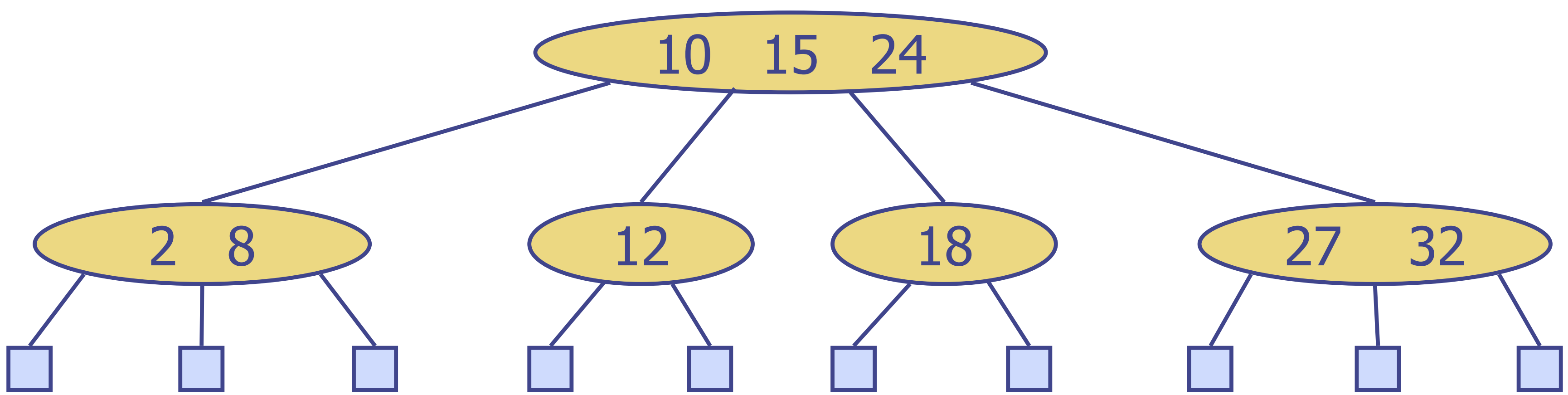
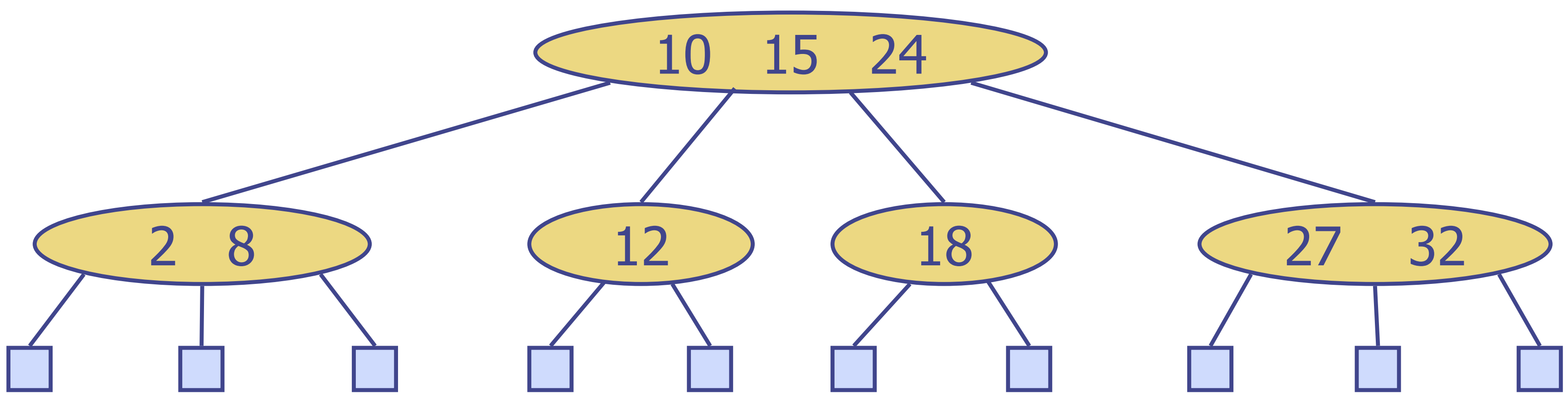
(2,4) Trees

