Balanced Binary Search Tree

AVL Tree

Review

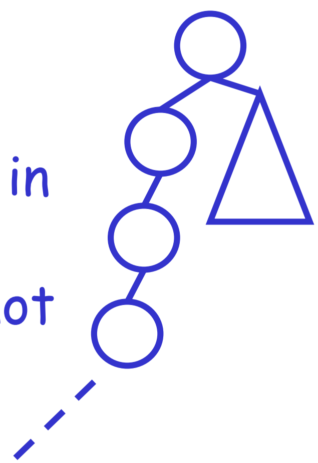
A binary search tree can perform the following operations, Insert, Delete, Search, Minimum, Maximum, Predecessor, Successor, in O(h) time where h is the height of the tree

What is h?
- Worst Case: O(n)
- Best Case: O(log n)

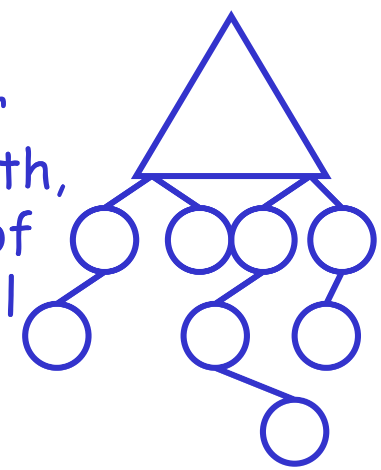
Aim: we want to achieve O(log n) worst case time complexity for all operations

Observation:

1 If there are a few long paths in the tree, probably it is not good.



2 If the paths in the tree have "more or less" the same length, the overall height of the tree should still be "short".



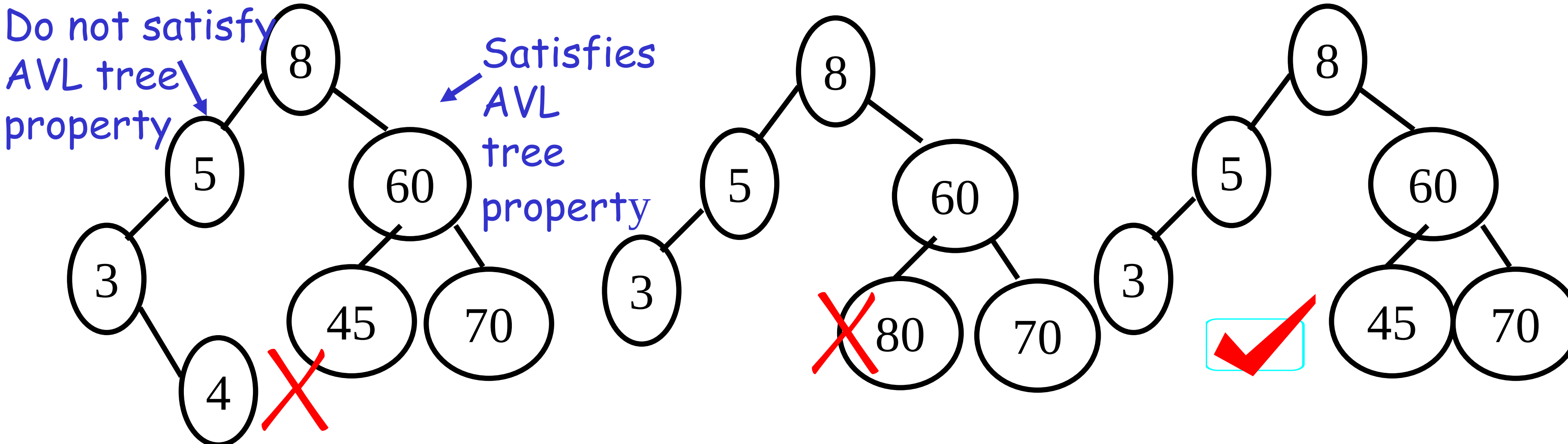
Idea: Construct a tree which is "balanced" (leaves have more or less the same height) and maintain this kind of balance after insertion and deletion.

s.t. h (height of tree) is always bounded by O(log n)

Examples:
- ♠ AVL tree
- ♠ 2-3 tree
- ♠ Red-black tree
- ♠ Splay tree

Two issues:
(1) Make sure that the height of the tree of n nodes is O(log n).
(2) Insertion and deletion of nodes must be done in O(log n) time.

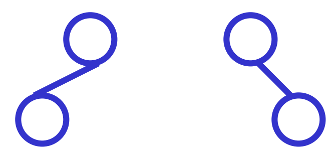
Definition: An AVL tree is a binary search tree such that, for every node, the difference between the heights of its left and right subtrees is at most 1. Note: height of a null tree is defined as -1



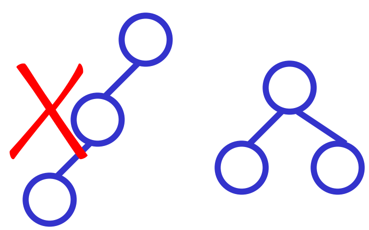
Some more examples:

AVL tree with one node:

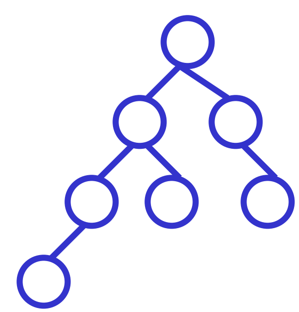
AVL tree with two nodes:



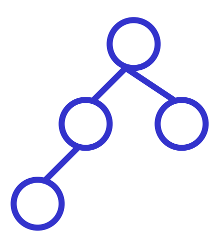
AVL tree with three nodes:



AVL tree with 7 nodes:



AVL tree with four nodes:



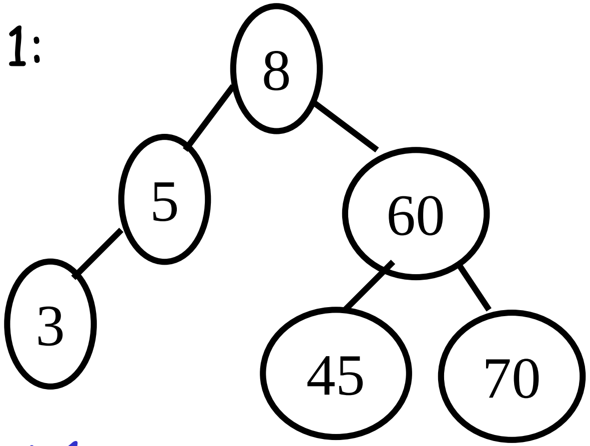
Implication:
With this property, the height of an AVL tree with n nodes is always O(log n)

[The MIT book ex. 13-3 (a)]
Prove that an AVL tree with n nodes has height O(log n).

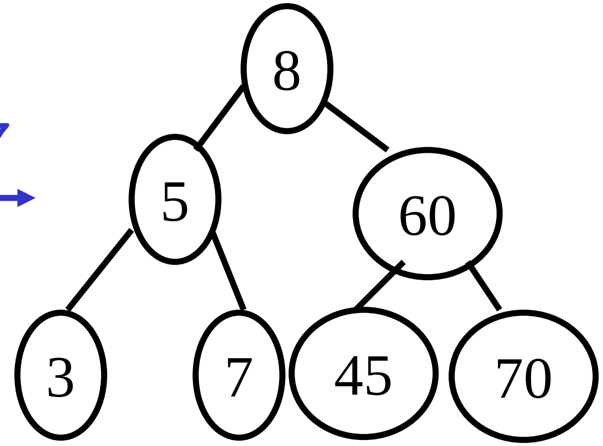
Proof:
We will talk about it later.

Insertion: need to maintain the AVL tree property after each insertion

Example 1:



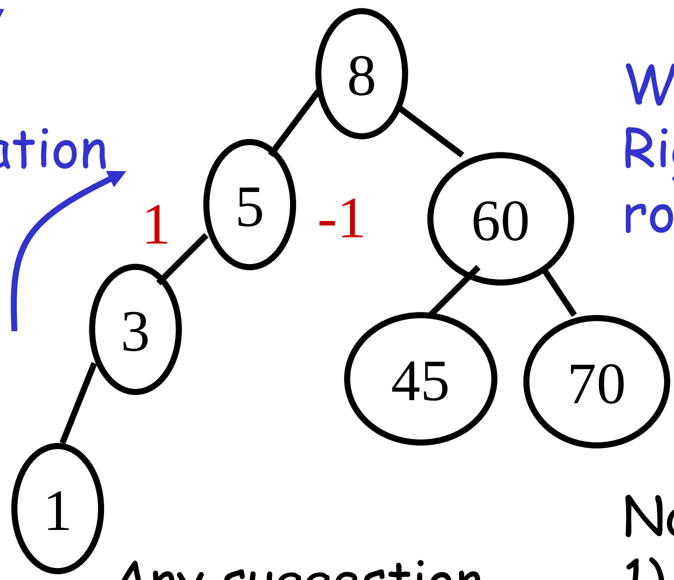
Insert 7



Still an AVL tree, so no further rebalancing needed

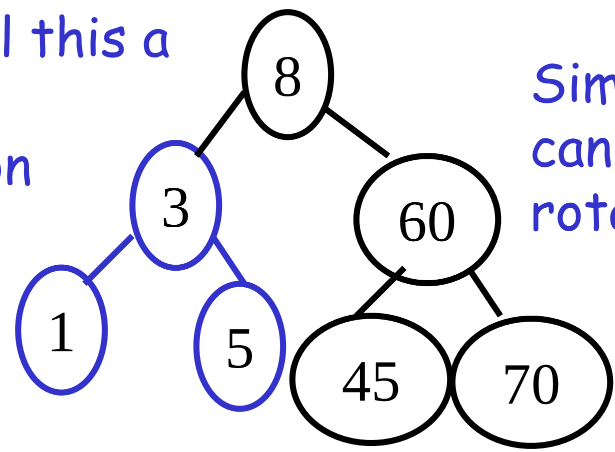
Insert 1

rotation



Any suggestion how we can do it?

