**BST:**
- `contains` \( \text{Time} = O(\text{depth}) \)
- `insert` \( = O(n) \) worst
- `remove` \( = O(\log n) \) average

If we could guarantee depth is always \( O(\log n) \), could get worst-case \( O(\log n) \) times.

**Balanced Binary Search Tree (AVL Trees):**
- BST with worst-case depth of \( O(\log n) \)
- Specifically: for every node \( n \), heights of left and right subtrees of \( n \) differ by at most \( 1 \)

**Ex:** AVL Tree  
**Not** AVL Tree (why not? 6)

Operations are same as in BST.  
If after operation, tree becomes imbalance, tree is rebalanced to have depth \( O(\log n) \)
Since depth is worst-case $O(\log n)$, all operations take $O(\log n)$ in the worst-case:

- insert
- remove

Time: $O(\text{depth}) = O(\log n)$ (worst & avg)

**How to rebalance?**

**Ex: insert**

1. Identify $a = \text{deepest imbalanced node}$
   - $a$ is the deepest node whose left & right subtrees have height difference $> 1$

2. Identify type of insert

**4 Types of inserts**

1. Right
2. Left
3. Right
4. Left

(Above example was inner insert)

**Inserts**

- "outer"  
  - Insert  
  - Requires single swap

- "inner"  
  - Insert  
  - Requires double swap
'inner example'

\[ \begin{array}{c}
\text{insert into left subtree of} \\
\text{a's right child.}
\end{array} \]

Step 3: Perform swap(s) to rebalance.

For outer: Single swap
1. swap a \( \rightarrow \) a's child

For inner: Double swap
1. swap a's child + grandchild
2. swap a \( \rightarrow \) a's new child.

ex (1)

\[
\begin{array}{c}
\text{AVL} \\
\text{insert (45)}
\end{array}
\]

Step 1: d? = 30

(Chain outer insert (chain chain).)
Step 3: swap 30 + 35

insert(45)

3. pointer changes:
- a's parent (right ptr) → a's child 20 → 35
- a's right child (left ptr) → a 35 → 30
- a's ... (right ptr) → a's successor 30 → 33

left/right switched for other side

ex.2 insert(4)

swap a (10) + a's child (5)

- a's parent (left ptr) → a's child 20 → 5
- a's left child (right ptr) → a 5 → 10
- a's new (left ptr) → a's successor 10 → null
AVL?  

insert (34)

AVL?  No  \( a \) ?  20

Which type of insert? inner.

⇒ left subtree of \( a \)'s right child.

double rotation - swap \( a \)'s child and grandchild (on side of imbalance).

swap \( a \) and \( a \)'s new child

After 1st swap:

swap 35 + 30

3 ptr changes: <from original tree>

\( a \)'s (right) \( \rightarrow \) \( a \)'s right child's left child  
20 \( \rightarrow \) 30

\( a \)'s grandchild (right ptr) \( \rightarrow \) \( a \)'s child  
30 \( \rightarrow \) 35

\( a \)'s child (left ptr) \( \rightarrow \) it's (prev) predecessor  
35 \( \rightarrow \) 33
After 2nd swap

Swap: 20 \rightarrow 30

3 ptr changes

<from previous tree>

a's parent's (right ptr) \rightarrow a's child
root \rightarrow 30

a's child's (left ptr) \rightarrow a
30 \rightarrow 20

a's (right ptr) \rightarrow a's (prev) successor
20 \rightarrow 25

Another example

\begin{align*}
\text{insert (9)}
\end{align*}

Type of insert? inner
(right subtree of a's left child)
Double rotation:

1. Swap α's child and grandchild.

```
      14
     / \
    12   15
   / \    \
  8   13   16
 /     /     \
7      9      13
      /     \
     5      7
```

2. Swap α and α's new child.

```
      14
     / \
    8   15
   /     \
  7      12   16
 /     /     \
5      9      13
```