- A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search tree with the following properties



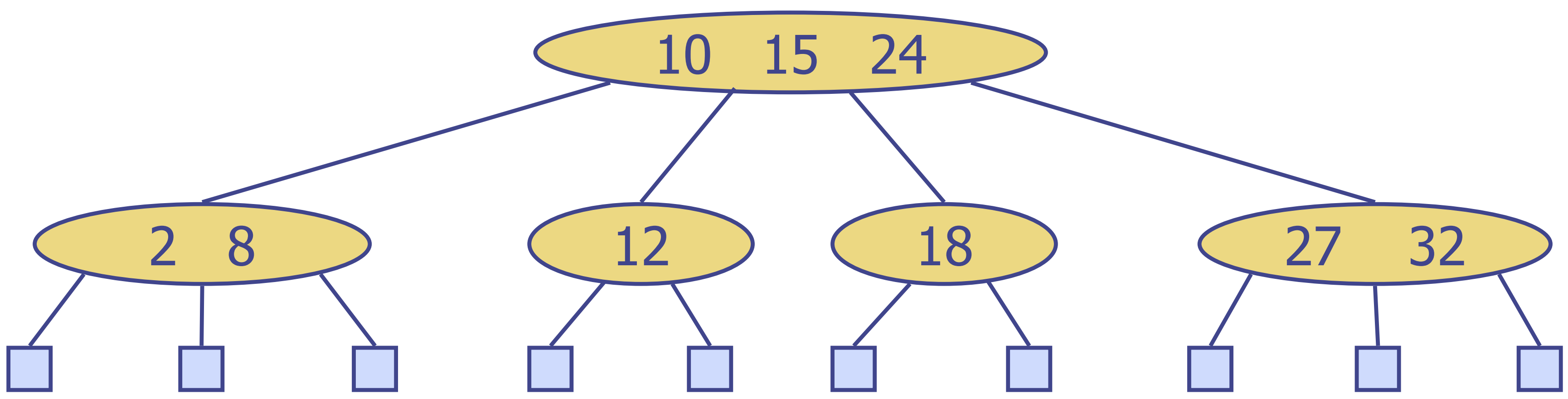
(2,4) Trees

- A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search tree with the following properties
  - Node-Size Property: every internal node has 2, 3, or 4 children