We call this a Right rotation

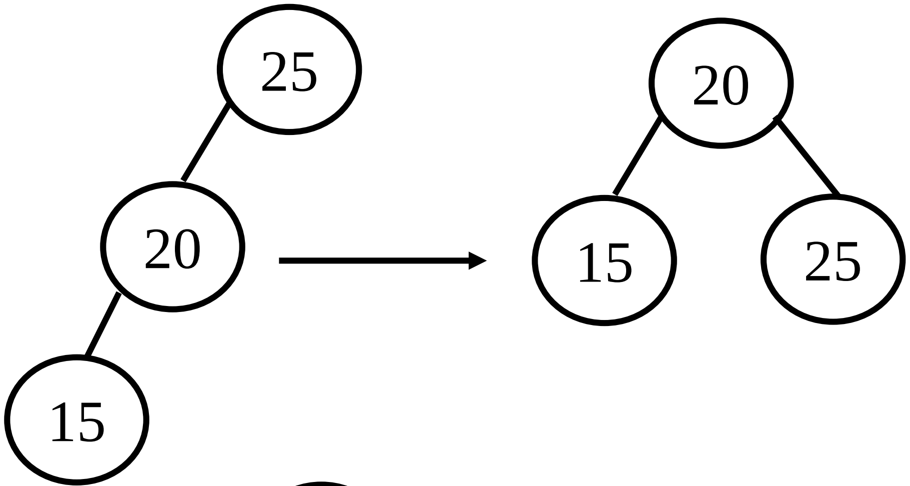


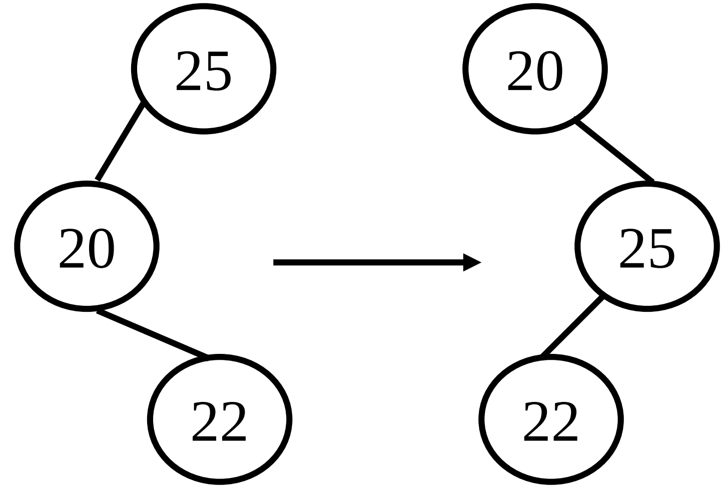
Similarly, we can have a left rotation

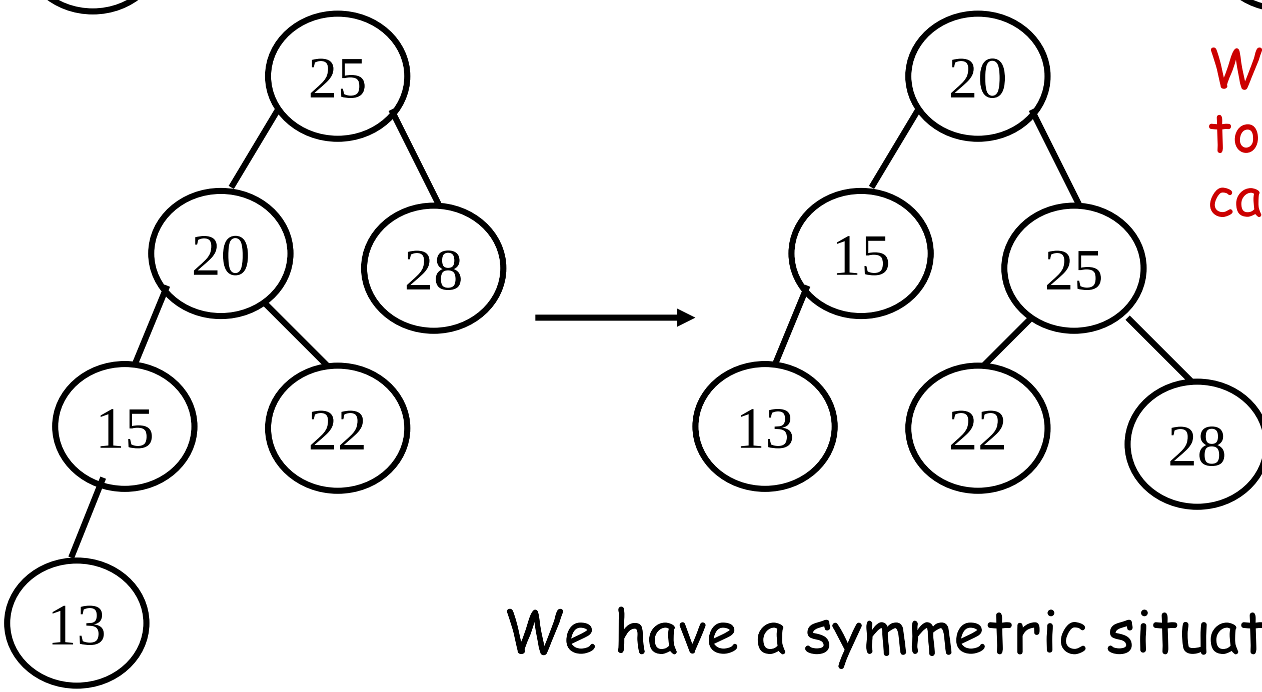
Note:

1) parent-child relationship of 3,5 reversed
2) binary search tree property preserved

Examples:



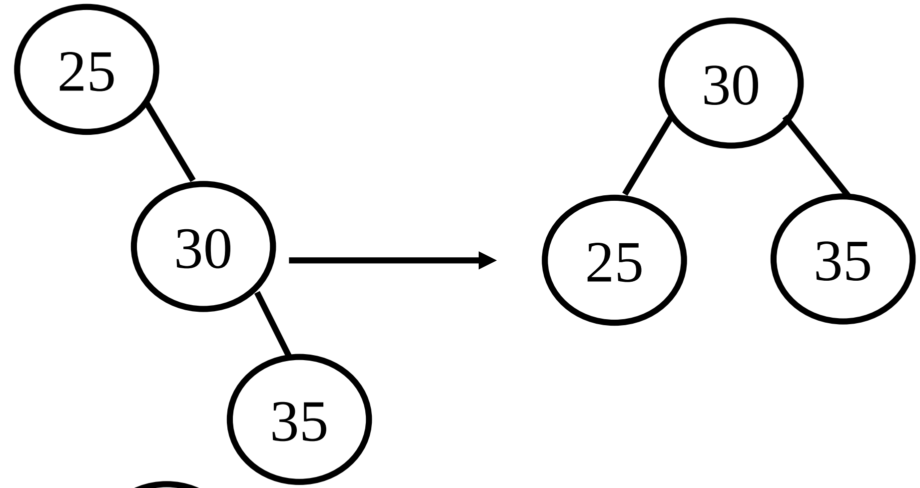




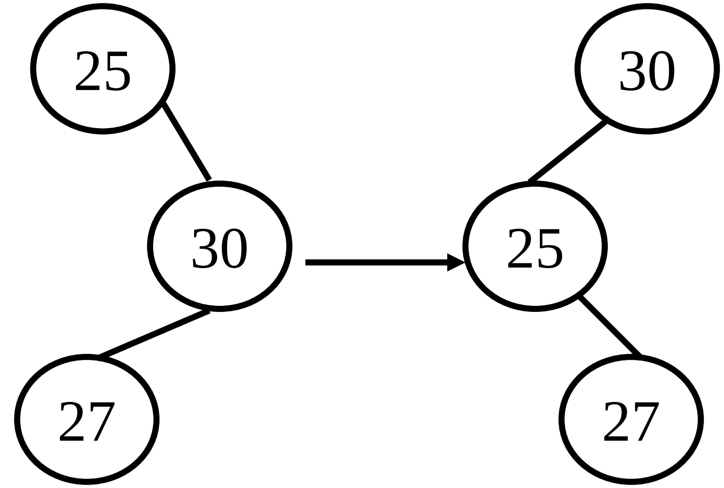
We will see how to handle this case

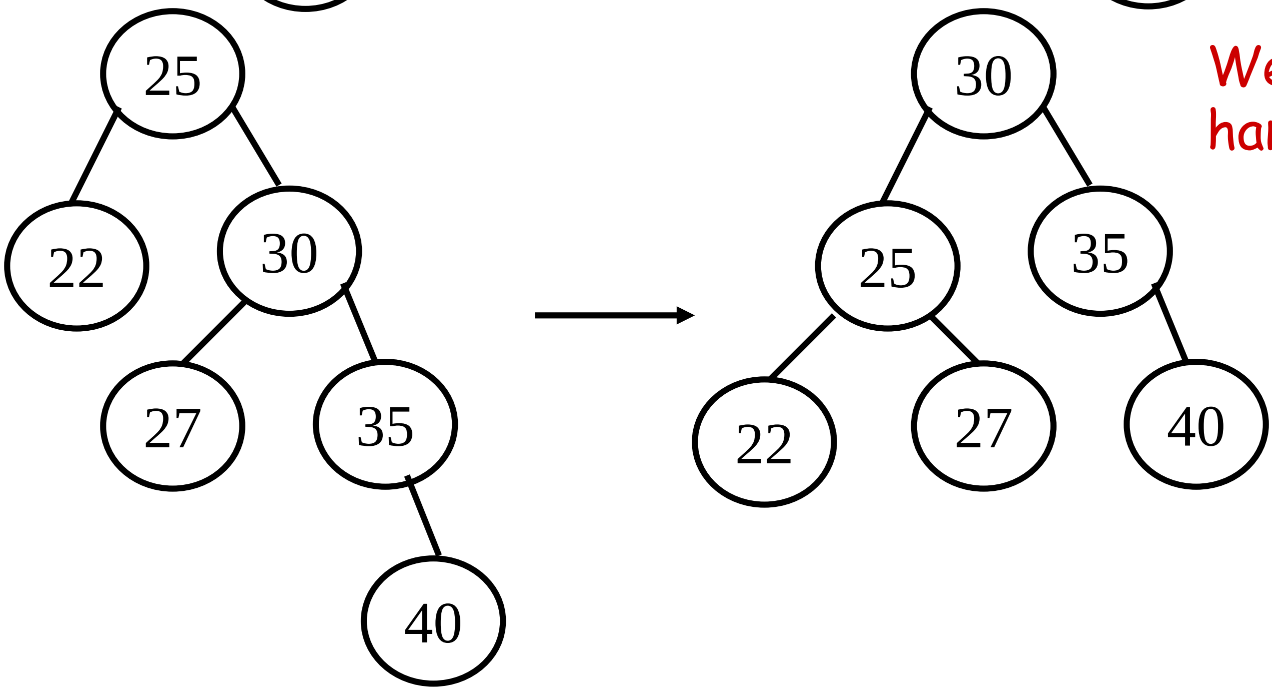
We have a symmetric situation for left rotation!

Examples:



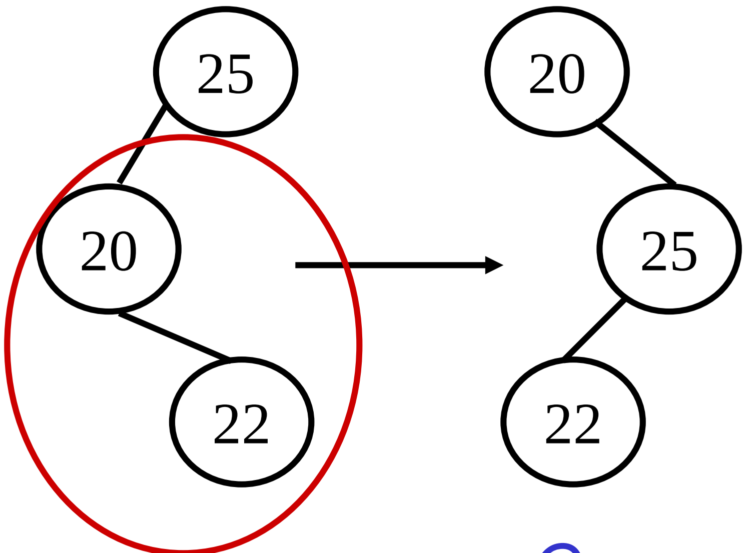
Similarly, this won't work!

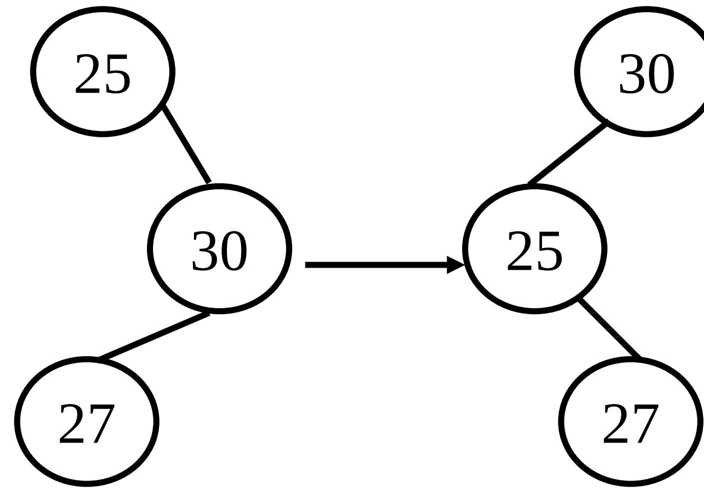




We will see how to handle this case

Two outstanding (symmetric) cases:



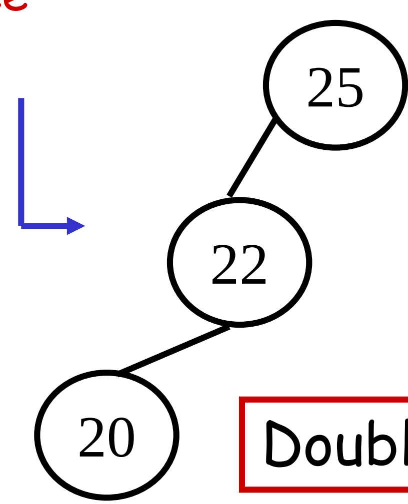


The right subtree of 20 is taller than left subtree

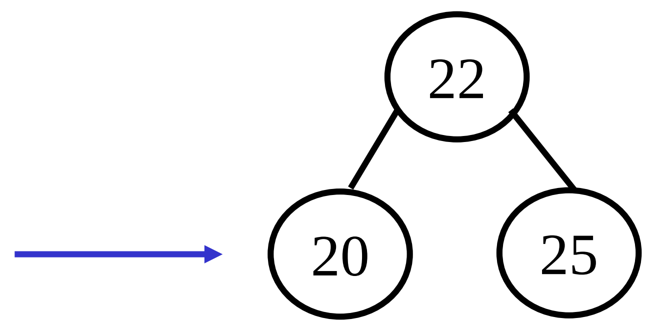
Q: can we rearrange them to avoid this problem?

Yes, we can do a left rotation at 20 first.

Left rotation at 20

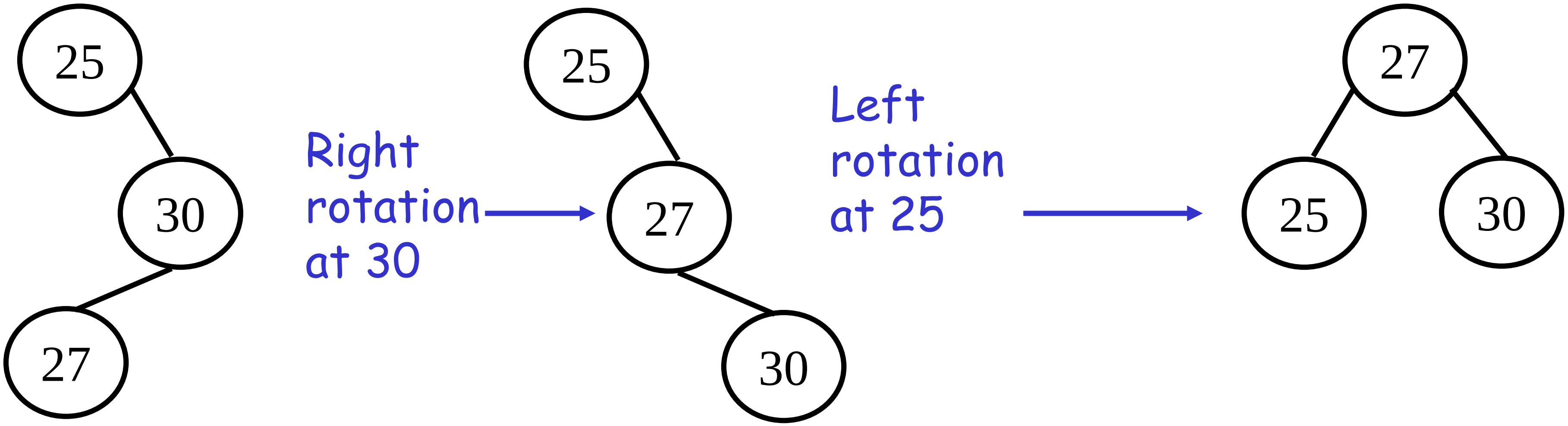


Right rotation at 25

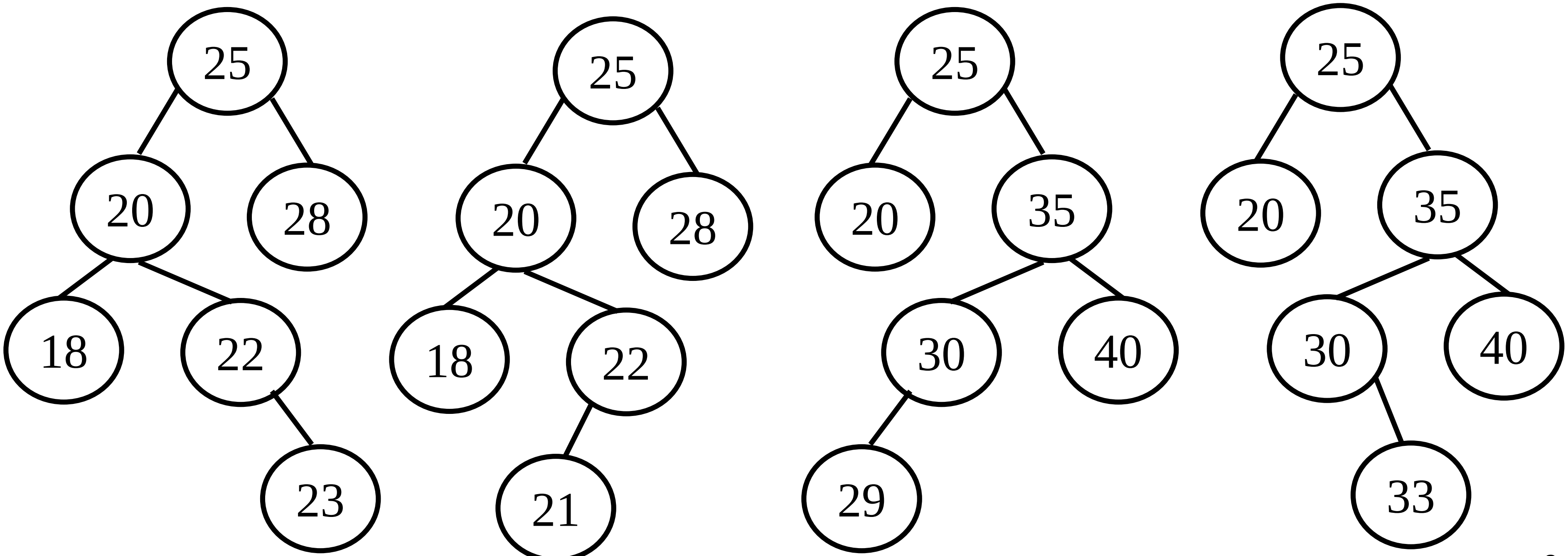


Double rotation: left-right double rotation

Double rotation: right-left double rotation



Do you know how to balance the following cases?

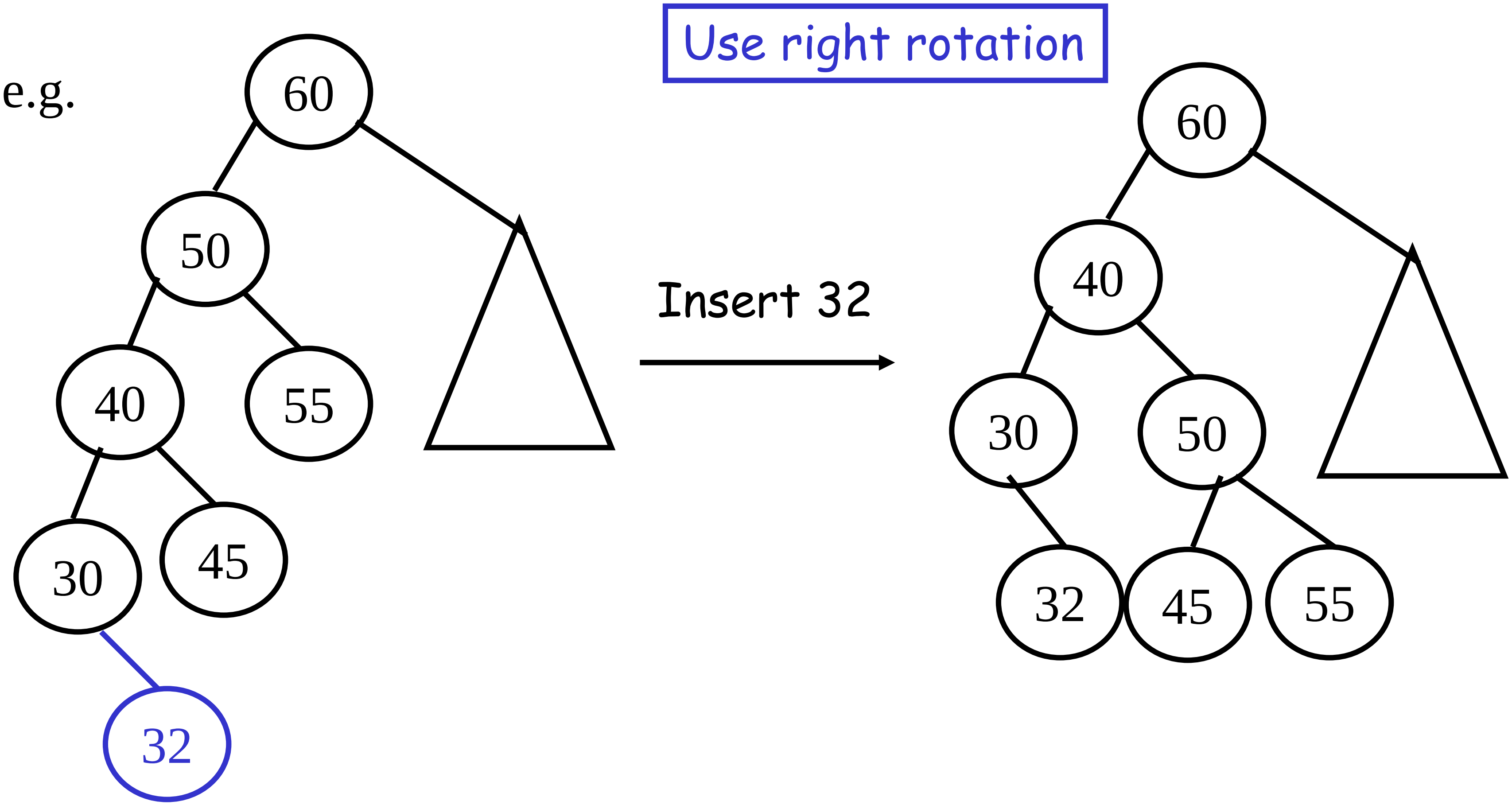


Summary (Insertion for AVL tree)

(1) After insertion, the left subtree of the unbalanced node is too tall.

(a) The new node is added to the left subtree of the left child.

e.g.



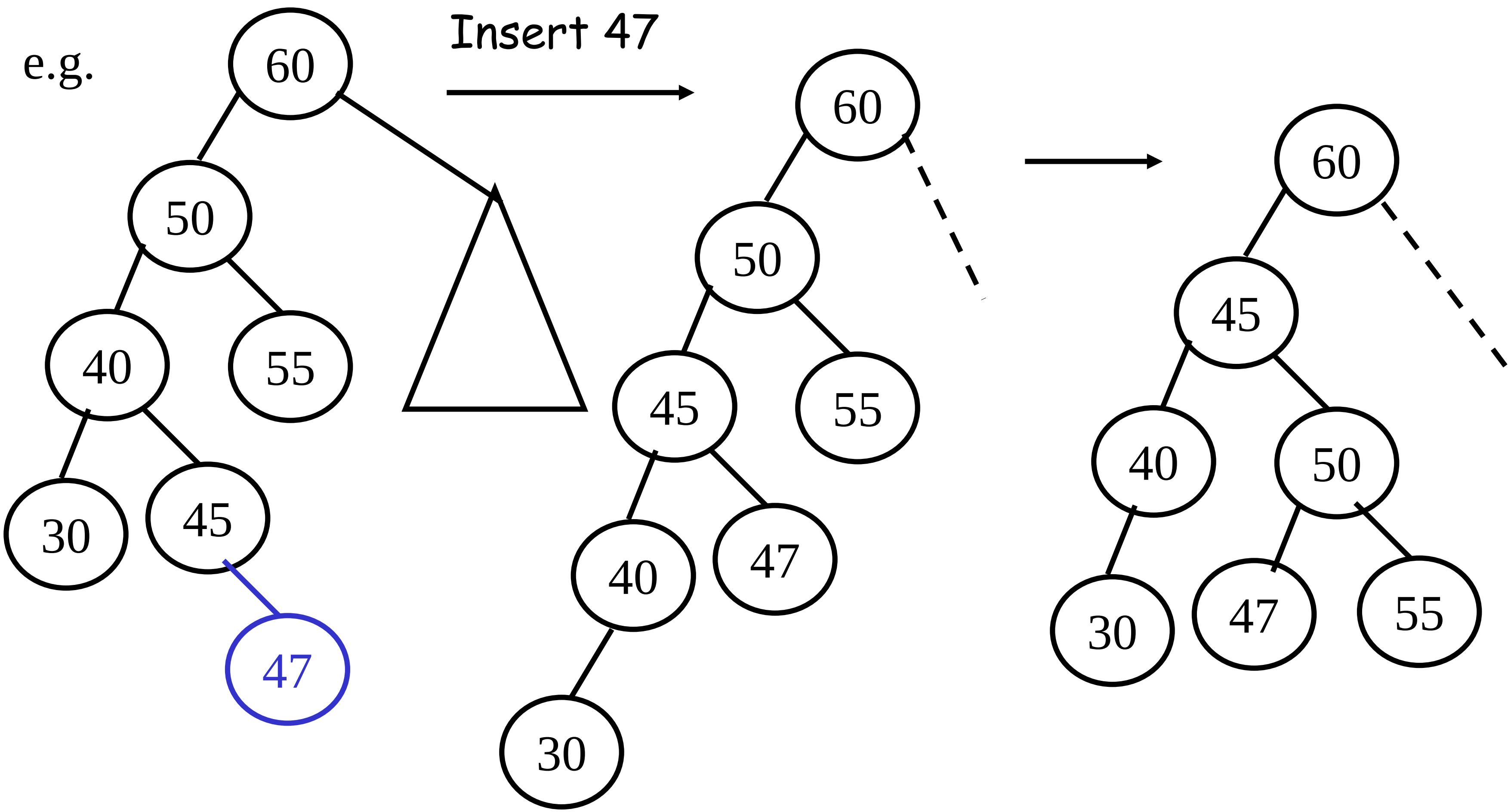
Q: How about inserting 25 instead of 32?

(1) After insertion, the left subtree of the unbalanced node is too tall.

(b) The new node is added to the right subtree of the left child.

Use left-right double rotation

e.g.



Similarly, we have the following two cases:

(2) After insertion, the right subtree of the unbalanced node is too tall.

(a) The new node is added to the right subtree of the right child.

Use left rotation

(b) The new node is added to the left subtree of the right child.

Use right-left double rotation

Exercise:
Insert 100, 56, 3, 8, 10, 30, 40, 50, 25, 46 one by one into an initially empty binary search tree.

Implementation:

Besides pointers to left child, right child, parent and storage for storing the element of the node, we need extra storage for "balance" information of the node.

```
struct node {
    element e;
    int b;              // b is called the balance factor
    node *left;         // -1: right subtree is taller;
    node *right;        // 0: equal height
    node *p;            // +1: left subtree is taller;
}
```

Insertion procedure:

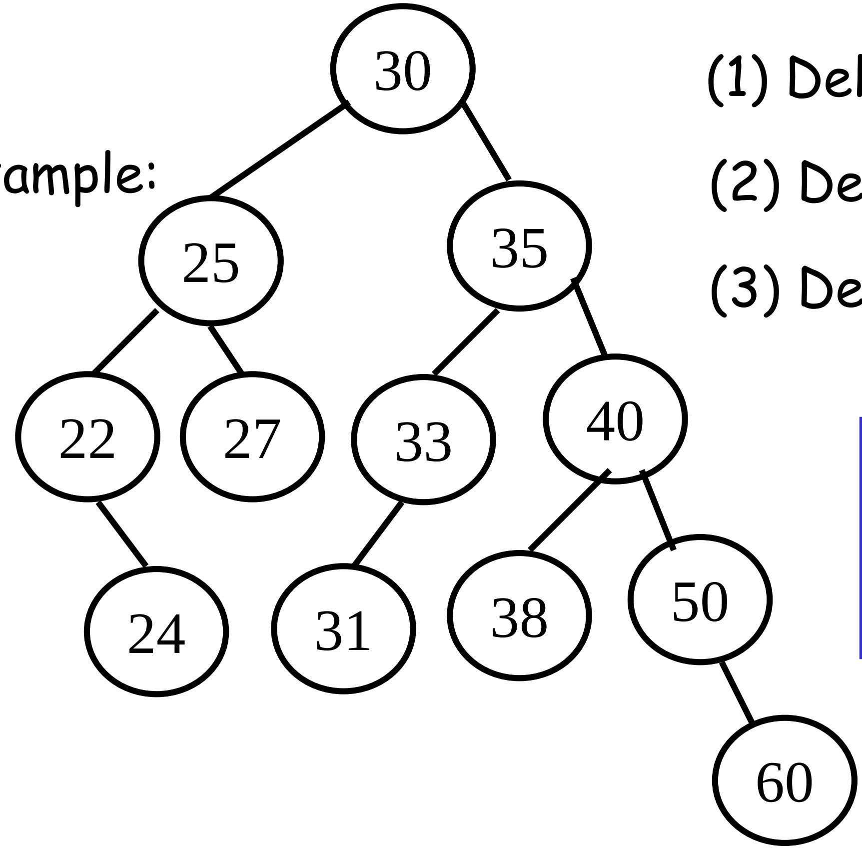
1) Insert the node as in the binary search tree
2) Go up to the root along the path from the inserted node, do the following for each node
   - update the value of b
   - perform rotation to restore balance if the node violates AVL tree property

How about deletion?

Procedure for Deletion:

1) Delete the node as in a binary search tree
2) Go up to the root along the path from the parent of the node just been deleted, do the following for each node
   - update the value of b
   - perform rotation to restore balance if the node violates AVL tree property

Example:



(1) Delete 60

(2) Delete 33

(3) Delete 24

Exercise: Write down the algorithm for deletion and analyze its time complexity

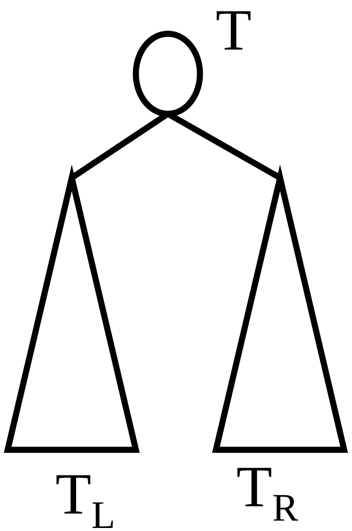
[The MIT book ex. 13-3 (a)]
Prove that an AVL tree with n nodes has height $O(\log n)$.

Hint:

(a) Show that an AVL tree of height h has at least $F(h)$ nodes where
$F(0) = 1$; $F(1) = 2$; $F(h) = F(h-1) + F(h-2)$ for $h \geq 2$
(can you recognize that it is the Fibonacci numbers?)

(b) Then, show that $F(h) \geq \phi^h$ (where $\phi = (1+\sqrt{5})/2$)

Proof of (a): By induction



1 If T is an AVL tree, then T_L and T_R are both AVL trees

2 Since T is an AVL tree, if the height of T is h, then
(a) the heights of T_L and T_R are both equal to h-1; or
(b) one of them is h-1 and the other is h-2.

<u>Induction step:</u>
Let h be the height of T.

By (2), without loss of generality, let the height of T_L be h-1 and the height of T_R be at least h-2.

By (1), T_L and T_R are both AVL trees.

By the induction hypothesis, the number of nodes in T_L is at least F(h-1) while the number of nodes in T_R is at least F(h-2).
So, the number of nodes in T is at least F(h-1) + F(h-2) = F(h)

(* you should be able to fill in other details *)

Proof of (b), $F(h) \geq \phi^h$ where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$

Again, by induction.

Induction step (fill in other details yourself)

$F(h) = F(h-1) + F(h-2)$

$\geq \phi^{h-1} + \phi^{h-2}$

$\geq \phi^h (\phi^{-1} + \phi^{-2})$

$\geq \phi^h$

Note that

$$\frac{1}{\phi} + \frac{1}{\phi^2} = \frac{\phi+1}{\phi^2} = \frac{\frac{1+\sqrt{5}}{2}+1}{(\frac{1+\sqrt{5}}{2})^2} = 1$$

Now, we show that $h = O(\log n)$

$n \geq F(h) \geq \phi^h$

$\Rightarrow \log n \geq h \log \phi$

$\Rightarrow h \leq 1.44 \log n$

In other words, $h = O(\log n)$

A loose bound:

$n \geq F(h) = F(h-1)+F(h-2)$

$> 2F(h-2)$

$> 2(2F(h-4)) = 2^2 F(h-4)$

$> 2^{h/2} F(0) = 2^{h/2}$

Then,

$n > 2^{h/2}$

$\Rightarrow \log n > h/2$

$\Rightarrow h < 2 \log n$

i.e., $h = O(\log n)$