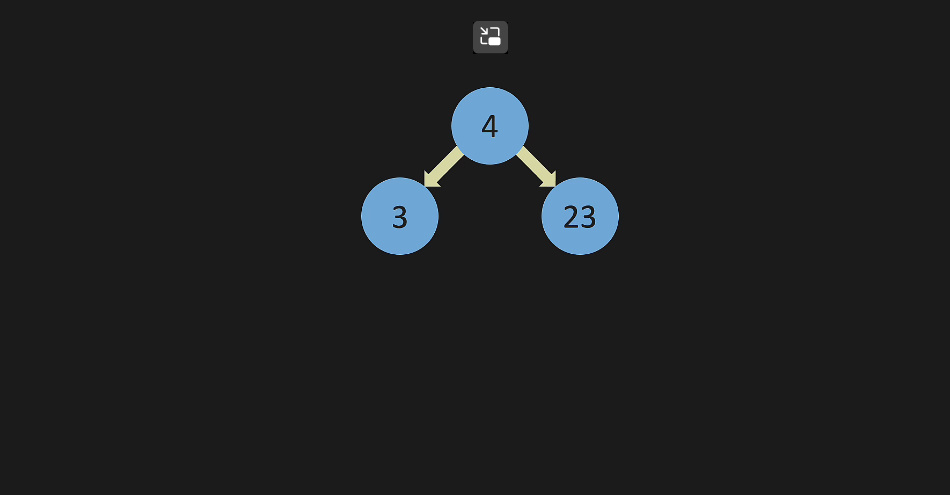
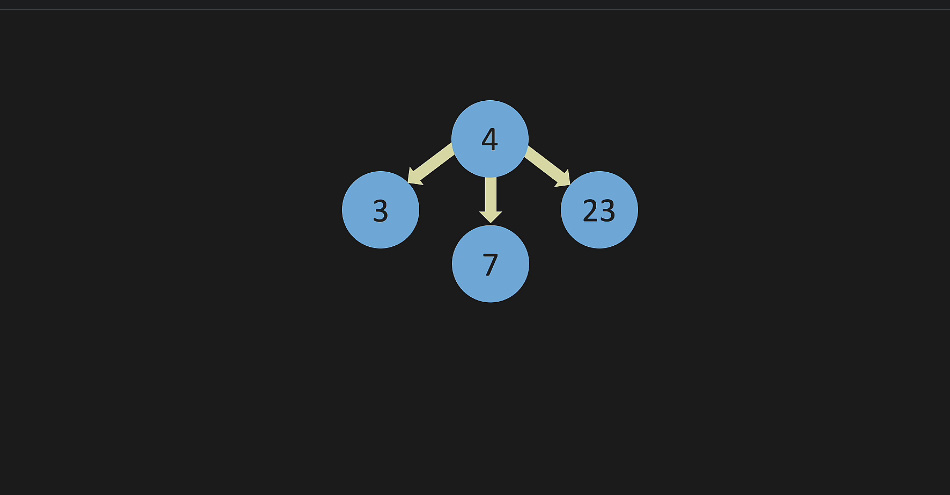
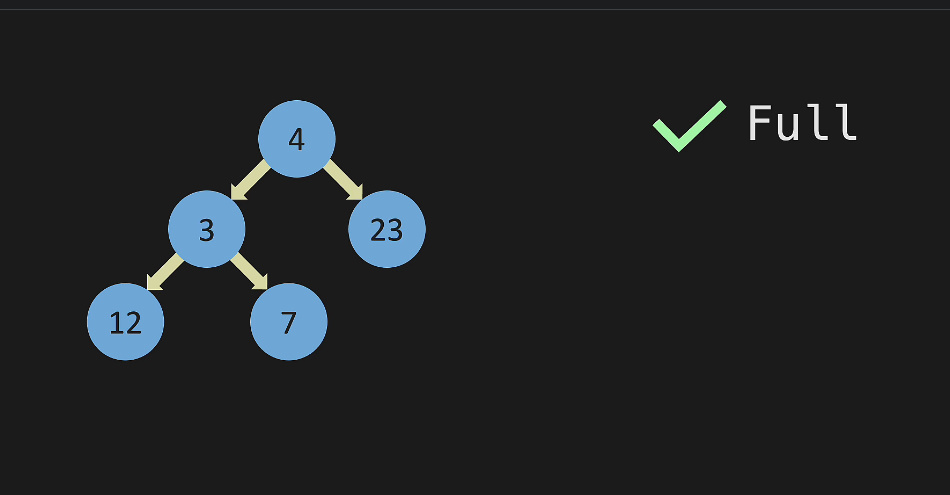
* LinkedList tot este un arbore
* **BinaryTree**: fiecare nod are referinta la inca 2 noduri:



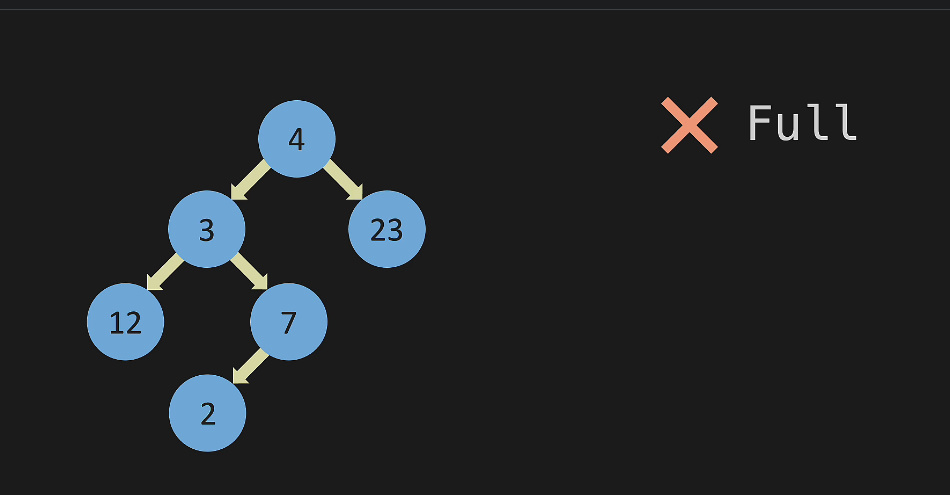
* Un Simple **Tree** are noduri ce pot avea oricate referinte la alte noduri.



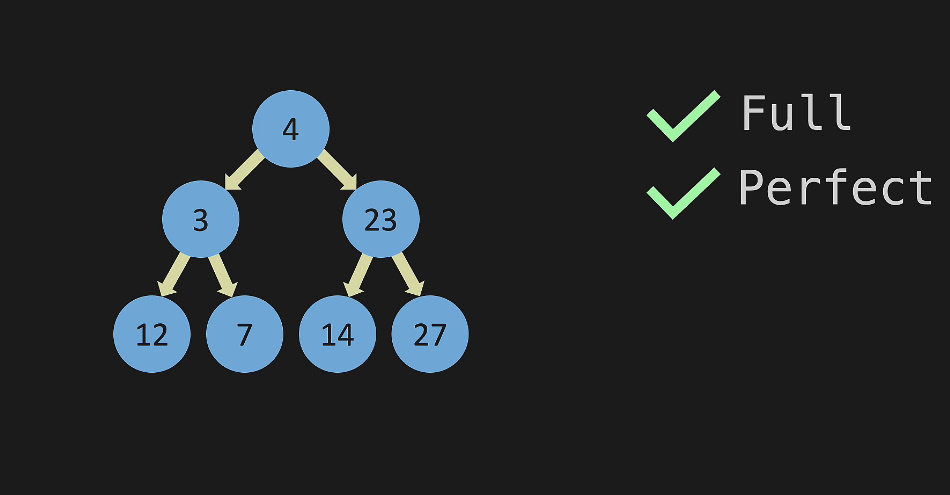
* Types of Tree:
* **Full**: fiecare nod fie pointeaza la 0 sau 2 noduri



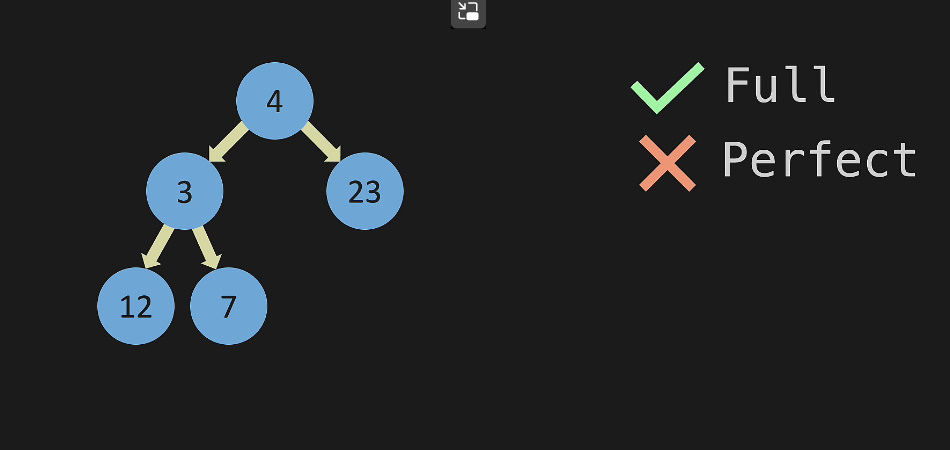
* **Non Full**: are noduri ce au si o referinta doar:



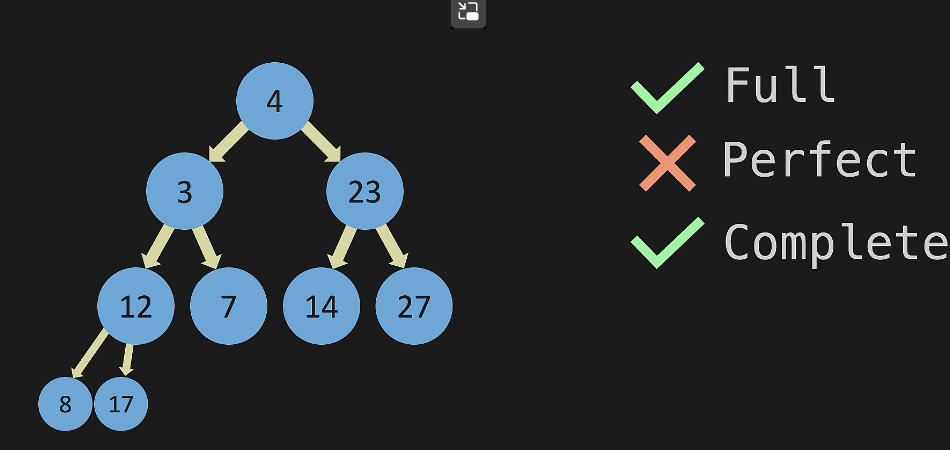
* **Perfect** – are ultima linie completa cu noduri



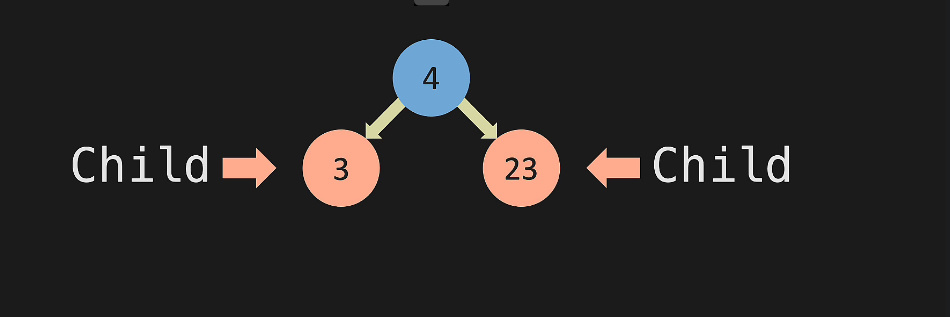
* Imperfect:



* **Complete** – Nodurile sunt adaugate de la stanga la dreapta:

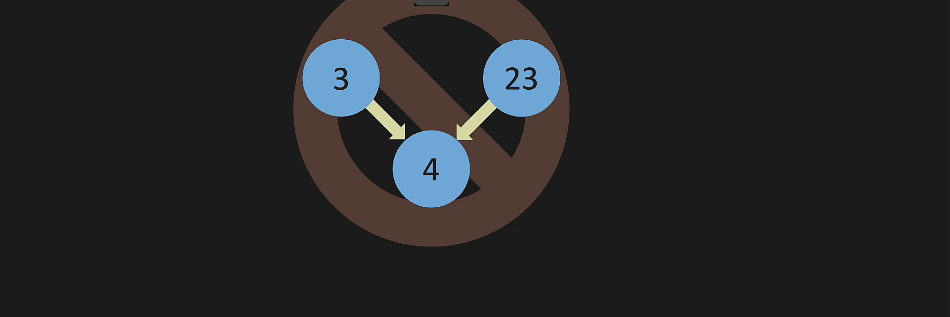


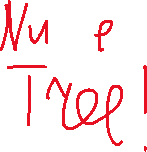


* 

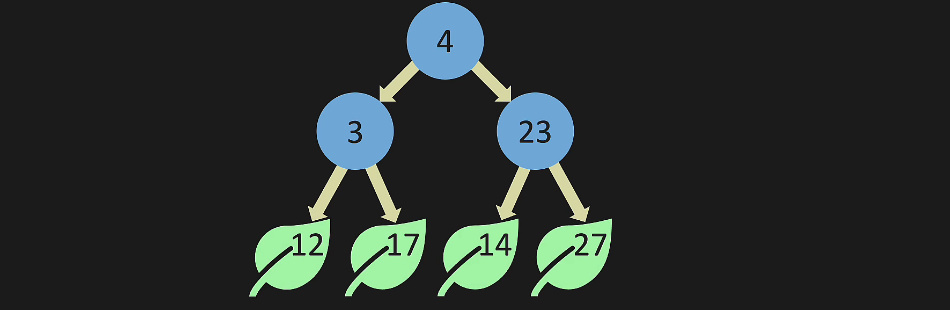


* 2 noduri copii adiacente se numesc si **gemeni(siblings)**
* Fiecare nod poate avea doar un parinte!



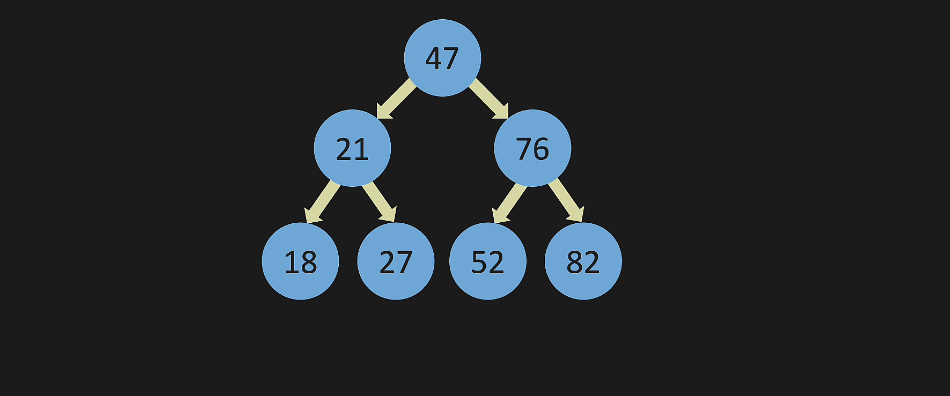


* **leaf** – nod fara copii



**BST**

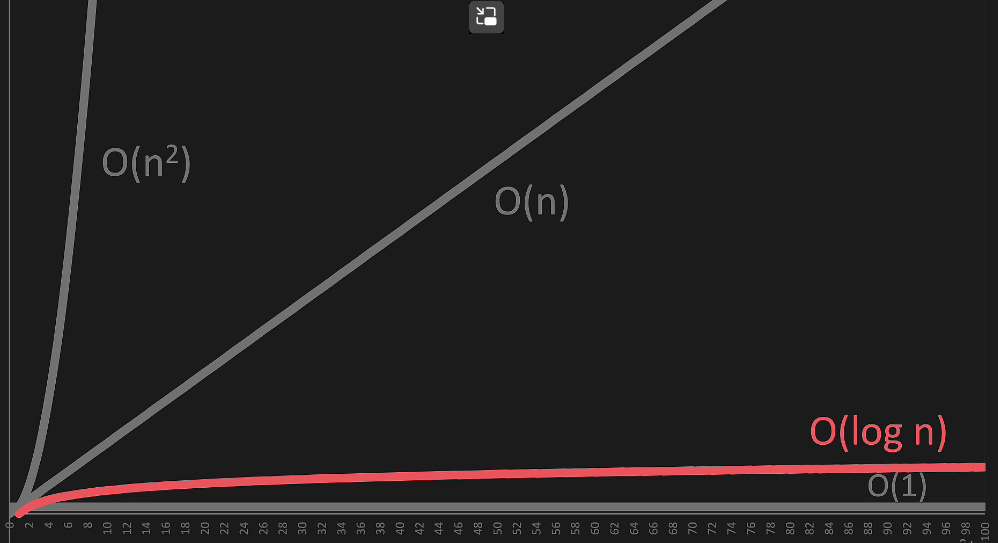
* **BST**: Binary Search Tree unde Nodul care are valoare mai mica, se duce in stanga la nodul parinte, cel cu valoare mai mare in dreapta:
* Plasarea unui nod mereu incepe de la nodul de sus, adica prim



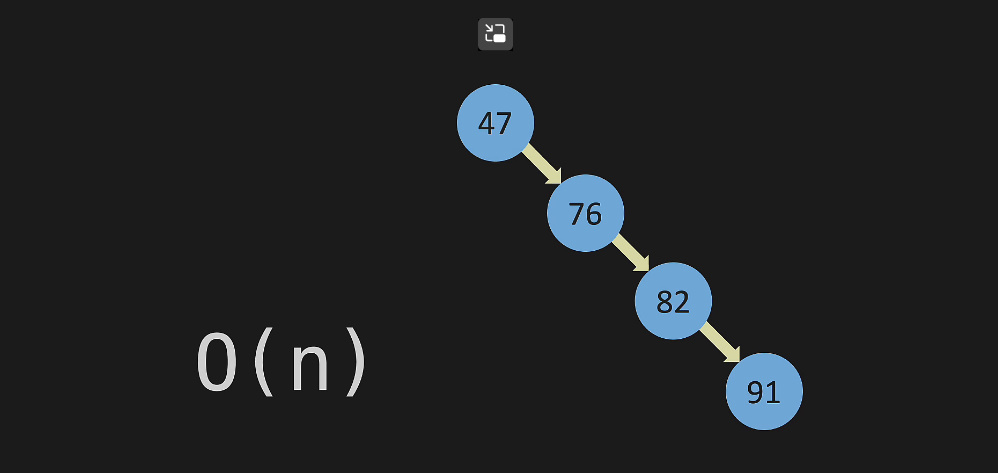
**BST Big O**

* Numarul de noduri: 2^h – 1 unde h este numarul de nivele sau adancimea arborelui

**BestCase:**

* search(value),remove(value),add(value) – O(logn)
* 

**WorstCase:**



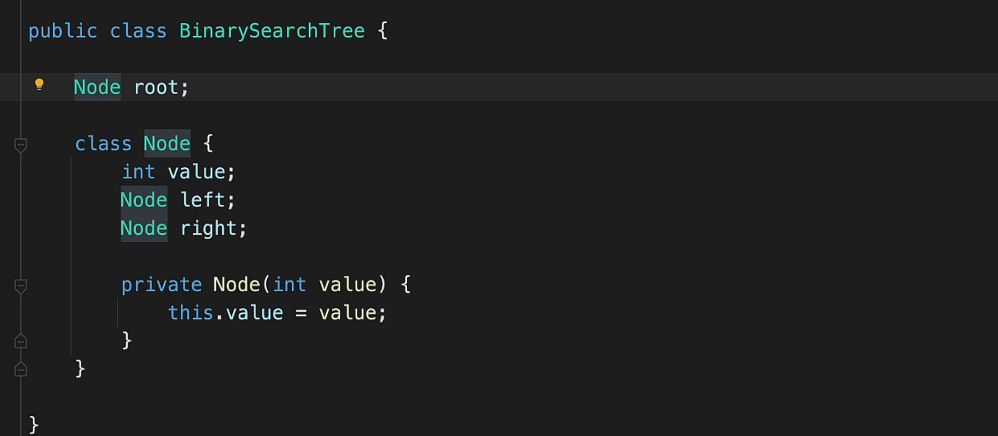
Deci Big O la un BST e totusi O(n), nu O(logn)

* Totusi, se considera ca un BST balansat e folosit in practica, nu o linie dreapta, si se considera ca BST are O(logN)
* 



**Methods**

* BST are neaparat un head(**root**), de altfel daca nu ar avea, totul ar fi sters de Garbage Selector si nici nu am avea cum sa iteram sau sa facem ceva cu el
* Un BST nu necesita length
* La un BST, primul nod se adauga cu metode, nu in timpul crearii



* insert(value) –

public boolean insert(int value){

Node newNode = new Node(value);

Node temp = root;

if(root == null){

root = newNode;

return true;

}

while(true){

if(newNode.value == temp.value)

return false;

if(newNode.value < temp.value){

if(temp.left==null){

temp.left = newNode;

return true;

}

else

temp = temp.left;

}

else if(newNode.value > temp.value){

if(temp.right == null){

temp.right = newNode;

}

else

temp = temp.right;

}

}

}

* contains(value)