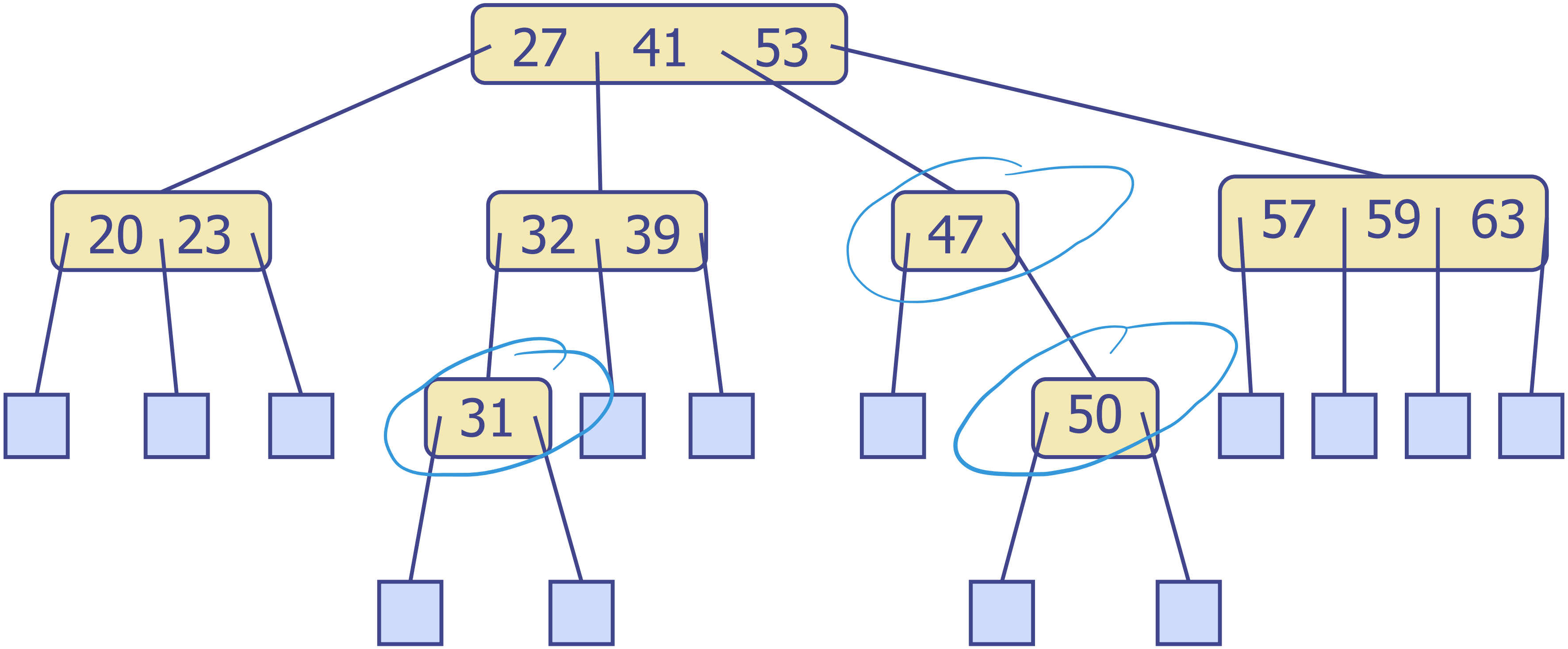
(2,4) Trees

- A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search tree with the following properties
  - Node-Size Property: every internal node has 2, 3, or 4 children
  - Depth Property: all the leaves are in the same level



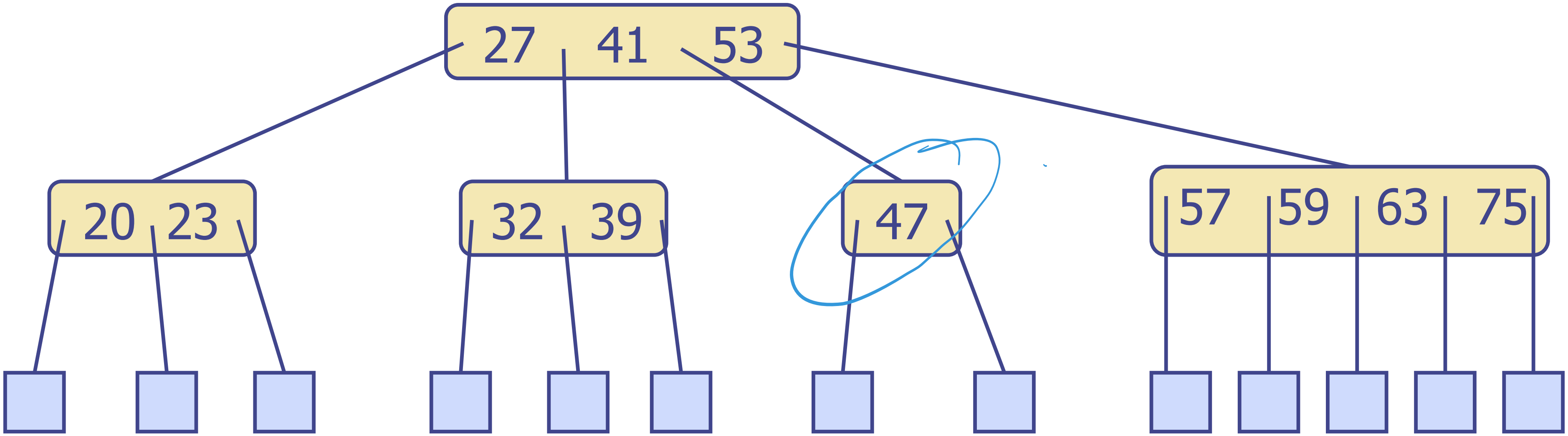
(2,4) Tree?

X



(2,4) Tree?

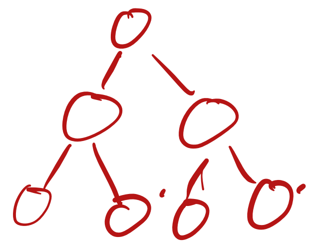
X



What is the Maximum Height of a (2,4) Tree?

max case: each node has only 2 nodes.

