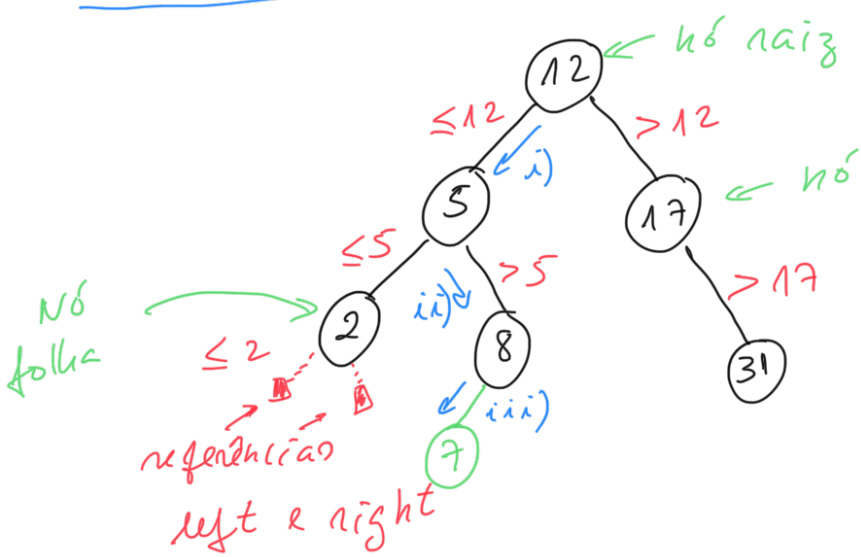
BST – Binary Search Trees

BST são árvores binárias de pesquisa → estrutura de dados em árvore, ordenada, em que cada nó contém dois filhos, no máximo

Também existem árvores não ordenadas e árvores com n filhos (n-árias)

Exemplo de BST – inserção

Inserir (7)



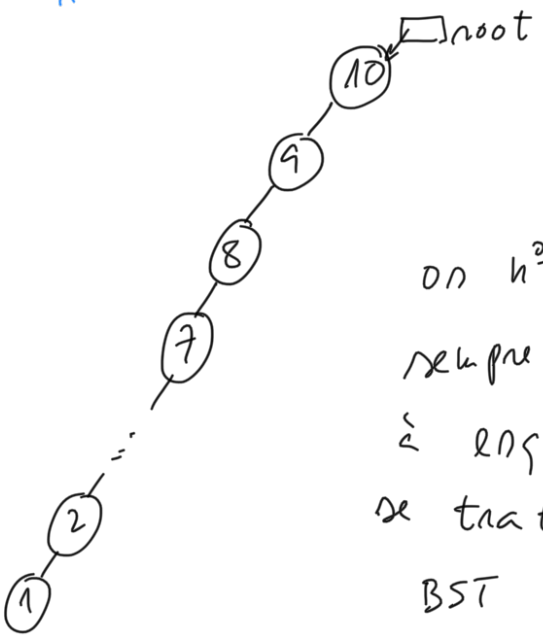
Inserir (7):

i) 7 < 12: inserir na subárvore esquerda

ii) 7 > 5: inserir na subárvore direita

iii) 8 é folha:
- cria nova folha com valor (7)
- 7 < 8: a nova folha é subárvore esquerda

E se inserirmos os nºs de 10 .. 1 numa árvore BST vazia?



os nºs não serão sempre inseridos à esquerda porque se trata duma BST (está ordenada)

PROBLEMA: Árvore não está balanceada ⇒ custo pesquisa $O(N)$ (equivalente a lista ligada)

Se a árvore estiver balanceada ⇒ custo $O(\log_2 N)$

```
fun <E> searchR(root: Node<E>?,
                elem: E,
                cmp: Comparator<E>)
        : Node<E>? {
    var root1 = root
    if (root1 == null)
        return null
    val c = cmp.compare(elem,
                        root1.item)
    if (c == 0)
        return root1
    if (c < 0)
        return searchR(root1.left, elem, cmp)
    else
        return searchR(root1.right, elem, cmp)
}
```

BST $\mathcal{O}(\log_2 n)$

```
class Node<E> {
    var item: E?
    var left: Node<E>?
    var right: Node<E>?
    constructor() {
        item = null
        left = right = null
    }
    constructor(elem: E) {
        item = elem
        left = right = null
    }
}
```

(2.)

E se não fosse BST, como fazia o search?

não está ordenada, custo $\mathcal{O}(n)$

```
return searchR(root1.left, elem, cmp)
    || searchR(root1.right, elem, cmp)
```

"OR" - A OR B → avalia "A"; se A for true retorna true; caso contrário avalia B.

OR short circuit

BST – pesquisa de forma iterativa

```
fun <E> searchIterative(n: Node<E>?, elem: E,
                        cmp: Comparator<E>): Node<E>?
{
    var root = n
    while (root != null) {
        val c = cmp.compare(elem, root.item)
        if (c == 0)
            break
        if (c < 0)
            root = root.left  // esquerda
        else
            root = root.right // direita
    }
    return root
}
```

BST - Inserção


```
fun <E> insertR(root: Node<E>?, elem: E,
                cmp: Comparator<E>): Node<E>
{
    if (root == null)
        return Node<E>(elem)
    if (cmp.compare(elem, root.item) < 0)
        root.left = insertR(root.left, elem, cmp)
    else
        root.right = insertR(root.right, elem, cmp)

    return root
}
```

devolve a árvore após a inserção