**SELF-BALANCING SEARCH TREES**

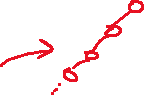
These trees make sure that height is logarithmic.

After an insertion or deletion, tree balances itself.

The performance of a bst is proportional to the height of the tree

A perfect binary tree of height k can hold - 1 items

If a bst is perfect and contains n items, the expected performance is O(logn)



However if bst is not perfect, actual performance is worse (can be O(n) for the worst case)

To solve this problem, we introduce self-balancing trees to achieve a balance so that the heights of the right and left subtrees are equal or nearly equal

Perfect tree 🡪 height of left and right subtrees are same

**TREE BALANCE AND ROTATION**

Why balance is important?

Searches into this unbalanced search tree are O(n), not O(logn):

Chart, scatter chart

Description automatically generated

Diagram

Description automatically generatedA realistic example of an unbalanced tree:

“The quick brown fox jumps over the lazy dog”

Take each word in this sentence one by one and add to the bst at the left.

**Rotation**

We need an operation on a binary tree that changes the relative heights of left and right subtrees, but preserves the bst property

A picture containing watch

Description automatically generated

Arrayimiz var diyelim, bu arraydeki elemanları tek tek bst’ye ekleyeceğiz. Tek tek eklemek mantıklı olmaz çünkü array sortedsa worst bst elde ederiz.

Root için ortanca değeri seçmeliyiz ki sağ sol subtreeler eşit uzunlukta olsun.

Algorithm for rotation:

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

What we did is rotateRight().

Diagram

Description automatically generated

Text, letter

Description automatically generated

private Node<E> rotateLeft(Node<E> root){

Node<E> temp = root.right;

root.right = temp.left;

temp.left = root;

return temp;

}