Height	Node	Keys
0	2^0	2^0
1	2^1	2^1
2	2^2	2^2
$\vdots$	$\vdots$	$\vdots$
h	2^h	2^h.



$$n = 2^0 + \cdots + 2^h$$

$$= \sum_{i=0}^{h} 2^i = \frac{2^h - 1}{2-1} \cdot 1$$

the first term is 2^0 rather than 2^1.

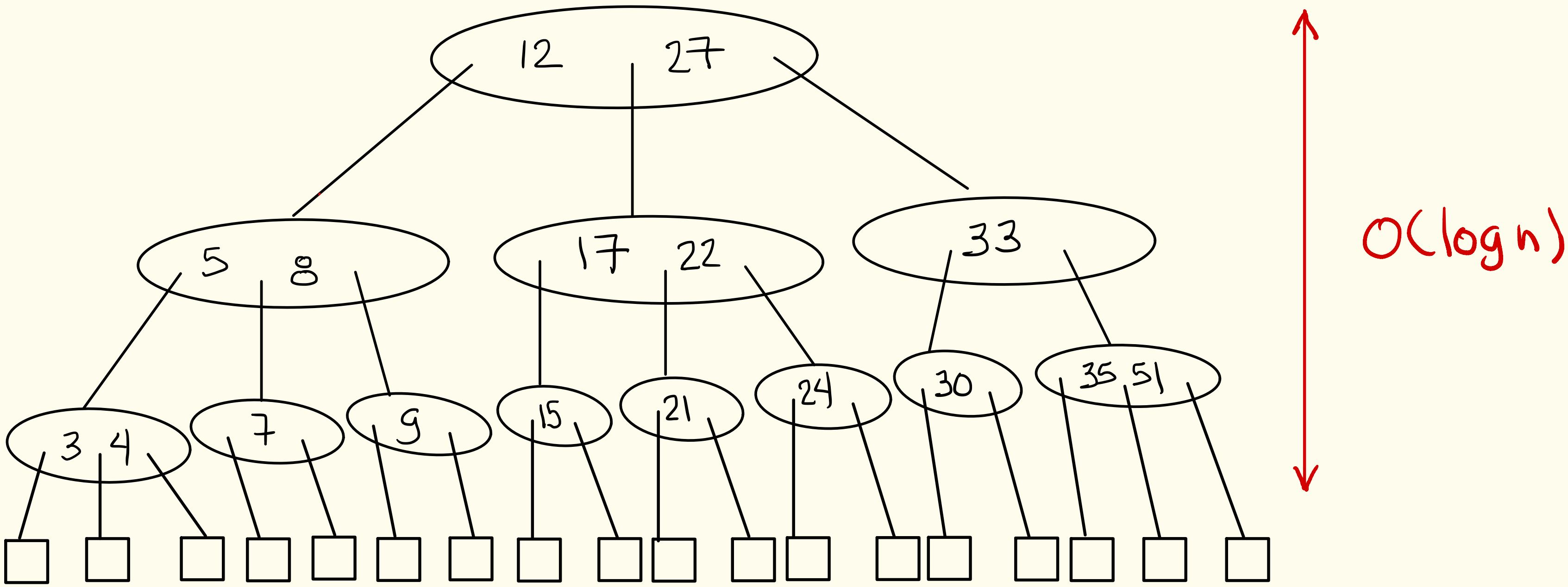
$$n = 2^{h+1} - 2$$

$$h = \log_2(n+2) - 1 \quad \Rightarrow \quad O(\log n)$$

What is the Maximum Height of a (2,4) Tree?

