Chapter 12 Binary Search Trees

12.1 Binary Search Trees are data-structures that support the operations:

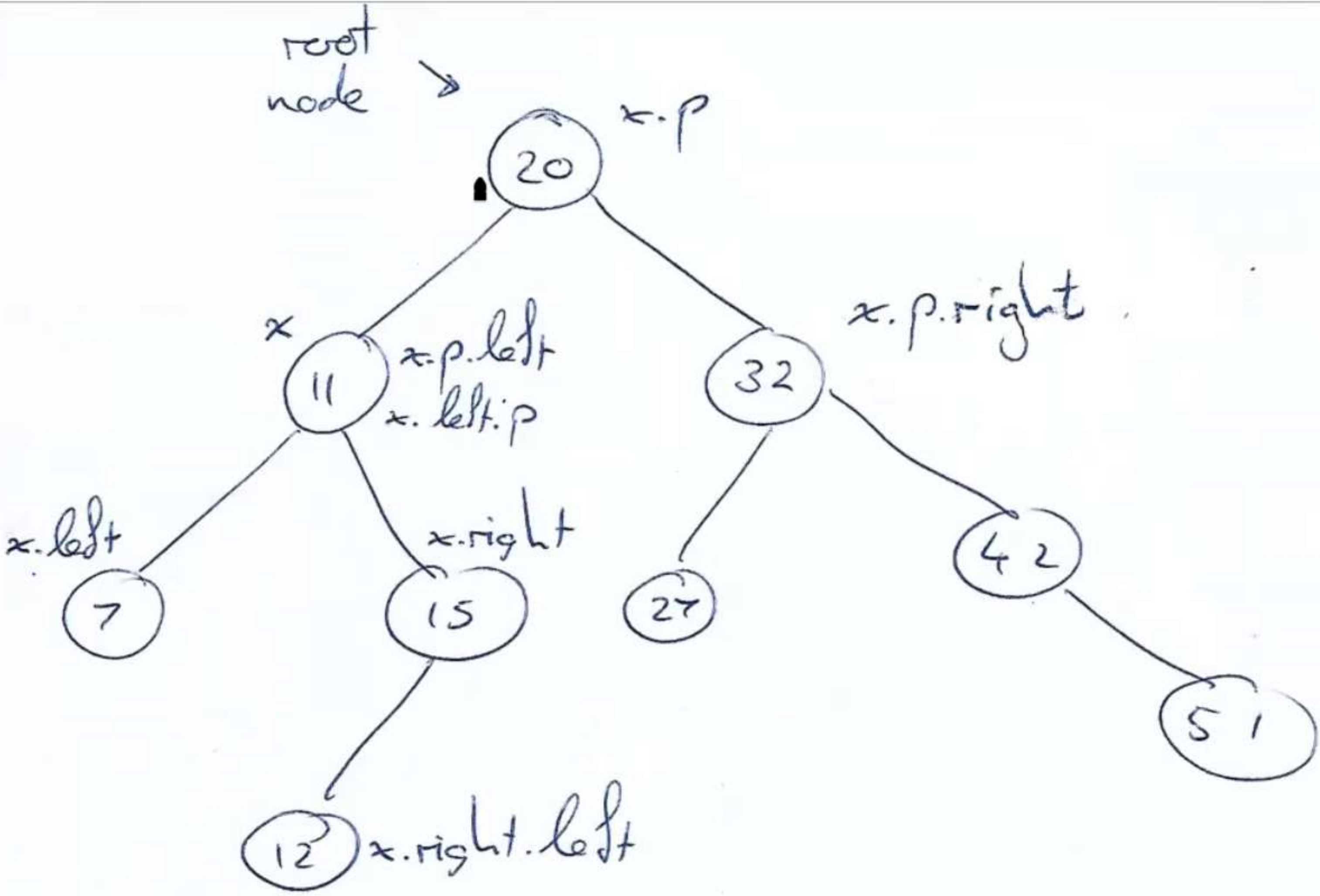
Search, Min, Max

Insert, Delete - (Dynamic)

A Binary Search Trees consists of nodes,
which are usually objects of the
following type:

```
class node
    int key
    node left
    node right
    node p                (p = parent)
```

If x is a node then $x.key$ is its key attribute, $x.left$ a pointer to its left child, etc.



root node
x.p
20
x
x.p.left
x.left.p
11
x.p.right
32
x.left
x.right
7
15
27
42
51
12
x.right.left

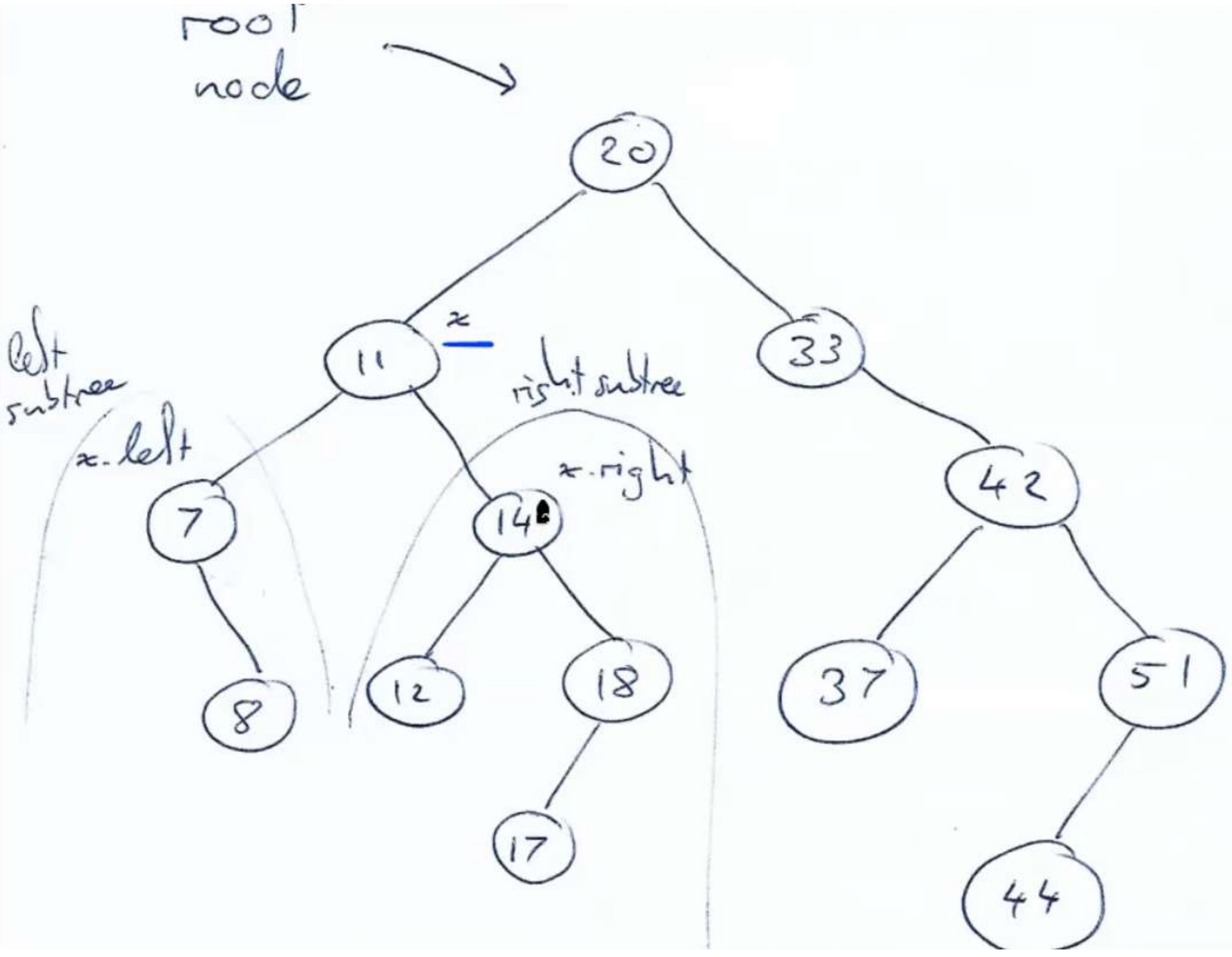
If node y has no left child
then y.left = NIL
If node y has no right child
then y.right = NIL.

Every node has a parent, except the root node.

A binary search tree has the

<u>binary-search-tree property</u>:

if x is any node in the tree and y is a node in the left subtree of x then $y.key \leq x.key$ and if y is a node in the right subtree of x then $y.key \geq x.key$

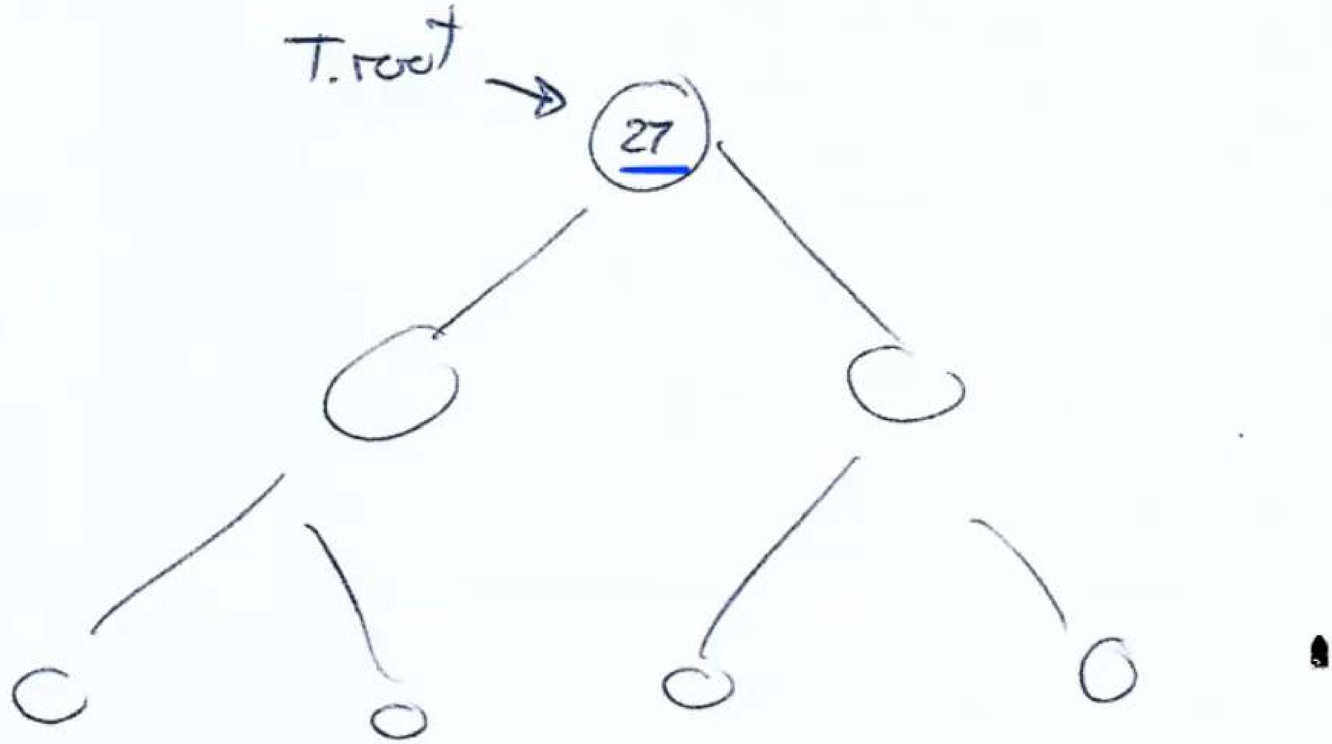

root
node
20
11
x
33
left
subtree
x.left
right subtree
x.right
7
14
42
8
12
18
37
51
17
44

A binary search tree, or BST, is an object of the following type

class BST
    node root.

Say $\underline{T}$ is a BST.
Then T.root is a pointer to
the root node of a tree.

If T.root = NIL then the tree is empty.


T.root
27

Inorder-Tree-Walk (x)

if $x \neq$ NIL

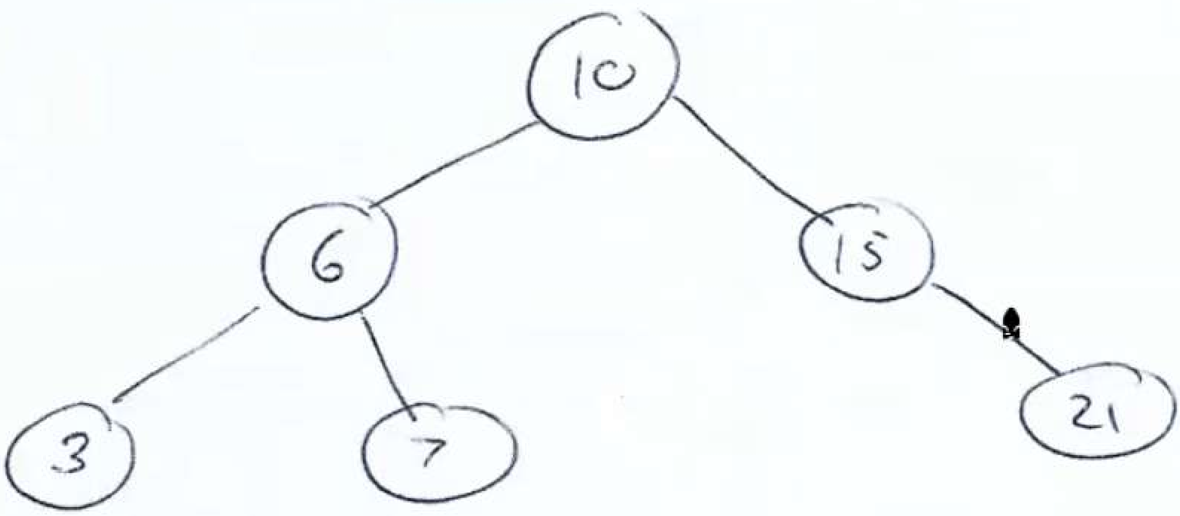
Inorder-Tree-Walk (x.left)

print x.key

Inorder-Tree-Walk (x.right)

The first call is to Inorder-Tree-Walk(T.root)

Trace the algorithm on:



If T is a BST (ie, has the binary-search-tree property) then Inorder-Tree-walk prints the keys of T in ascending order.

In above example you should get

3, 6, 7, 10, 15, 21

Proof: Since Inorder-tree-walk visits every node in the tree, its run-time is $\Omega(n)$, so $T(n) = \Omega(n)$

We show that $T(n) = O(n)$, hence $T(n) = \Theta(n)$

use substitution method

Assume $T(m) \leq cm$ for all $m < n$.

Then $T(n) \leq T(k) + T(n-k-1) + d$
(for some $k < n$.)

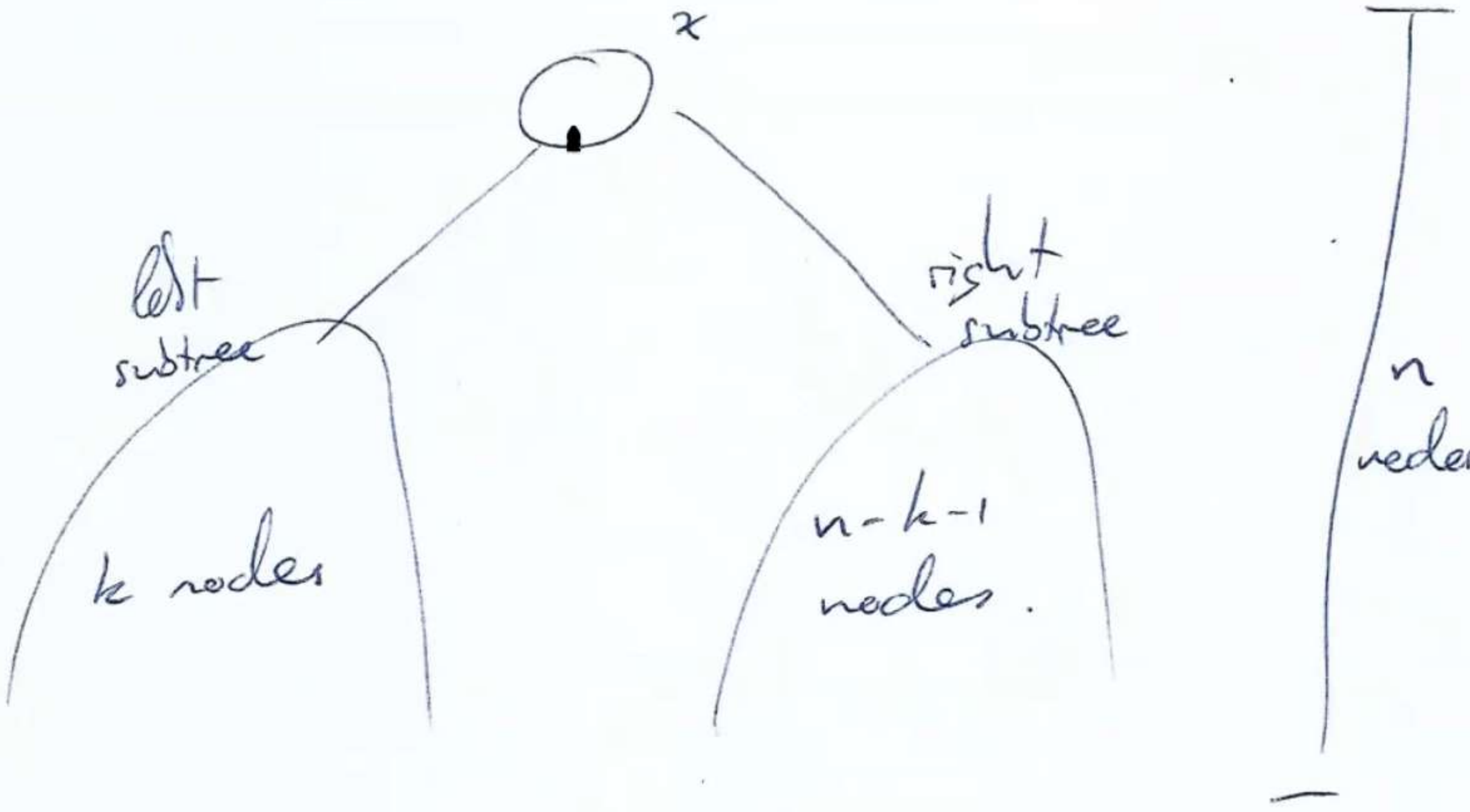
$\leq ck + c(n-k-1) + d$

$= cn - (c-d)$

$\leq cn$ iff $c - d \geq 0$

iff $c \geq d$.

Thus $T(n) = \Theta(n)$


x
left
subtree
right
subtree
k nodes
n-k-1
nodes.
n
nodes