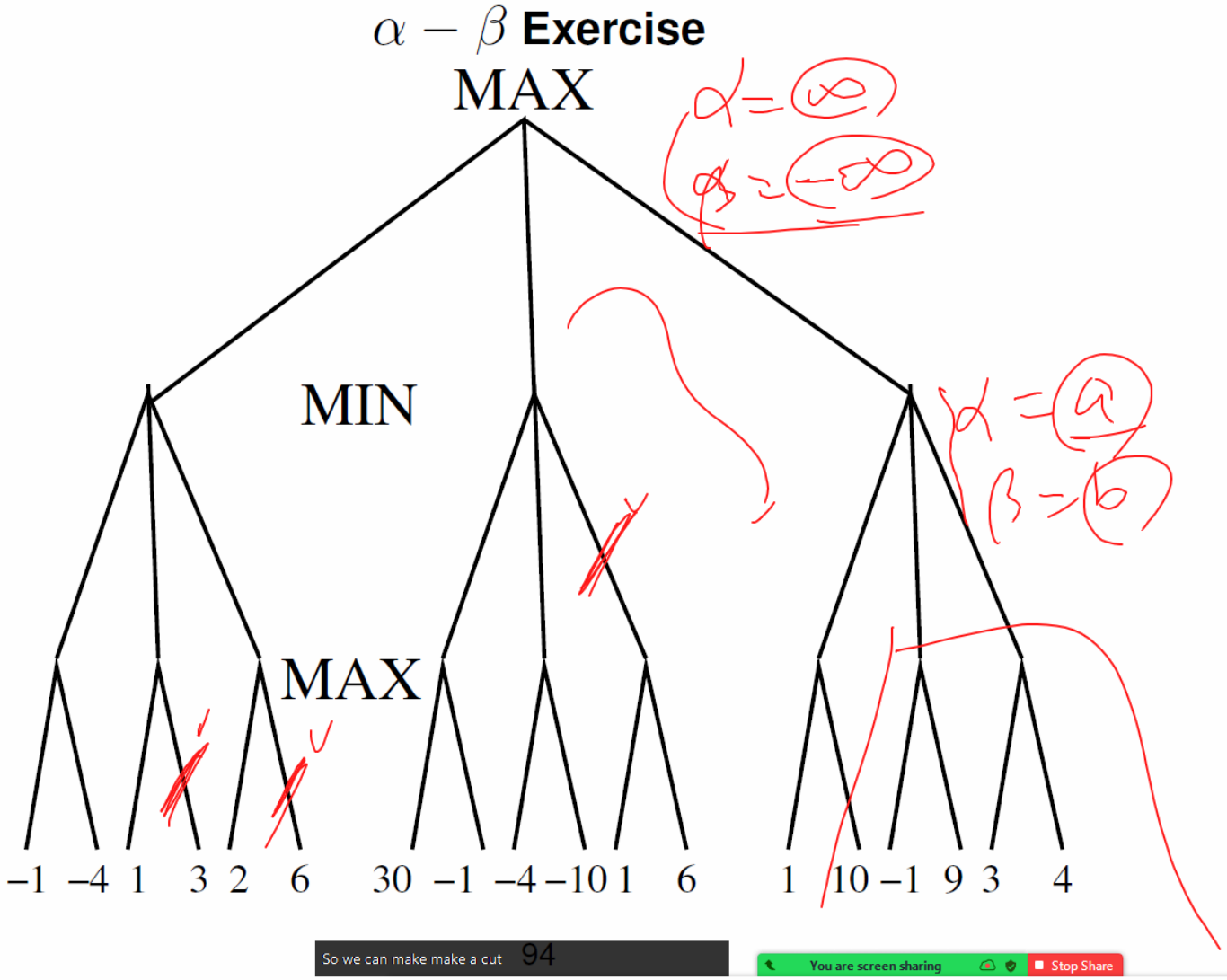

Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Who can see what you share here? Recording On
Choe, Yoonsuck
α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
Step 2
- Compare β=−1 & 1
Q: Is 1 worse than −1 as min?
Is 2 worse than β=1 as a min?
Step 1
Step 2
Let's remember that enhance this would have a Mac min of minus 1 one and 2, so we'll have last one, returned
You are screen sharing
Stop Share

$\alpha - \beta$ Exercise

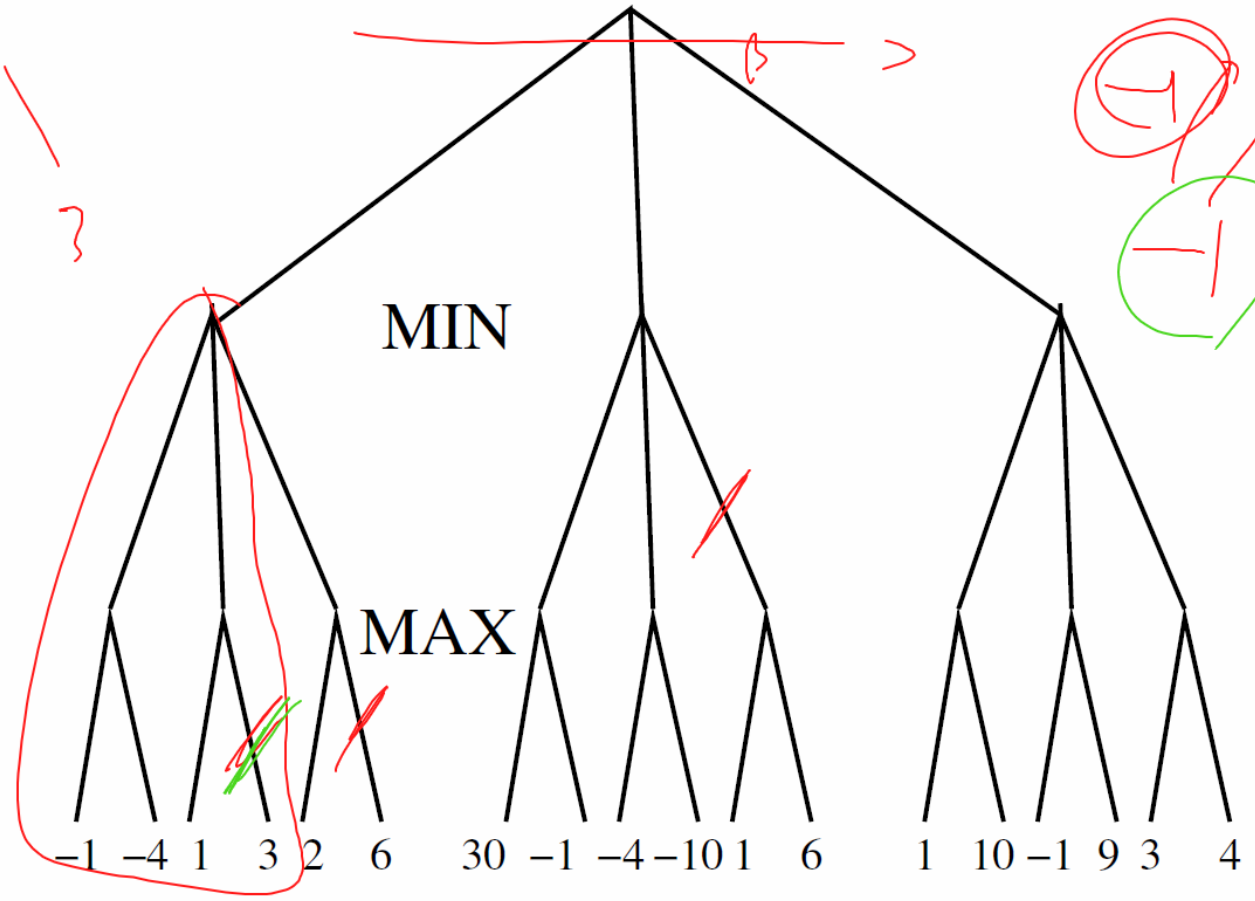


Let's remember that enhance this would have a Mac min of minus 1 one and 2, so we'll have last one, returned

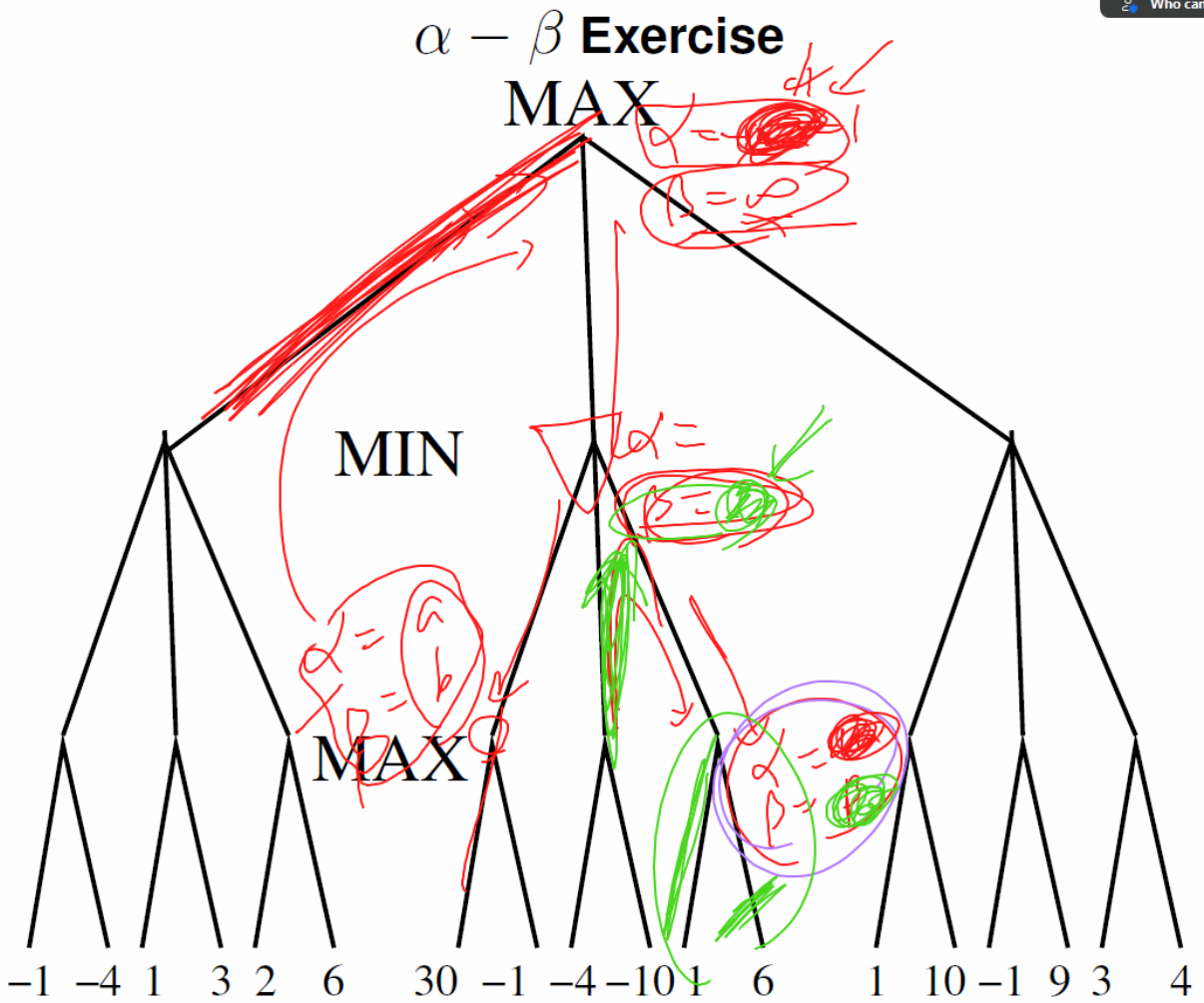

α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
Is 30 worse than α=−1 as max
Is −4 worse than β=30 as min?
Choe, Yoonsuck
Who can see what you share here? Recording On
So actually, the rest is just
94
You are screen sharing
Stop Share

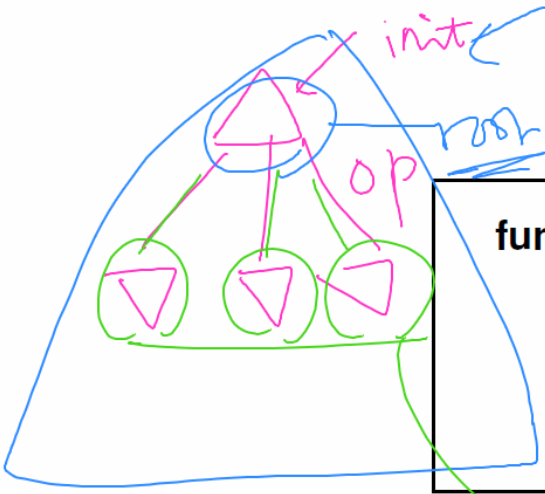

α − β Exercise
MAX
MIN
MAX
−1 −4 1 3 2 6 30 −1 −4 −10 1 6 1 10 −1 9 3 4
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Choe, Yoonsuck
So we can make make a cut
94
You are screen sharing
Stop Share

$\alpha - \beta$ Exercise



So now this is that So you have. The alpha equals minus infinity.





Minimax Decision

function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)

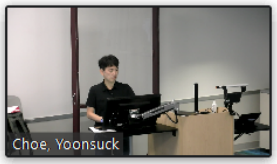
else if Max's move **then**

→ **return** max of successors' Minimax-Value

else

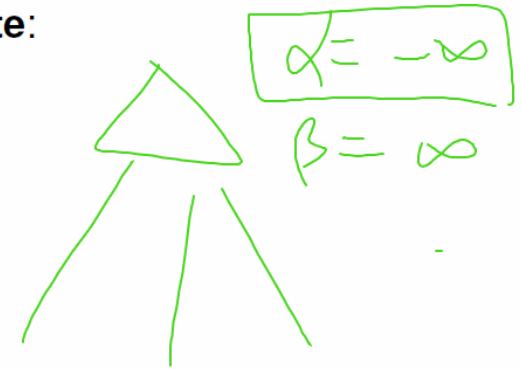
→ **return** min of successors' Minimax-Value

$\alpha - \beta$ Pruning: Initialization



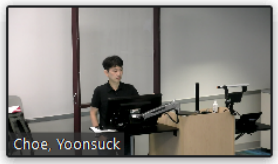
Along the path from the beginning to the current **state**:

- α: best MAX value
  - initialize to $-\infty$
- β: best MIN value
  - initialize to ∞



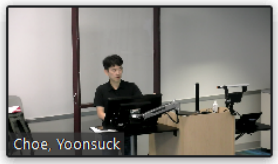
$\alpha - \beta$ Pruning Tips

- At a MAX node:
  - Only α is updated with the MAX of successors.
  - Cut is done by checking if returned $v \geq \beta$.
  - If all fails, MAX(v of succesors) is returned.
- At a MIN node:
  - Only β is updated with the MIN of successors.
  - Cut is done by checking if returned $v \leq \alpha$.
  - If all fails, MIN(v of succesors) is returned.



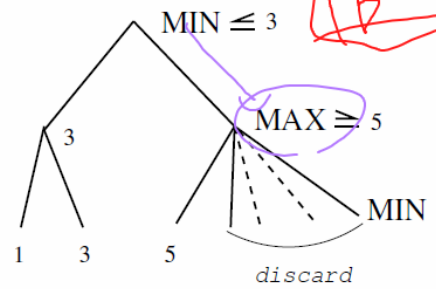
$\alpha - \beta$ Pruning Tips

- At a MAX node:
  - Only α is updated with the MAX of successors.
  - Cut is done by checking if returned $v \geq \beta$.
  - If all fails, MAX(v of succesors) is returned.
- At a MIN node:
  - Only β is updated with the MIN of successors.
  - Cut is done by checking if returned $v \leq \alpha$.
  - If all fails, MIN(v of succesors) is returned.



Min Alpha is the best, Max. So if you just keep this in mind, then understanding the algorithm is a participant

$\alpha - \beta$ Pruning Algorithm: Max-Value



function Max-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility(state)

$v \leftarrow -\infty$

for each *s* in Successor(state) **do**

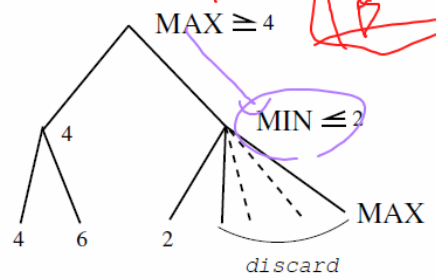
- $v \leftarrow$ Max(v, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* β **CUT!!** */
- $\alpha \leftarrow$Max(α, v)

end

return v

Then this often beta value gets transmitted to you through this requested call

$\alpha - \beta$ Pruning Algorithm: Min-Value



function Min-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility (state)

$v \leftarrow \infty$

for each *s* in Successor(state) **do**

- $v \leftarrow$ Min(v, Max-Value(s,game,α,β))
- **if** $v \leq \alpha$ **then return** v /* α **CUT!!** */
- $\beta \leftarrow$Min(β,v)

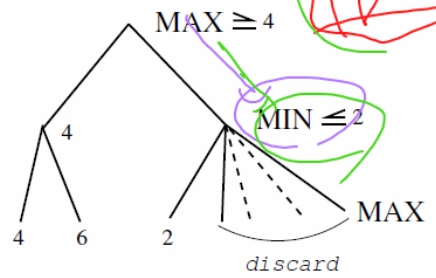
end

return v

So now that we're we have the mean value function

You are screen sharing Stop Share

$\alpha - \beta$ Pruning Algorithm: Min-Value



function Min-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility (state)

$v \leftarrow \infty$

for each *s* in Successor(state) **do**

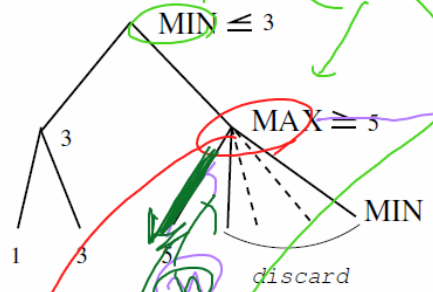
- $v \leftarrow$ Min(v, Max-Value(s,game,α,β))
- **if** $v \leq \alpha$ **then return** v /* α **CUT!!** */
- $\beta \leftarrow$Min(β,v)

end

return v

Sorry on a minnode, and then you still have the same alpha and beta value transmitted from your current or your opponent through this

$\alpha - \beta$ Pruning Algorithm: Max-Value



function Max-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility(state)

$v \leftarrow -\infty$

for each *s* in Successor(state) **do**

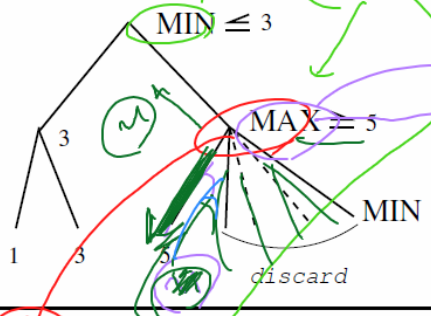
- $v \leftarrow$ Max(v, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* β **CUT!!** */
- $\alpha \leftarrow$Max(α, v)

end

return v

Alright. So that will happen along this direction.

$\alpha - \beta$ Pruning Algorithm: Max-Value



function Max-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility(state)

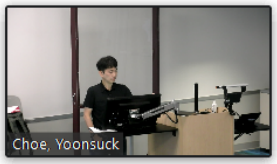
$v \leftarrow -\infty$

for each *s* in Successor(state) **do**

- $v \leftarrow$ Max(v, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* β **CUT!!** */
- $\alpha \leftarrow$Max(α, v)

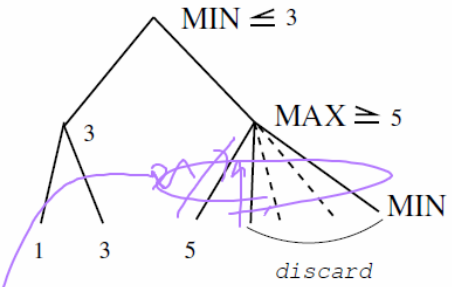
end

return v



So far is rules then cut. Okay, and also the update would happen for the particular value of a beta value that you marked putting to the particular type of your note

$\alpha - \beta$ Pruning Algorithm: Max-Value



function Max-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility(state)

$v \leftarrow -\infty$

for each *s* in Successor(state) **do**

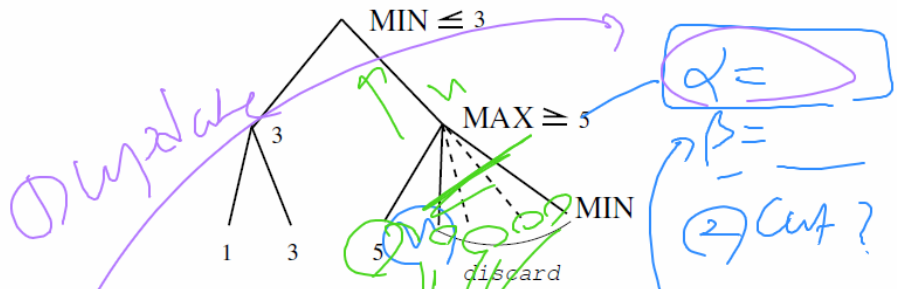
- $v \leftarrow$ Max(v, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* β **CUT!!** */
- $\alpha \leftarrow$Max(α, v)

end

return v

So far as you stole the branches, so these mean value functions will be called on all

$\alpha - \beta$ Pruning Algorithm: Max-Value



function Max-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility(state)

$v \leftarrow -\infty$

for each *s* in Successor(state) **do**

- $v \leftarrow$ Max(v, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* β **CUT!!** */
- $\alpha \leftarrow$Max(α, v)

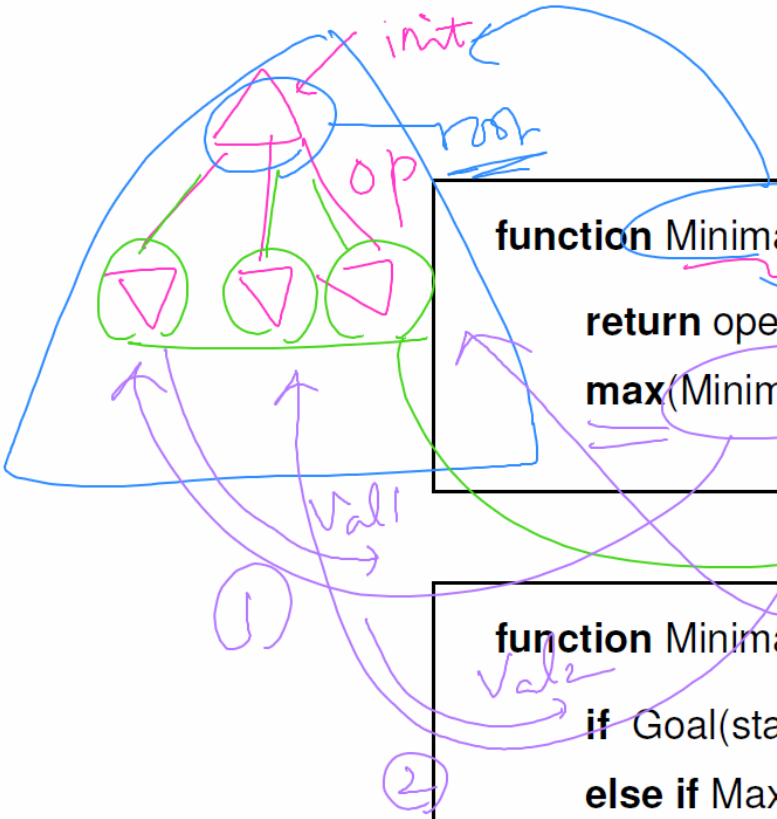
end

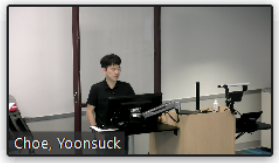
return v

So if you cut here, then the V value would be 5

You are screen sharing Stop Share

Minimax Decision





function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)

else if Max's move **then**

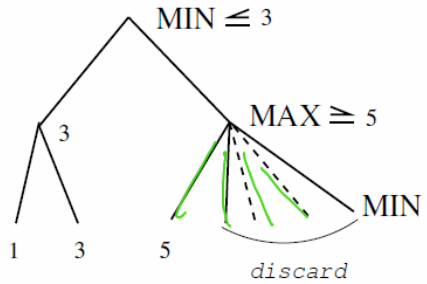
→ **return** max of successors' Minimax-Value

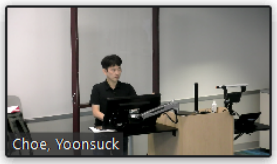
else

→ **return** min of successors' Minimax-Value

I'm sorry, child, which are written you under the value then now, you just return the Max with the operator that gives gave you the Max

$\alpha - \beta$ Pruning Algorithm: Max-Value





function Max-Value (state, game, α, β) **return** utility value

α: best ~~MAX on path~~ to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility(state)

$v \leftarrow -\infty$

for each *s* in Successor(state) **do**

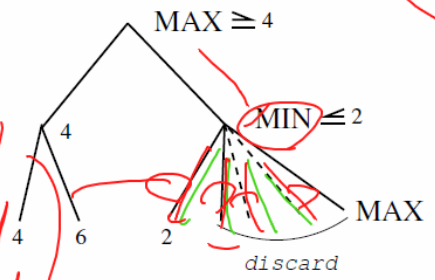
- $v \leftarrow$ Max(v, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* β **CUT!!** */
- $\alpha \leftarrow$Max(α, v)

end

return v

On his children.

$\alpha - \beta$ Pruning Algorithm: Min-Value



function Min-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility (state)

$v \leftarrow \infty$

for each *s* in Successor(state) **do**

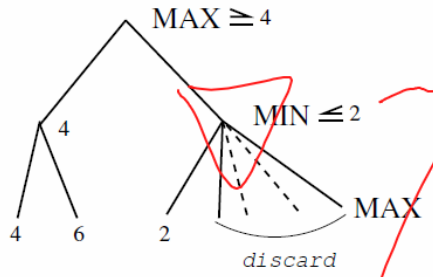
- $v \leftarrow$ Min(v, Max-Value(s,game,α,β))
- **if** $v \leq \alpha$ **then return** v /* α **CUT!!** */
- $\beta \leftarrow$Min(β,v)

end

return v

So all of these would be these

$\alpha - \beta$ Pruning Algorithm: Min-Value



function Min-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility (state)

$v \leftarrow \infty$

for each *s* in Successor(state) **do**

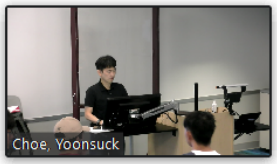
- $v \leftarrow$ Min(v, Max-Value(s,game,α,β))
- **if** $v \leq \alpha$ **then return** v /* α **CUT!!** */
- $\beta \leftarrow$Min(β,v)

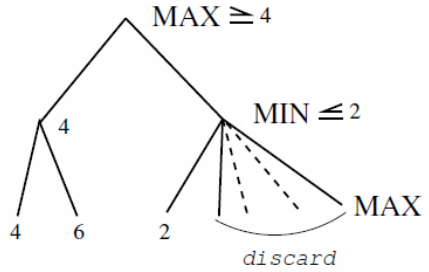
end

return v

Sorry I didn't have a chance to talk about the homework, so i'll probably send out to beef email with some tips, hey, zeal, and I again forgot to calculate this

$\alpha - \beta$ Pruning Algorithm: Min-Value





function Min-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility (state)

$v \leftarrow \infty$

for each *s* in Successor(state) **do**

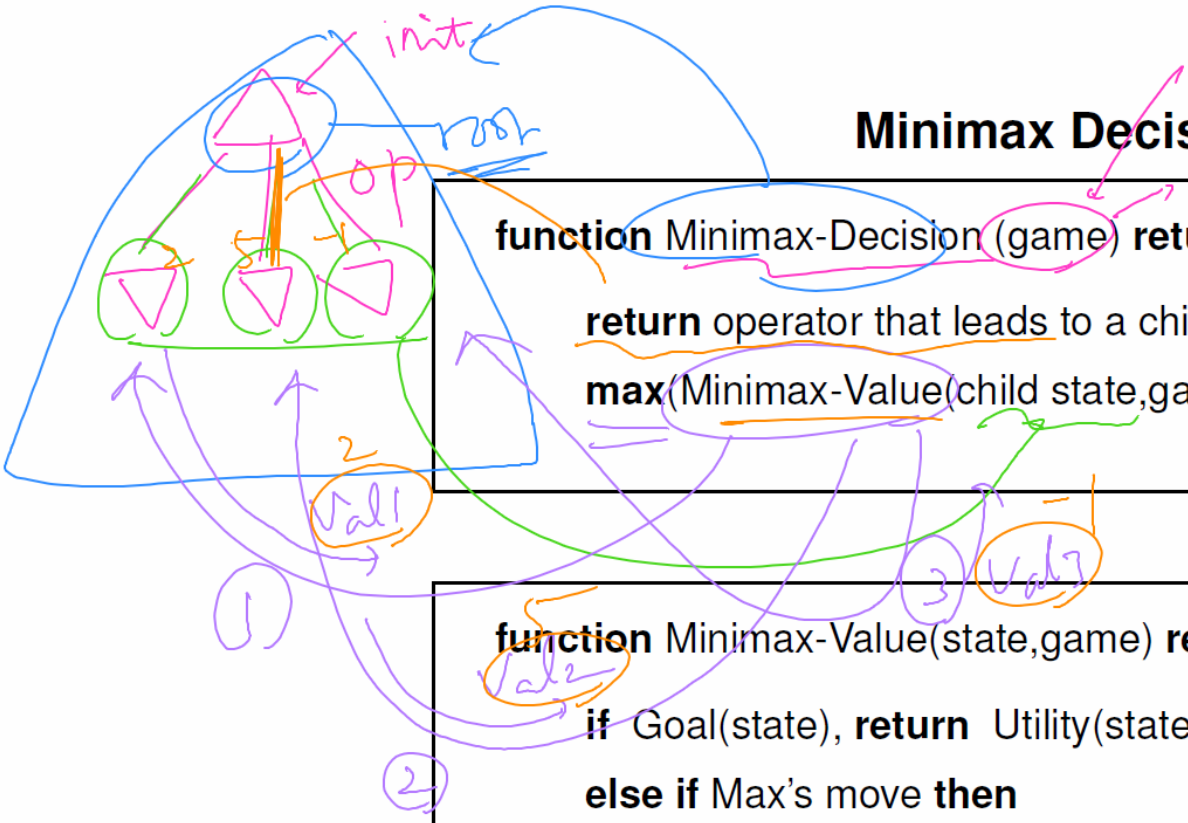
- $v \leftarrow$ Min(v, Max-Value(s,game,α,β))
- **if** $v \leq \alpha$ **then return** v /* α **CUT!!** */
- $\beta \leftarrow$Min(β,v)

end

return v

Sorry I didn't have a chance to talk about the homework, so i'll probably send out to beef email with some tips, hey, zeal, and I again forgot to calculate this

Minimax Decision



function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)

else if Max's move **then**

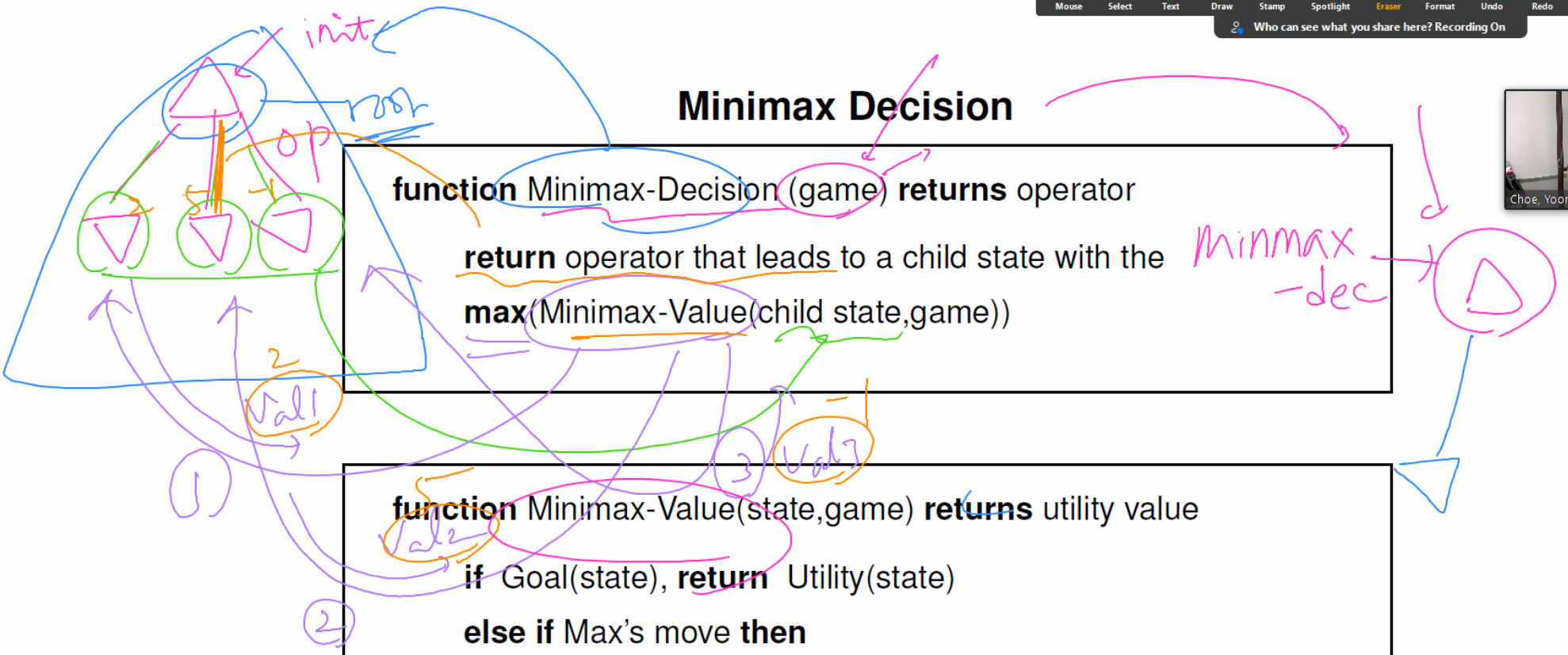
→ **return** max of successors' Minimax-Value

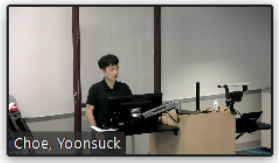
else

→ **return** min of successors' Minimax-Value

Then, what this return is, The operator at least

Minimax Decision





function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)

else if Max's move **then**

→ **return** max of successors' Minimax-Value

else

→ **return** min of successors' Minimax-Value

Children

Minimax Decision

function Minimax-Decision (game) **returns** operator

return operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

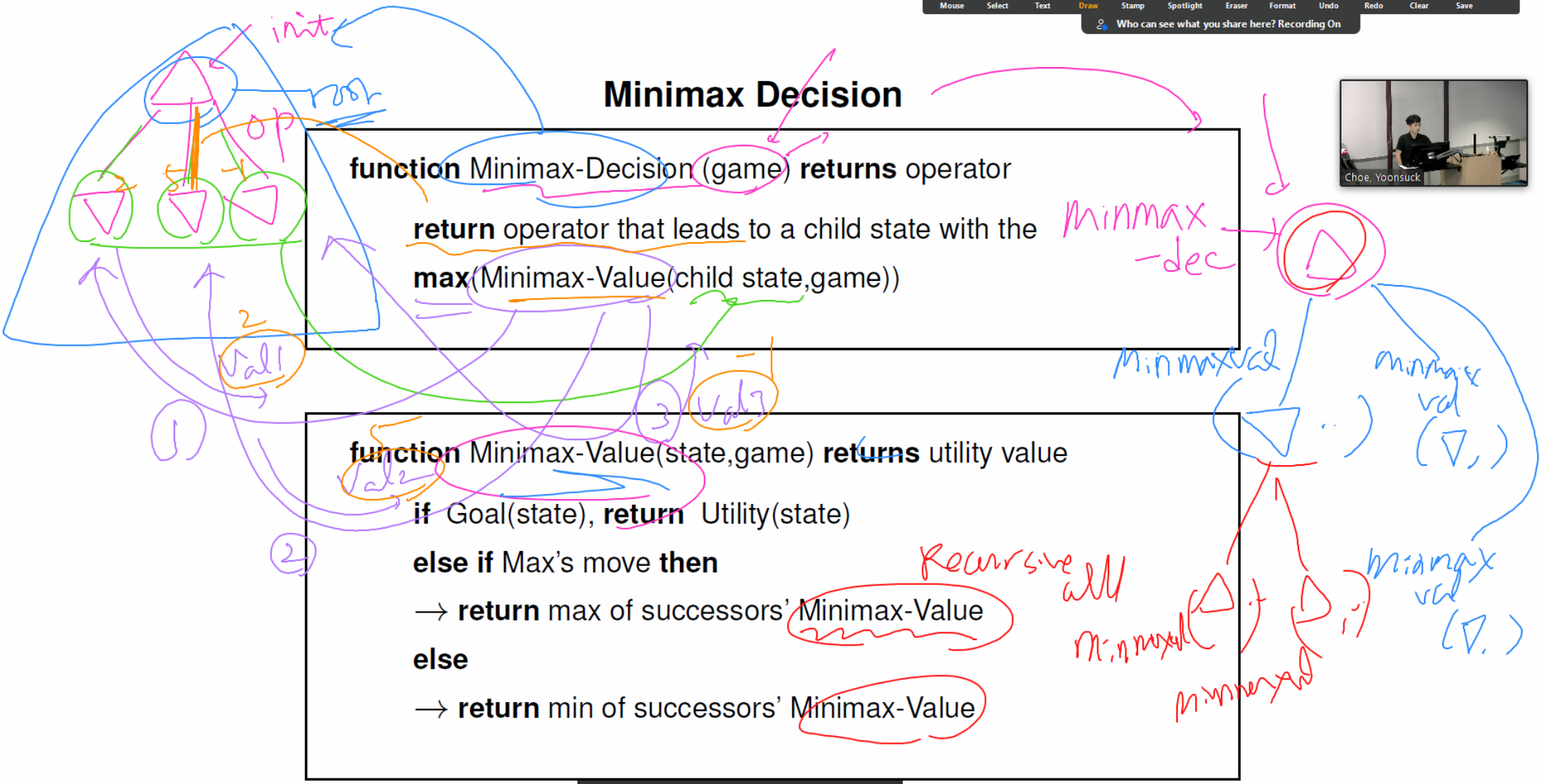
if Goal(state), **return** Utility(state)

else if Max's move **then**

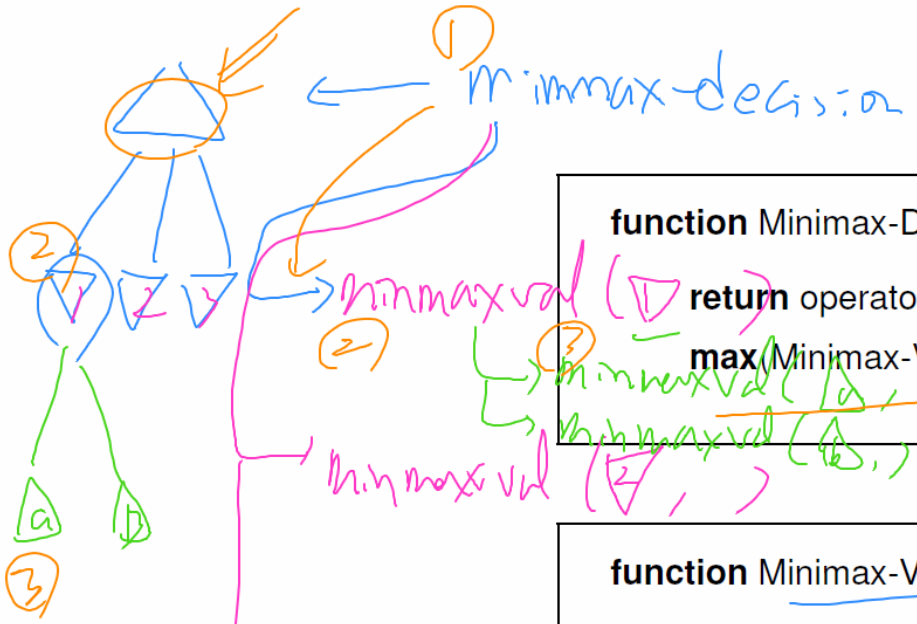
→ **return** max of successors' Minimax-Value

else

→ **return** min of successors' Minimax-Value



Then another min, next value on the second child, and so on.



Minimax Decision

function Minimax-Decision (game) **returns** operator

→ **return** operator that leads to a child state with the **max**(Minimax-Value(child state,game))

function Minimax-Value(state,game) **returns** utility value

if Goal(state), **return** Utility(state)
else if Max's move **then**
→ **return** max of successors' Minimax-Value
else
→ **return** min of successors' Minimax-Value