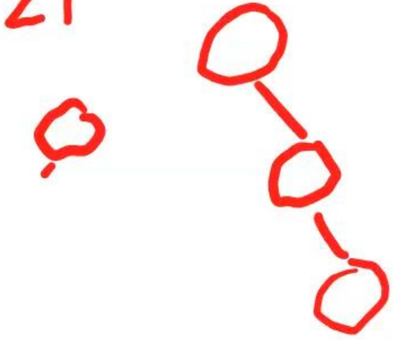
randomly created

1 2 3

3 2 1

Depth of a Binary Search Tree



- Imagine that each internal node has exactly two children.
- A depth of $\log_2 n$ is the best possible.
- So the depth can be between $\log_2 n$ and n.
- What is the average depth?

08:09 Request control

Solving the Recurrence Relation

- Summing these equation, we should get

$$\boxed{D(n) / (n+1) = D(1)/2 + 2c \sum_{j=2}^{n} 1/j}$$

where $c=(j-1)/(j+1)$ is close to 1

- Now, notice that the summation on the right is about $H(n) = O(\log n)$.

H(n) – ln(n) approaches a constant – Euler Mascheroni constant

$$\int_1^n \frac{1}{x}dx = ln(n)$$

- Therefore, $D(n) = O(n\log n)$.

Solving the Recurrence Relation

- Summing these equation, we should get

$$\boxed{D(n) / (n+1) = D(1)/2 + 2c \sum_{j=2}^{n} 1/j}$$

where $c=(j-1)/(j+1)$ is close to 1

27:31 Request control

AS

Solving the Recurrence Relation

- Consider

$$\boxed{nD(n) = (n+1)\, D(n-1) + 2(n-1).}$$

- Try another telescoping but after some adjustment.

  Divide the whole equation by n(n+1) to get:

$$D(n) / (n+1) = D(n-1) / n + 2(n-1) / (n(n+1))$$

- Now, write the above equation for n, n-1, …, all the way to n = 2.

$$D(n-1) / n = D(n-2) / (n-1) + 2(n-2) / ((n-1)n)$$

$$D(n-2) / (n-1) = D(n-3) / (n-2) + 2(n-3) / ((n-2)(n-1))$$

$$\vdots$$

$$D(2) / 3 = D(1) / 2 + 2 / 2*3$$

Solving the Recurrence Relation

- Consider $D(n) = (2/n) \left(\sum_{j=0}^{n-1} D(j)\right) + n - 1$.
- Rearrange as

  $n\,D(n) = 2 \left(\sum_{j=0}^{n-1} D(j)\right) + n(n-1)$

- Now, write the equation with n-1 replacing n.

  $(n-1)\,D(n-1) = 2 \left(\sum_{j=0}^{n-2} D(j)\right) + (n-1)(n-2)$

- Subtract the two equations to get:

  $nD(n) - (n-1)\,D(n-1) = 2\,D(n-1) + 2(n-1)$

- Rearrange as:

  $nD(n) = (n+1)\,D(n-1) + 2(n-1)$

25:16 Request control

Solving the Recurrence Relation

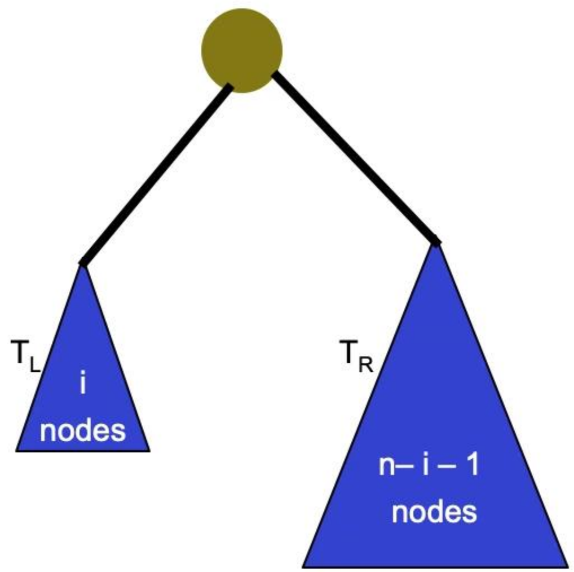
- The recurrence relation simplifies to

$$\boxed{D(n) = (2/n)\left(\sum_{j=0}^{n-1} D(j)\right) + n - 1}$$

- Can be solved using known techniques as follows.

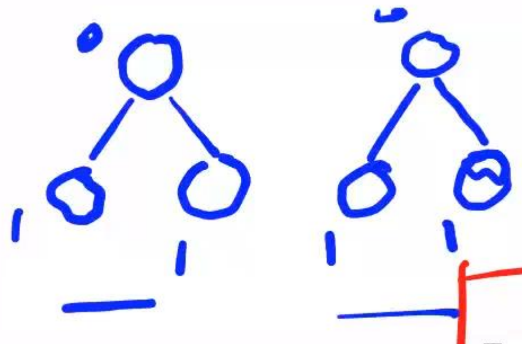
24:02 Request control

Average Depth

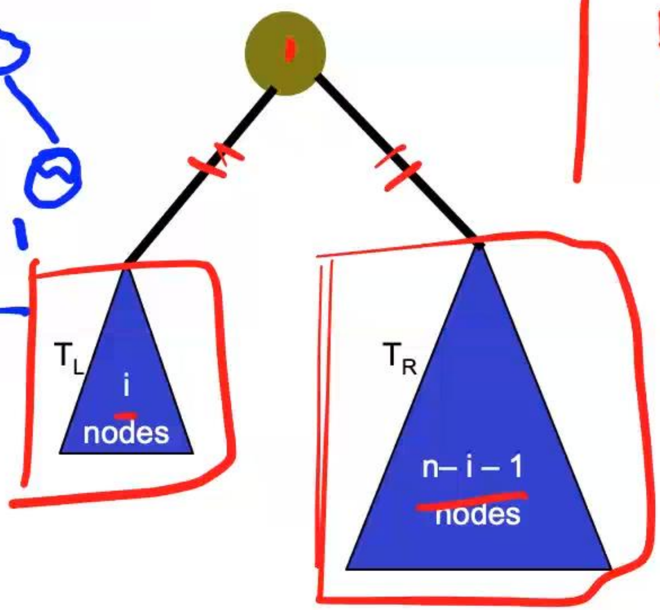


- So, $\boxed{D(N) = D(i) + D(n-i-1) + n-1}$

Average Depth

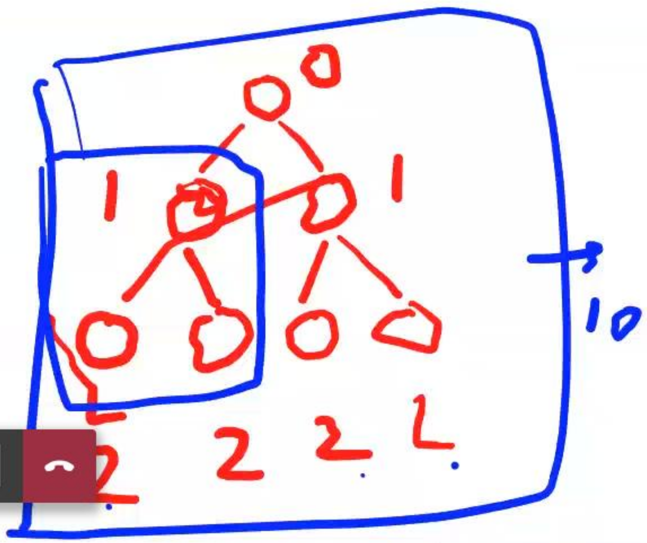


$2 + 2 + 6$
$= 10$



$D(T_R)$

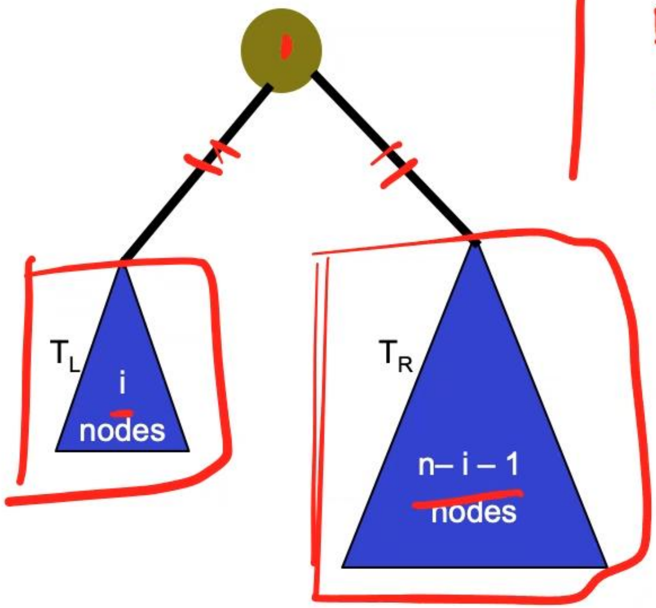
$D(T) = D(T_L) + D(T_R) + n - 1.$



10

- Further, if now these trees are attached to the root
  - the depth of each node in T_L and T_R increases by 1.

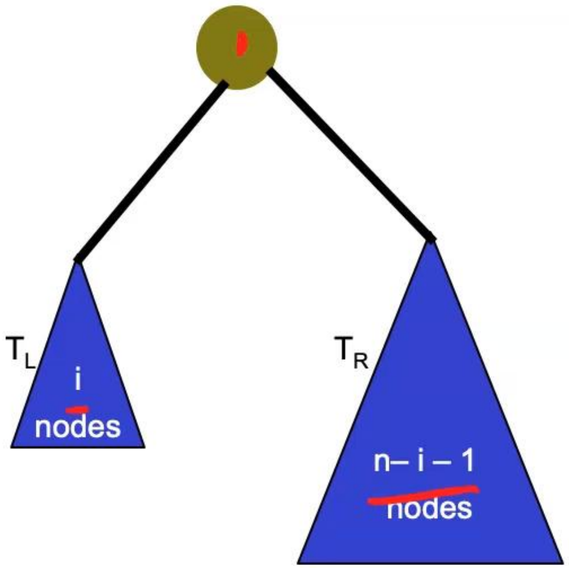
Average Depth



$D(T) = D(T_L) + D(T_R) + n-1.$

- Further, if now these trees are attached to the root
  - the depth of each node in T_L and T_R increases by 1.

Average Depth



- Further, if now these trees are attached to the root
  - the depth of each node in T_L and T_R increases by 1.

Average Depth

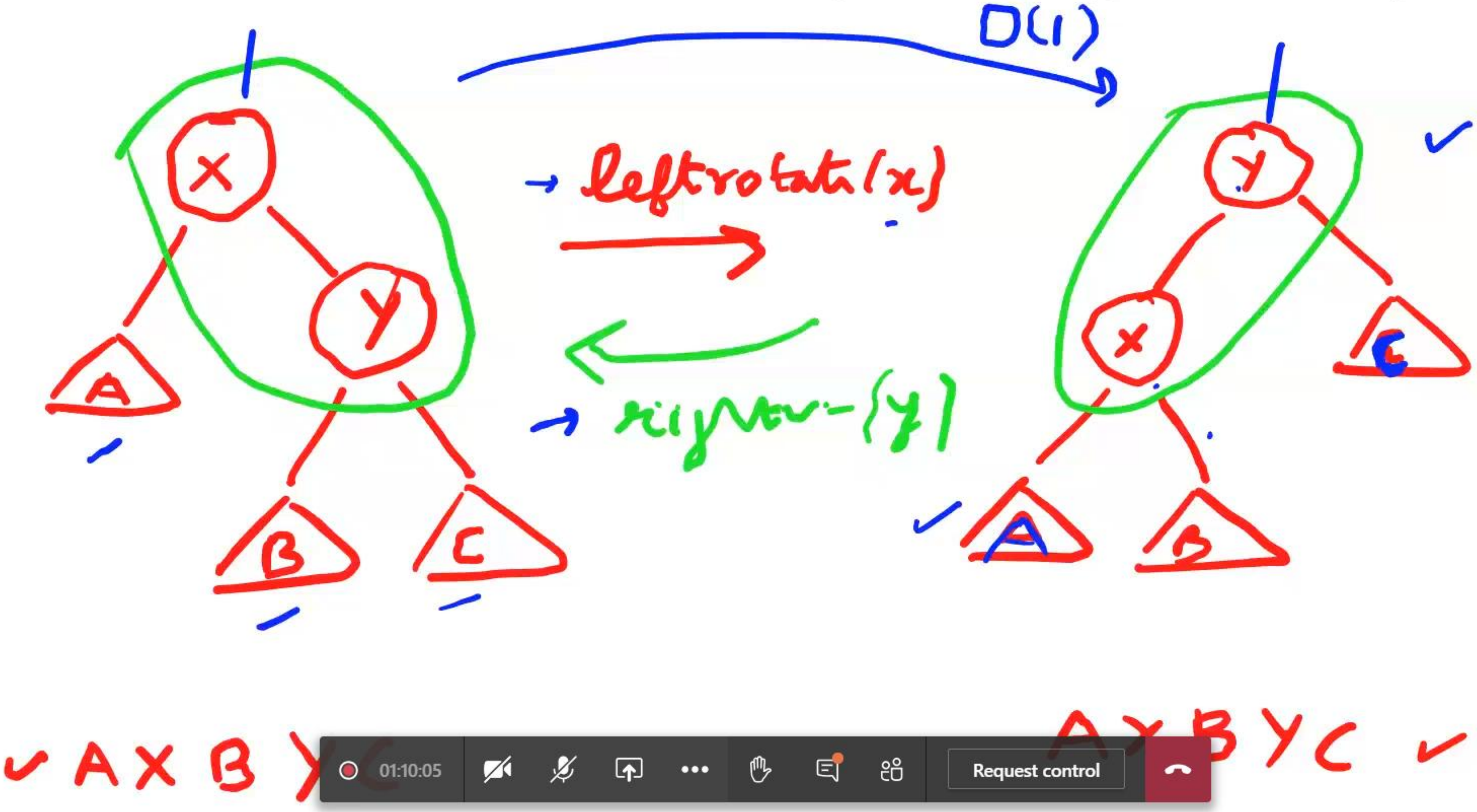
- Internal path length : The sum of the depths of all nodes in a tree.
- Let D(n) to be the internal path length of some binary search tree of n nodes.
  - $\boxed{D(n) = \sum_{i=1}^{n} d(i)}$ where d(i) is the depth of node i.
- Note that D(1) = 0.

Average Depth

- A good notion as most operations take time proportional on the depth of the binary search tree.
- Still, not a satisfactory measure as we wanted worst-case performance bounds.

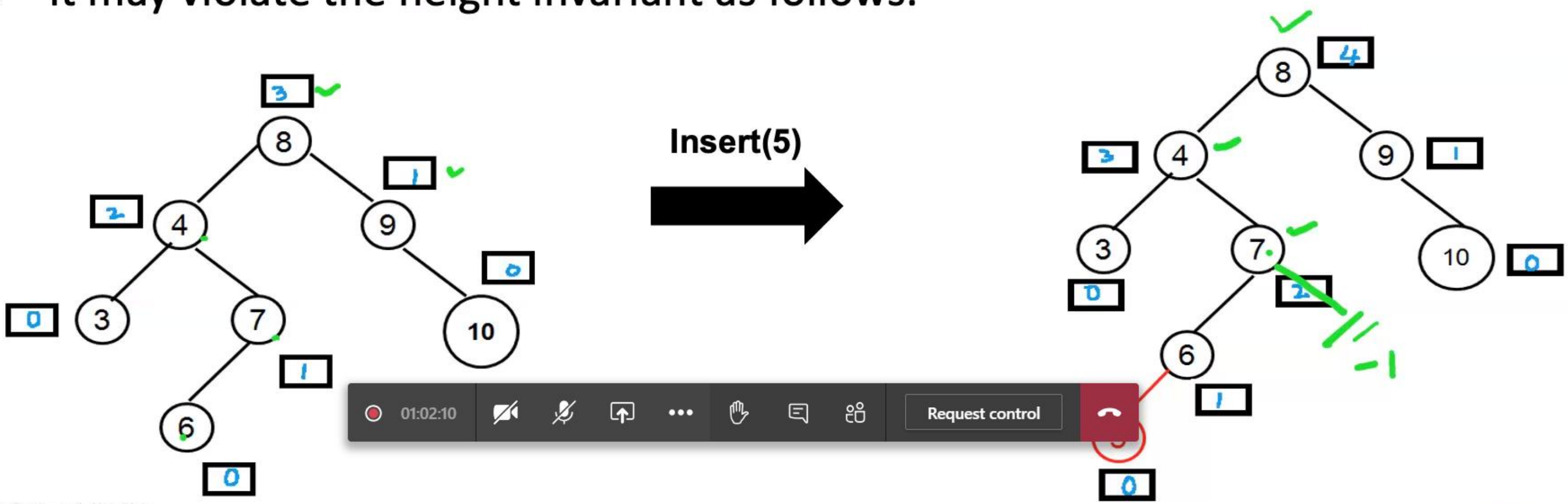
12:47 Request control

The idea of left heavy and right heavy



Insert in an AVL Tree

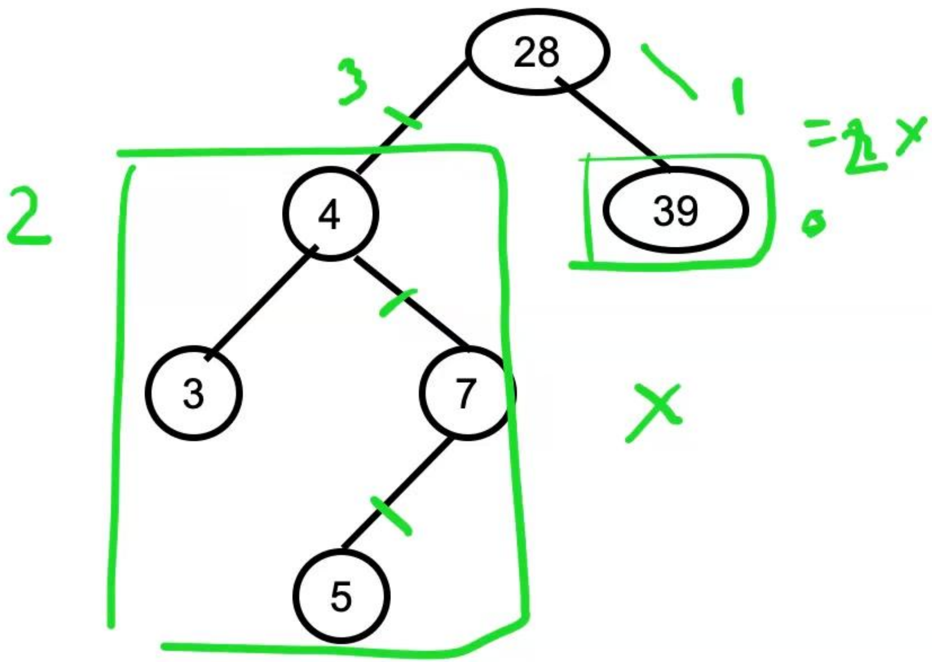
- Proceed as insertion into a search tree.
  - At least satisfies the search invariant.
- It may violate the height invariant as follows.



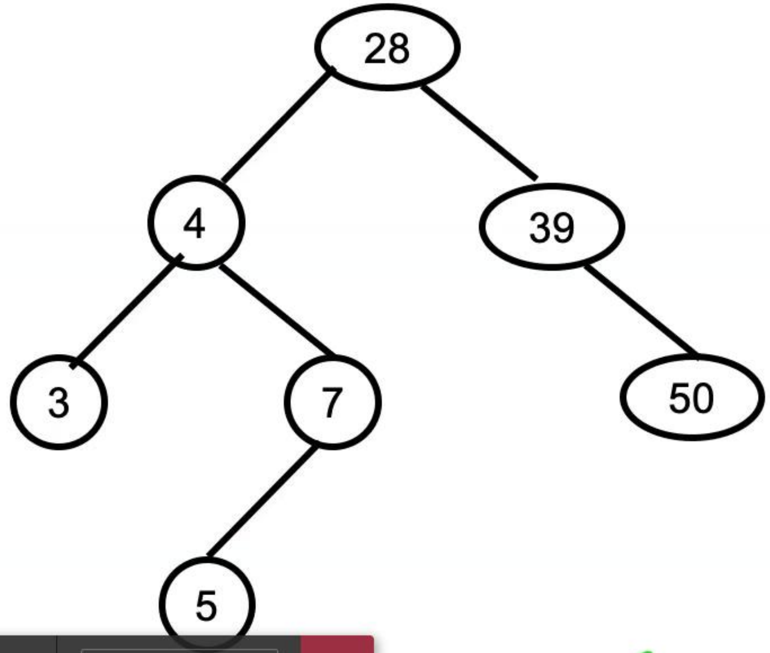
The AVL Tree

- A binary tree satisfying the
  - search invariant, and
  - the height invariant
- is called an AVL tree.
- Named after its inventors, Adelson, Velski and Landis.
- Throughout, let us define the height of an empty tree to be -1.

Towards Height Balanced Trees



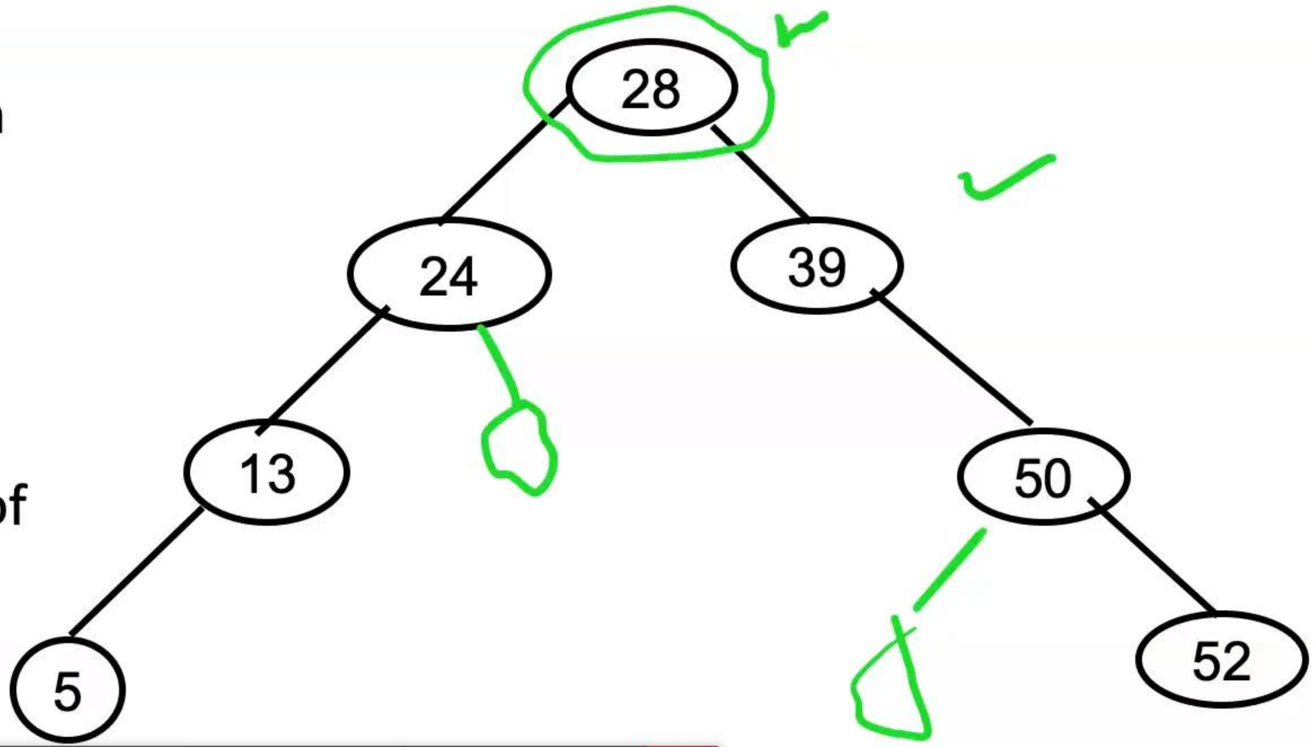
Not a Height Balanced Tree



Height Balanced Tree

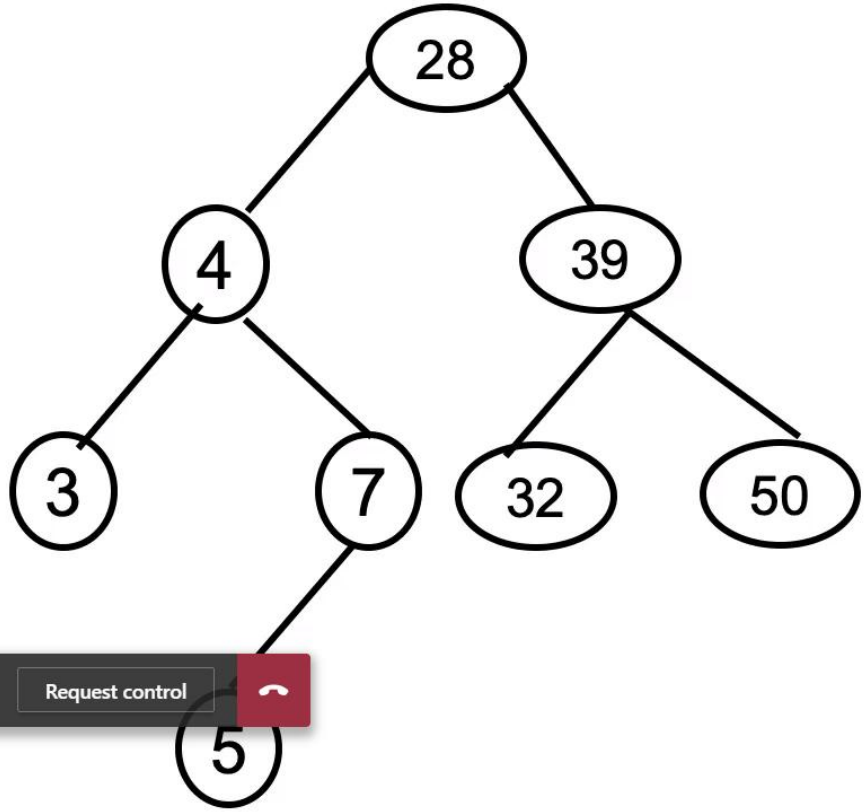
Towards Height Balanced Trees

- Attempt 1: Would it suffice if we say that the root has both a left and a right subtree of equal height?
- Still, the depth of the tree is not O(log n).
- In the this tree, irrespective of values at the nodes, the root has left and right subtrees of equal height.



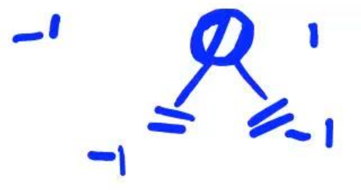
Towards Height Balanced Trees

- How can we control the height of a binary search tree?
  - should still maintain the search invariant
  - additional invariants required.

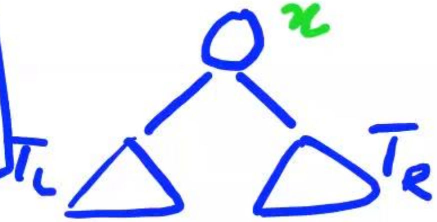


54:50
Request control

Towards Height Balanced Trees

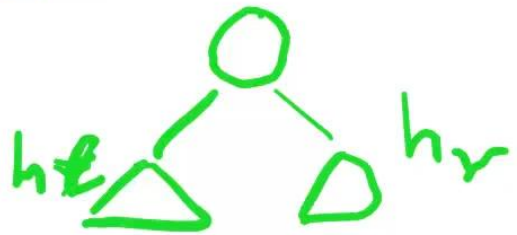


$\boxed{\max\{\text{height (left child)}, h(\text{right})\} + 1}$



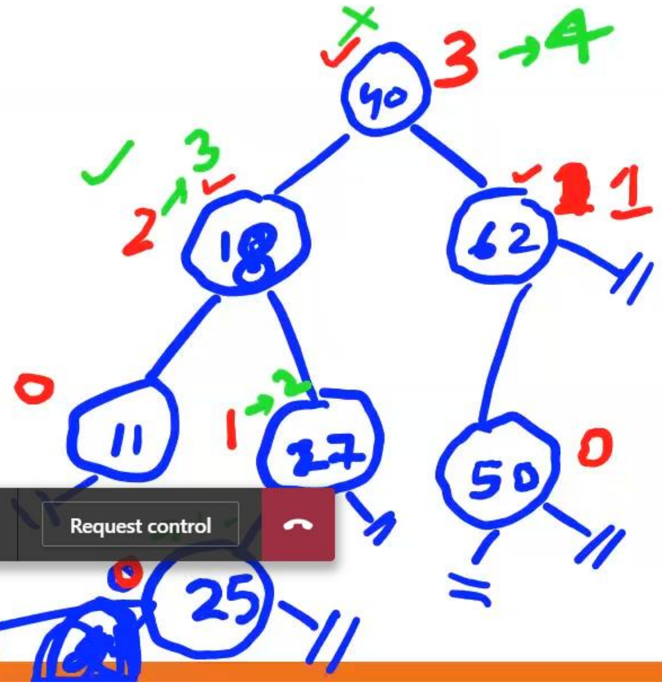
$\boxed{\text{DS avg}}$

AVL Tree –



$\boxed{|h_l - h_r| \leq 1}$

(for every node)



Average Runtime

- Now, remove() operation may introduce a skew.
- Replacement node can skew left or right subtree.
- Can pick the replacement node from the left or the right subtree uniformly at random.
  - Still not known to help.
- So, at best we can be satisfied with an average O(log n) runtime in most cases.
- Need techniques to restrict the height of the binary search tree.

Average Runtime

- Now, remove() operation may introduce a skew.
- Replacement node can skew left or right subtree.
- Can pick the replacement node from the left or the right subtree uniformly at random.
  - Still not known to help.
- So, at best we can be satisfied with an average O(log n) runtime in most cases.

37:02 Request control

AS

Solving the Recurrence Relation

- If all subtree sizes are equally likely then D(i) is the average over all subtree sizes.
  - That is, i ranges over 0 to n – 1.
  - Can hence see that $\boxed{D(i) = (1/n) \sum_{j=0}^{n-1} D(j)}$
- Similar is the case with the right subtree.
  - So, $\boxed{D(n - i - 1) = (1/n) \sum_{j=0}^{n-1} D(j)}$

32:56 Request control

Solving the Recurrence Relation

- The solution to D(n) is D(n) = O(n log n).
- How is D(n) related to the average depth of a binary search tree.
  - There are N paths in any binary search tree from the root.
  - So the average internal path length is O(log n).
- Does this mean that each operation has an average O(log n) runtime.
  - Not quite.