Build the tallest possible (2,4) tree with n keys : $K_1, K_2, \ldots, K_n$

Implementing an Ordered Dictionary with a (2,4) tree



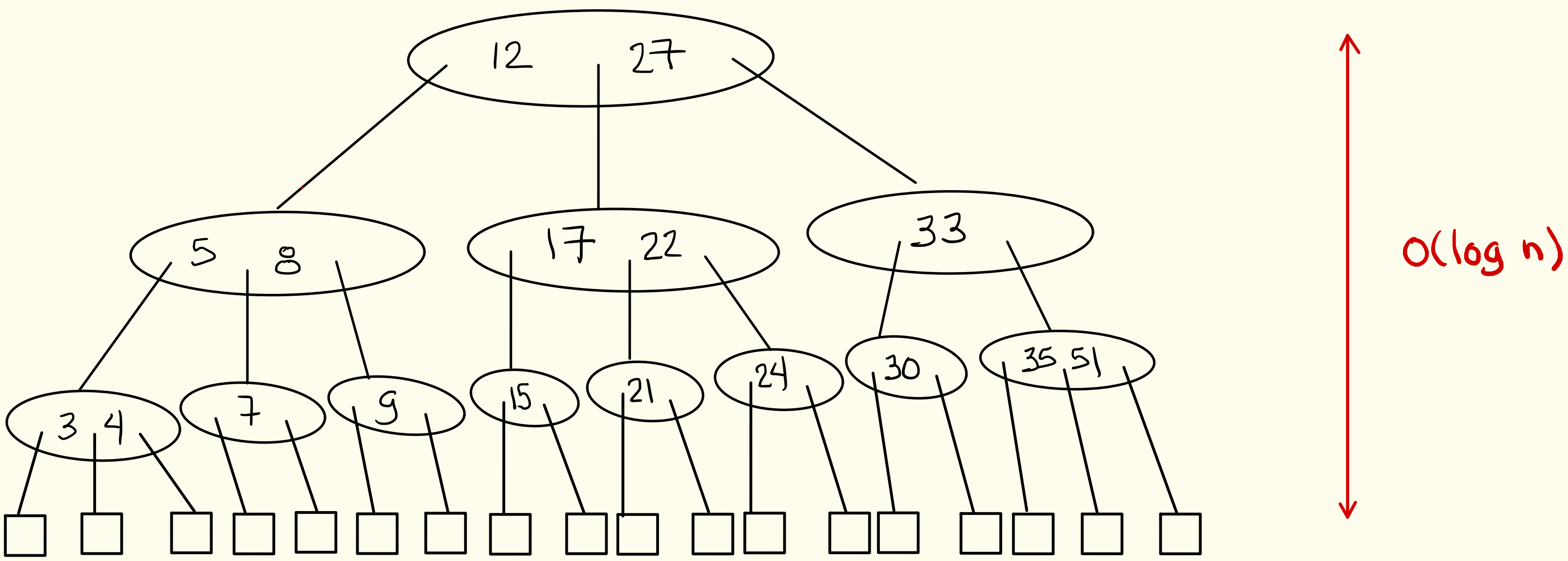
O(log(n))

get(k)
smallest()
largest()
successor(k)
predecessor(k)
put(k,d)
remove(k)

Ordered Dictionary Operations on a Multiway Search Tree of Degree d

smallest	O(height)
largest	O(height)
get	O(height $\times$ log d)
successor	O(height $\times$ log d)
predecessor	O(height $\times$ log d)
put	O(d + height $\times$ log d)
remove	O(d + height $\times$ log d)

Implementing an Ordered Dictionary with a (2,4) tree



get(k)
smallest()
largest()
successor(k)
predecessor(k) } O(log n)
put(k,d)
remove(k)

insert 12

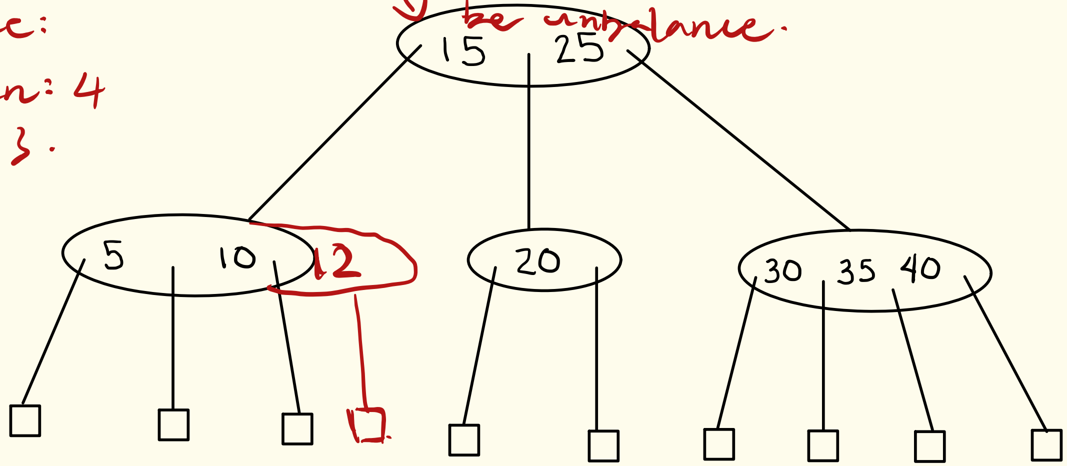
in a 2,4 tree:

max #children: 4

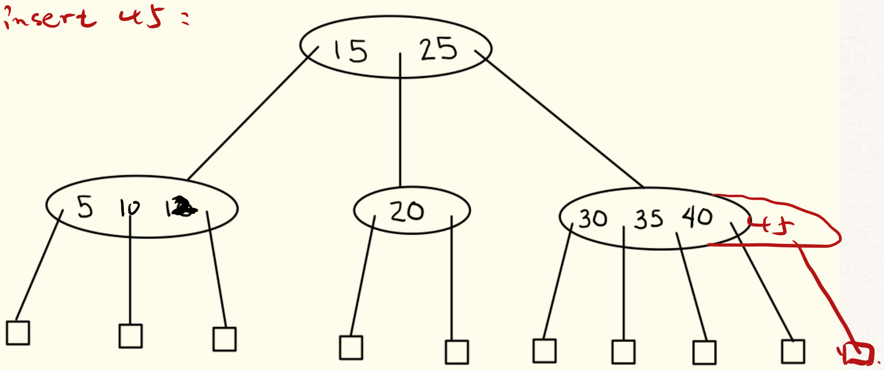
max #keys: 3.

it cannot be added here because it will be unbalance.

children: #key + 1.

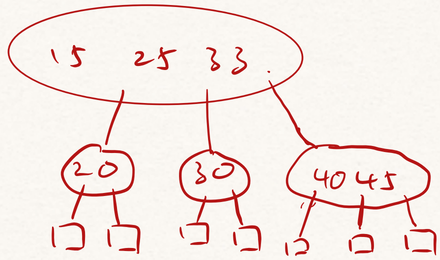


insert 45 :



=> split when the number of node is more than 3 (overflow).

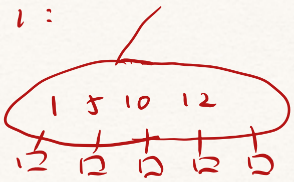
=>



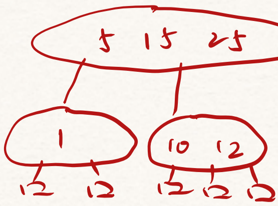
Split: ↑ until the root

=> O(height) = O(log n)

insert 1 :

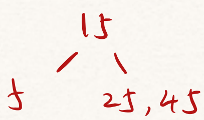


=> split



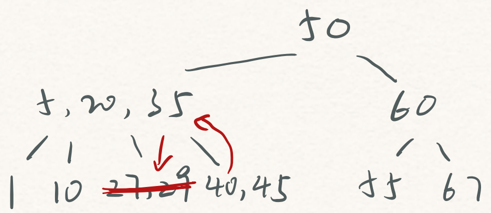
if the root is full and overflow

=> the root is splitted and a new root is generated.



overflow => split

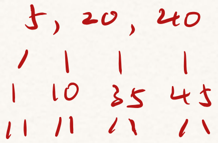
underflow => { transfer
fusion



remove: 27 => just remove it from the list.

remove: 29 => 1. remove this child node

=> not all leaves are at a same height. (underflow)



2. to fix the tree, move the next smallest key in parent node to the empty list and move the smallest key in the next available right node to parent node

remove 45: fusion 35, 40, 45 into the same node.

if the parent occurs an underflow, another fusion operation would preferred in parent node.

* transfer has a higher priority than fusion. fusion occurs only if all its siblings have only one (1) node.

if the root is empty

=> get a new root from its children.

All algorithum above are of $O(\log n)$.

why using 2,4 tree instead of AVL tree?

=> it is good for the case that some data

B tree: