COMP 250 - Lecture 27

1) Pairs of sum x

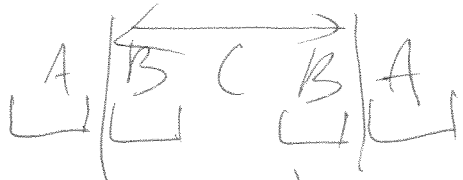
Sort $\Rightarrow O(n \log n)$

```
for (int i = 1 to n) {
    if (binarySearch(a, x - a[i])) -> O(log n)
        counter++
}
```

$O(n \log n)$

$\Rightarrow O(n \log n)$

3)



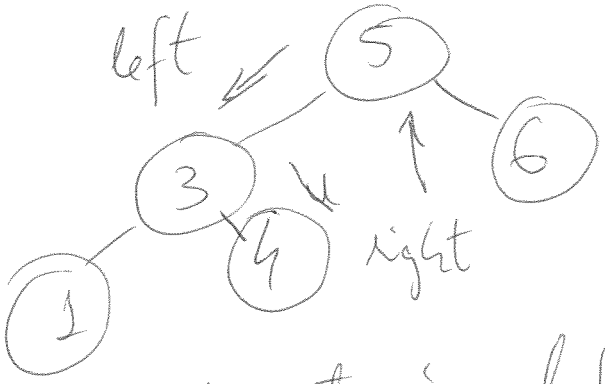
pal(1 ... n) = property && pal(2 ... n-1)

```
sum = a[0] + a[n-1]
check(a, 0, n-1, sum)
```

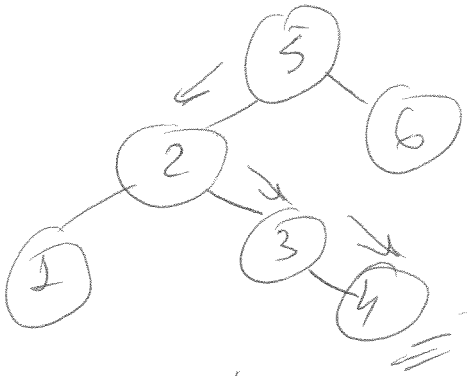
↑ recursive function

$T(n) = c + T(n-2)$ $\quad O(n)$

Removing an element from a binary Search tree

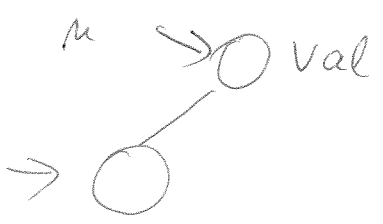


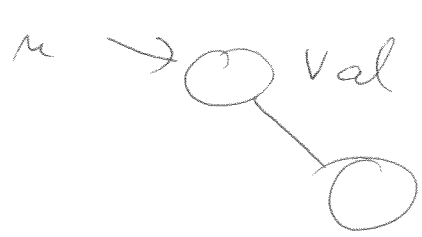
Largest element in left subtree or
Small in right subtree



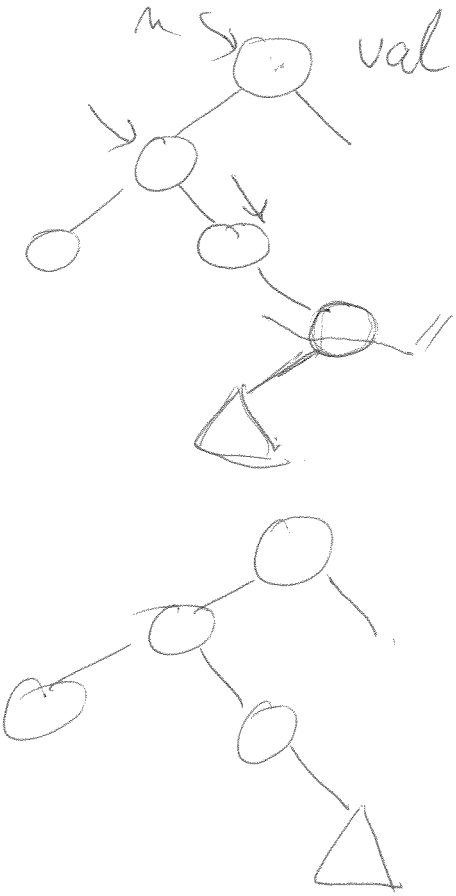
Algorithm remove (BSTnode node, val)
Method in BSTnode

```
BSTNode n = search (node, val)
// returns a reference to the node that contains val,
   or null
if (n == null) return;
if (n.left == null && n.right == null) {
    // at a leaf
    replace (n, null) return;
}
```



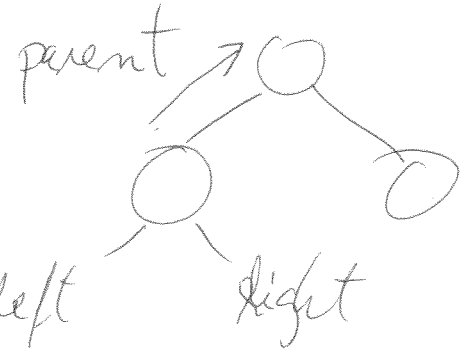


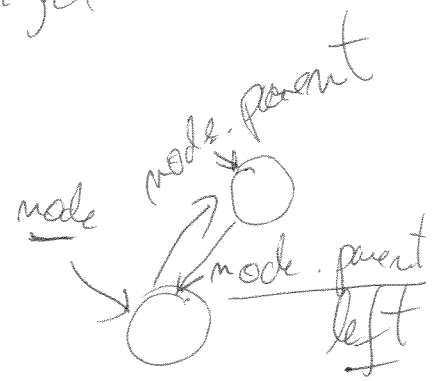

```
if (n.left != null && n.right == null) {
    replace(n, n.left) return; }
if (n.left == null && n.right != null) {
    replace(n, n.right) return; }
// two children → replace w largest el. in left
BSTNode suc = n.left
while suc.right != null
    suc = suc.right
// change n
n.content = suc.content
replace(suc, suc.left);
return
}
```

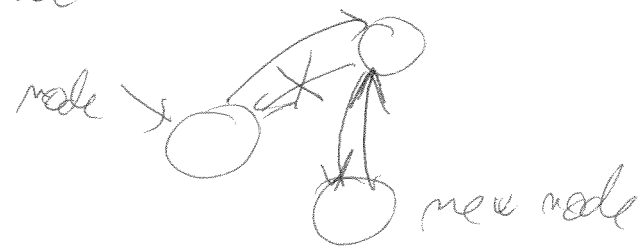


Algorithm replace (BSTNode node, BSTNode newNode)

```
if (node.parent != null) {
  // are we putting the new
  content on left or right?
  if (node.parent.left == node)
    node.parent.left = newNode;
  else
    node.parent.right = newNode;
}
newNode.parent = node.parent
// nullify refs in node
```







O(1)

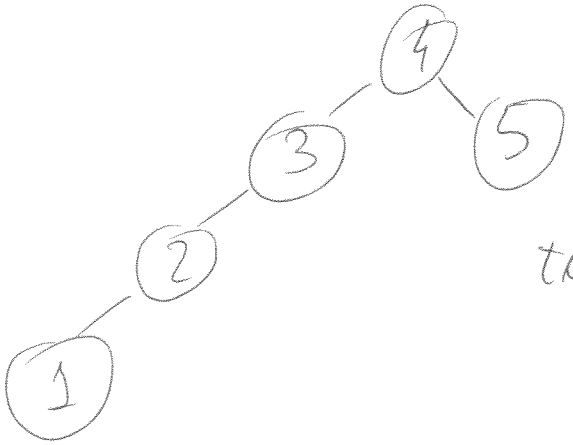
Remove O( ) ; cost is (worst-case) length of longest path down the tree aka height of tree

Balanced tree: $O(\log n)$ n = nr. nodes

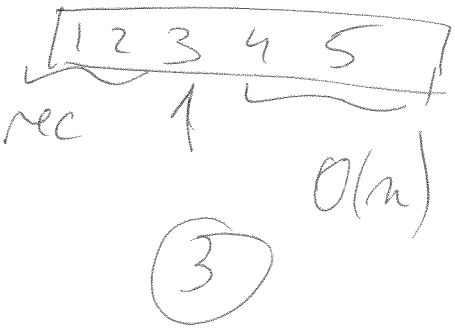
Imbalanced tree: $O(n)$

Keeping Binary Search Trees Balanced:

1)



⇒ in-order traversal $O(n)$



node.traverse()

↳ node.left.traverse()
↳ process(node.content)
↳ node.right.traverse()

2) Balance as we go

Extra information at the node to detect imbalance

Nr. nodes or height of subtrees

AVL tree: Keep difference in height of left and right subtrees to be ≤ 1

↓ another variable at the node

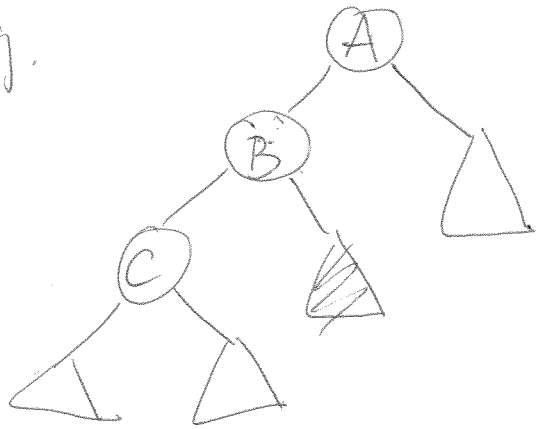
insert / remove need to adjust heights

insert / remove as before (but modify height)

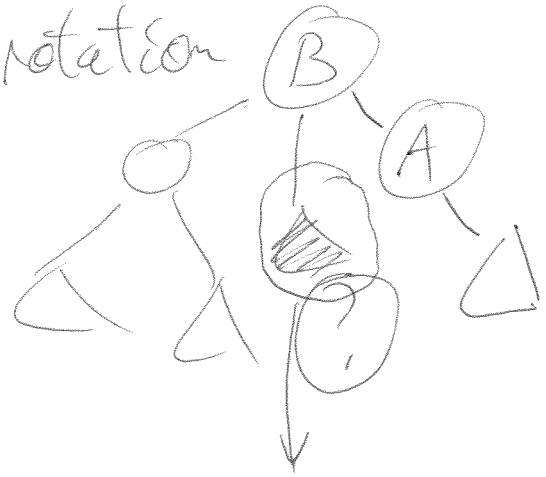
Check if imbalance has happened & reorganize tree

"rotation" operation

E.g.

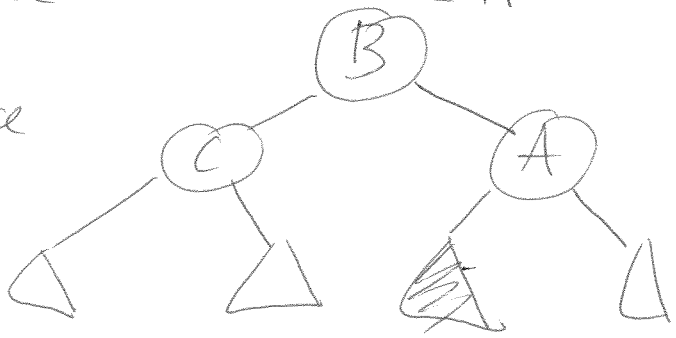


rotation



$\geq B$

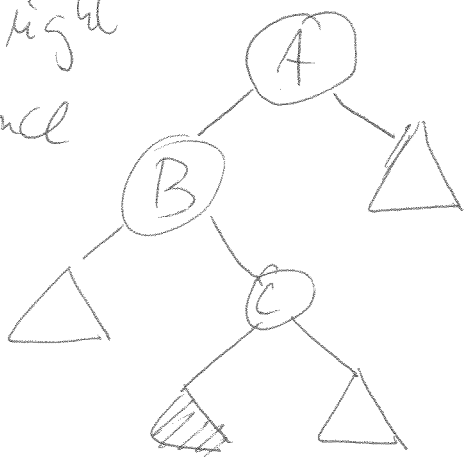
$\leq A$



"Left-left" imbalance

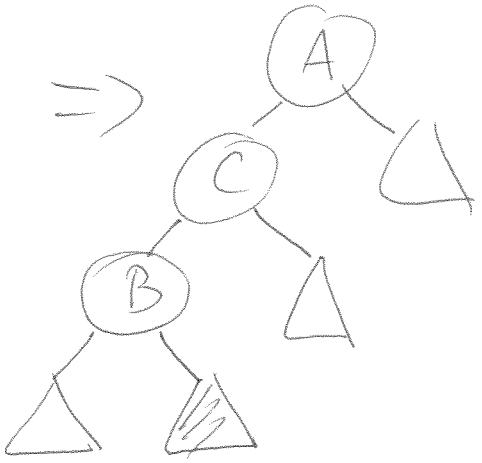
"Right-right" imbalance (similar)

"Left-right" imbalance



$\leq C$

$\geq B$



Left-left imbalance (solve as above)