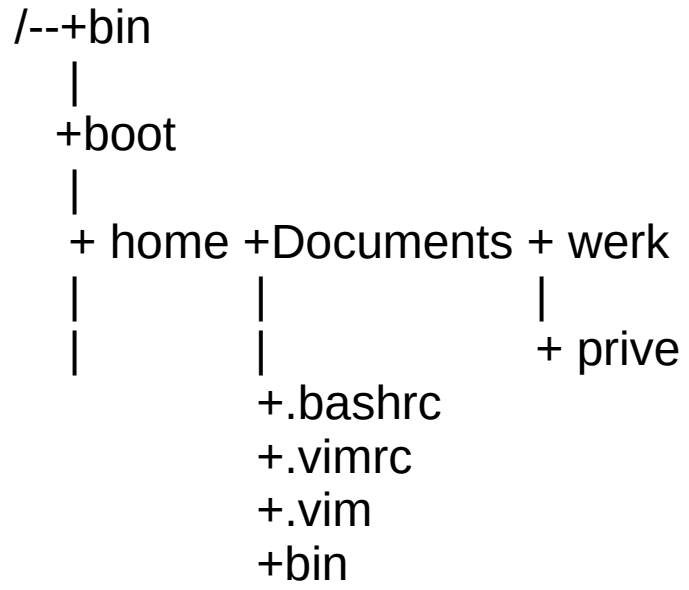
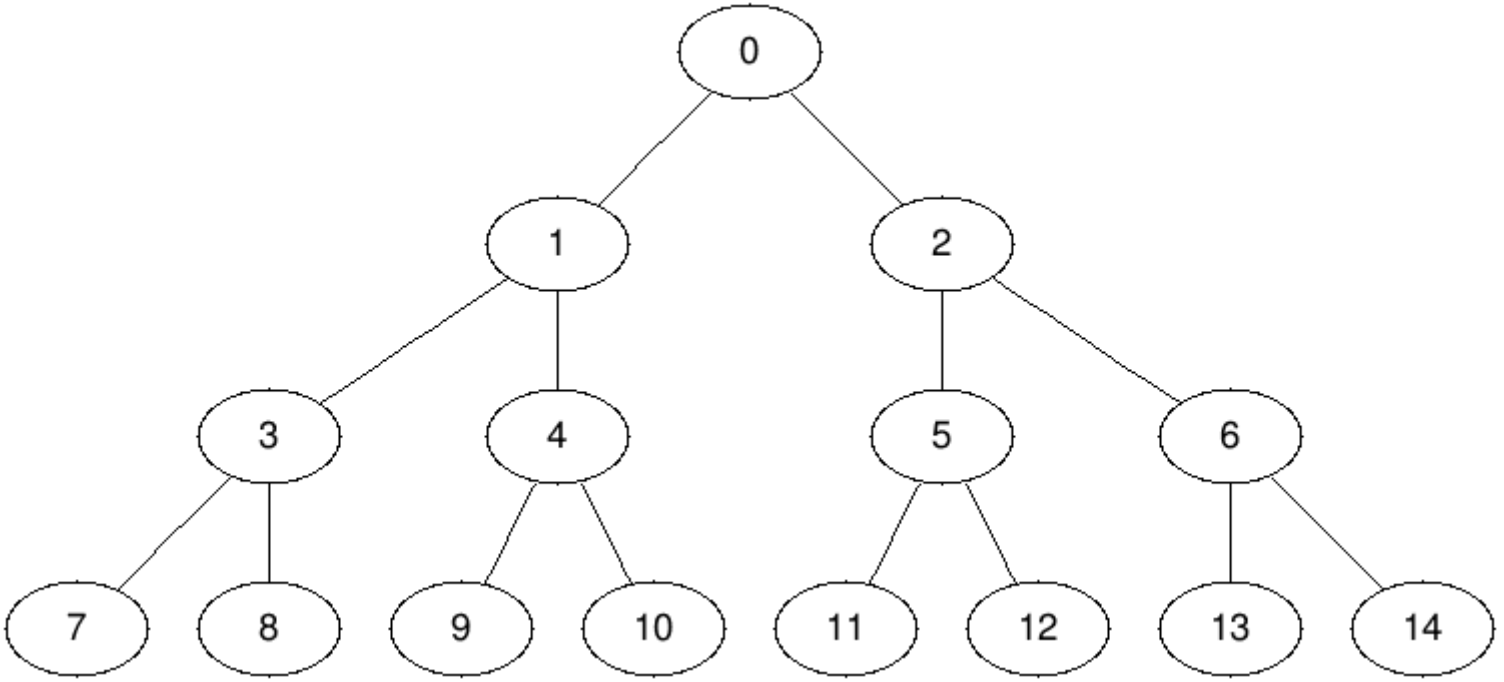

/--+bin
|
+boot
|
+ home +Documents + werk
| | |
| | + prive
+.bashrc
+.vimrc
+.vim
+bin

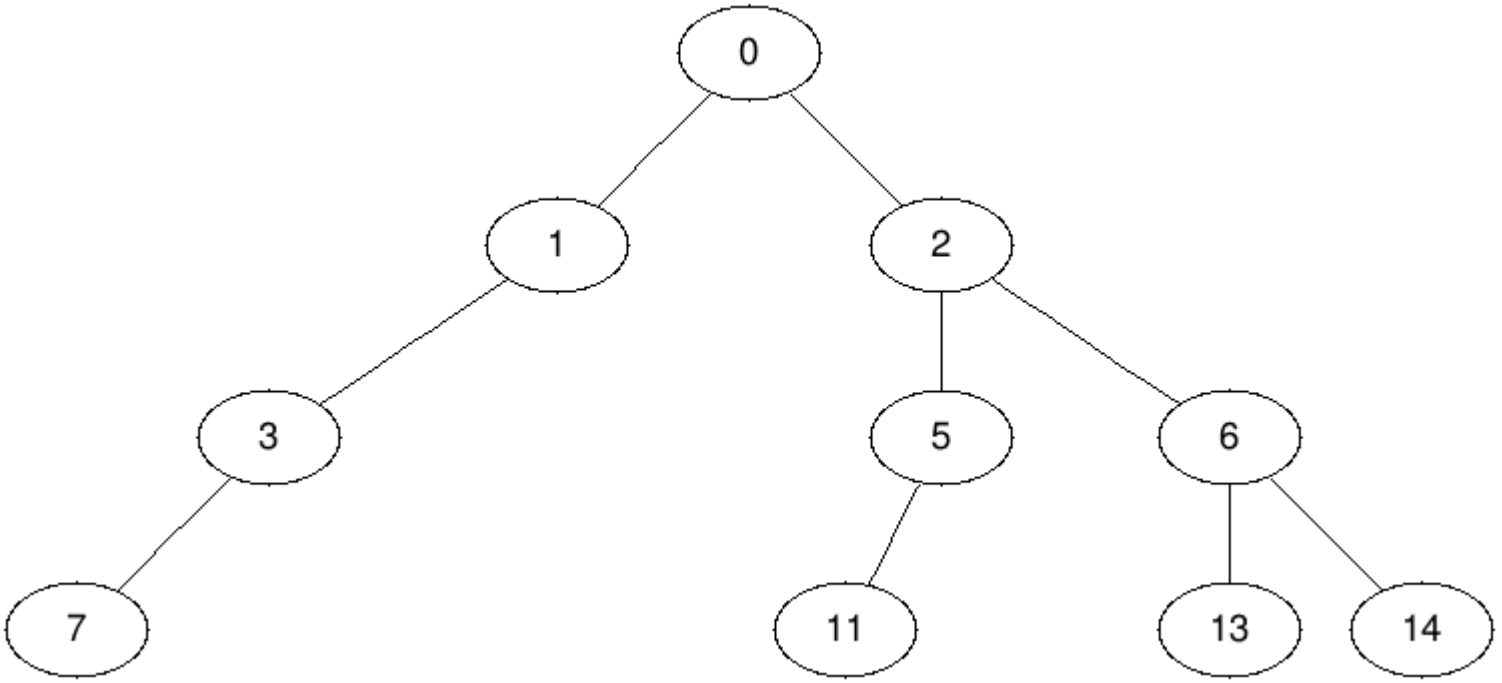
Bomen

- speciale vorm van een 'Graaf'
  - vertices (knopen) verbonden door 'randen' (edges)
- is hierarchisch georganiseerd
  - heeft een 'root' als begin
  - een node heeft maximaal 1 parent
  - een node heeft 0, 1, of meer children
  - nodes met dezelfde parent zijn siblings
  - (bet(achter(klein)))kinderen zijn descendants
  - (bet(over(groot)))ouders zijn ancestors
  - Geen descendants? => external (of leaf)
  - Descendants? => internal

Soorten

- geordend:
  - alle children staan op een bepaalde volgorde
- Ongeordend:
  - ...
- Binair:
  - Elke node heeft maximaal 2 children (links en rechts)
    - als alleen 0 of 2 kinderen: zuiver (proper) binair
  - De children zijn geordend


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

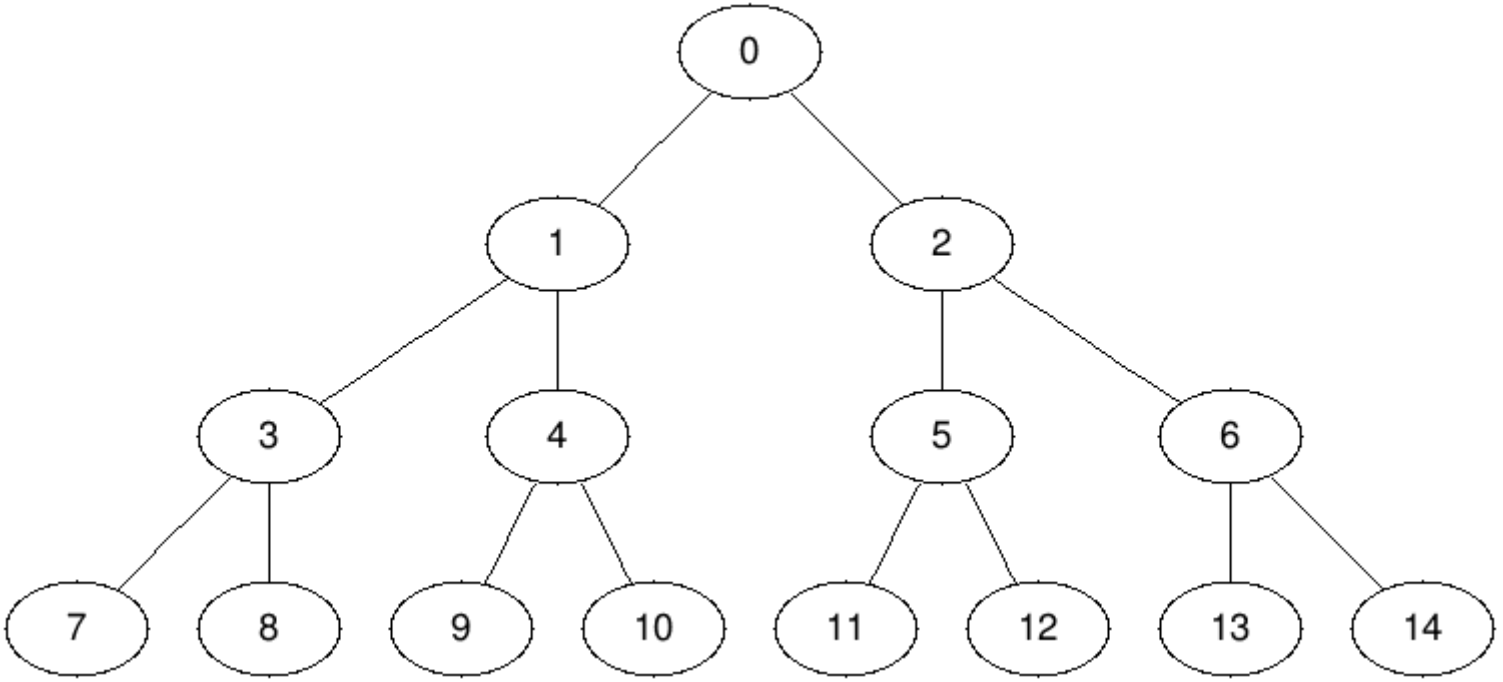

0
1
2
3
5
6
7
11
13
14

Hoogte, Diepte

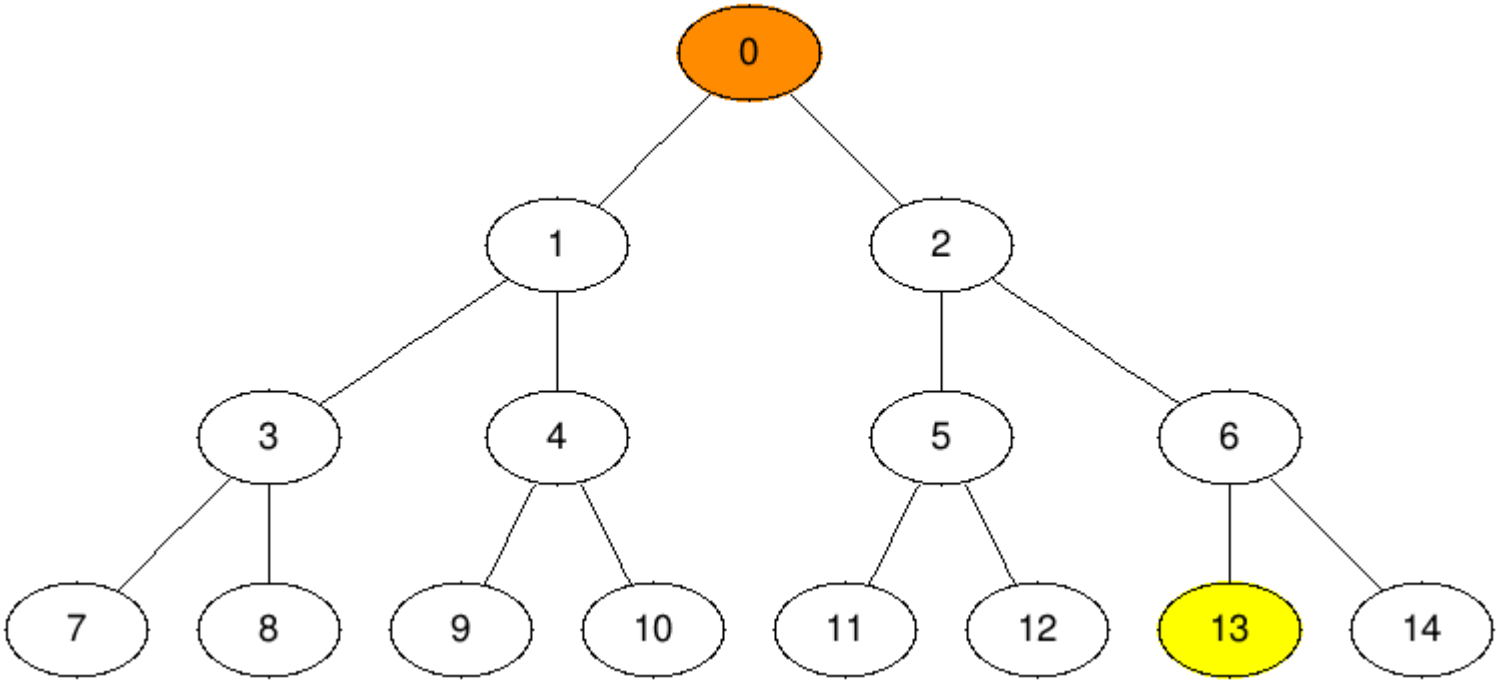
- Diepte:
  - het aantal ancestors van een node
- Hoogte:
  - 0, als blad
  - 1 + de maximale hoogte van de kinderen

Hoogte boom: log2(n)

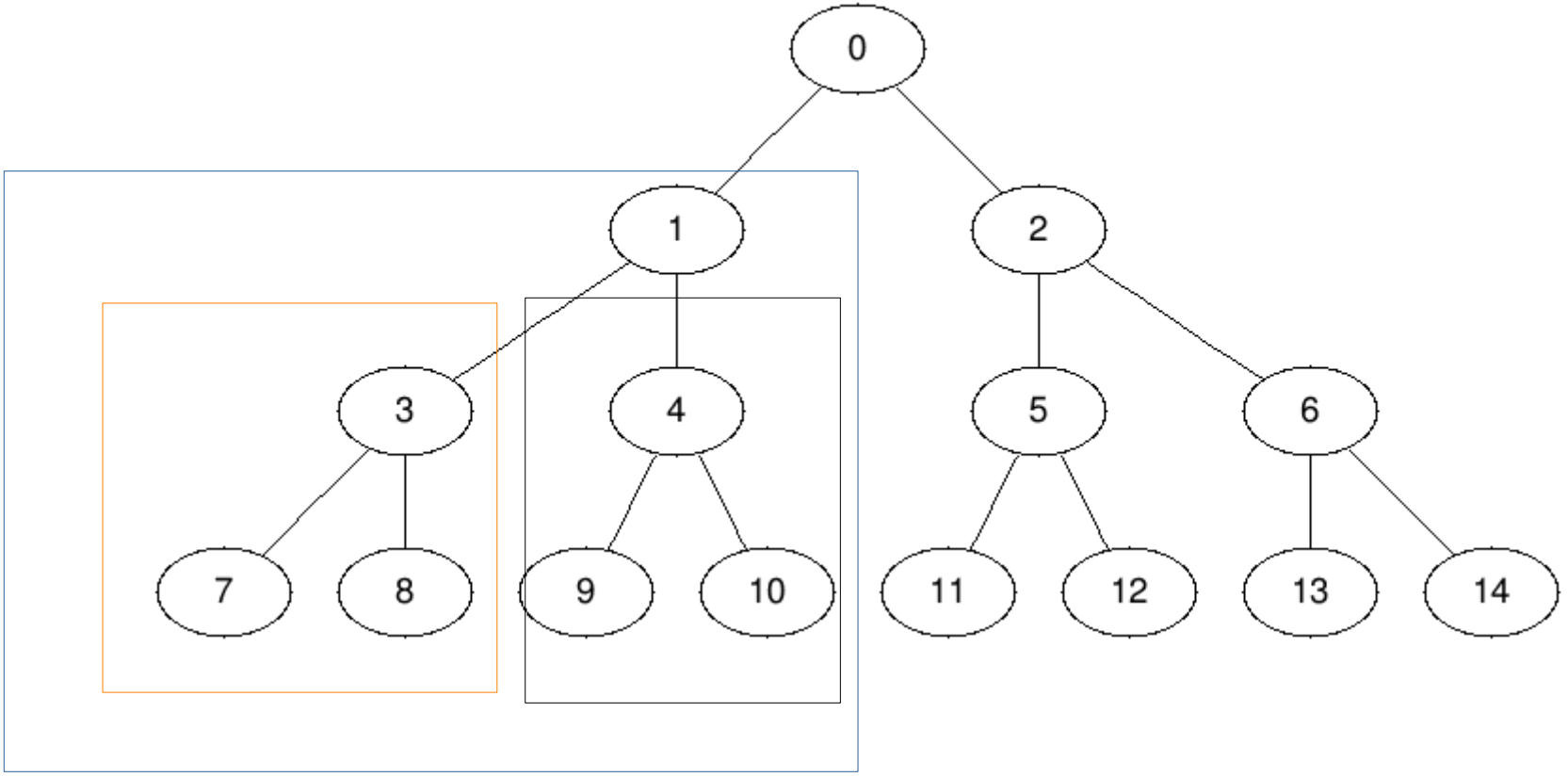
Op bovenste laag (laag 0): 2^0 = 1 nodes
Op 2e laag (laag 1): 2^1 = 2 nodes
Op 3e laag (laag 2): 2^2 = 4 nodes
...



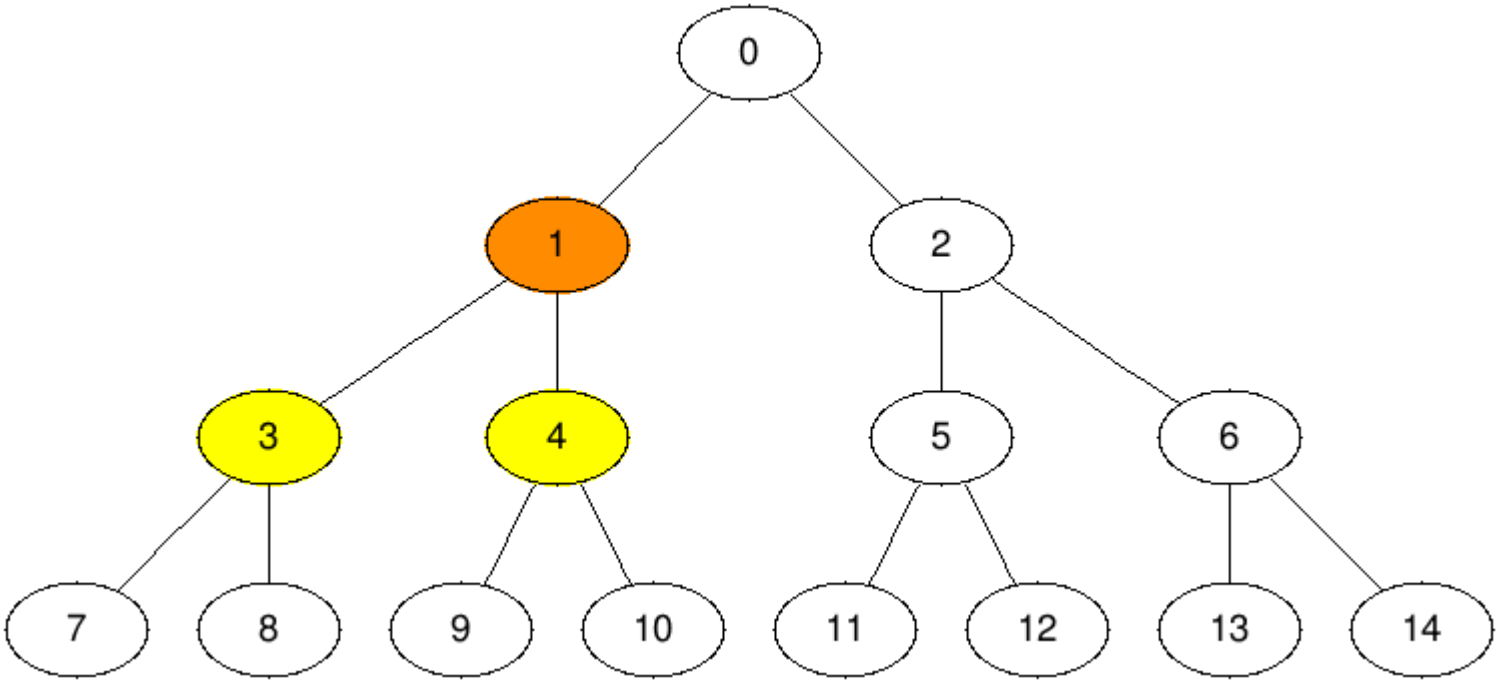
Root node en leaf node



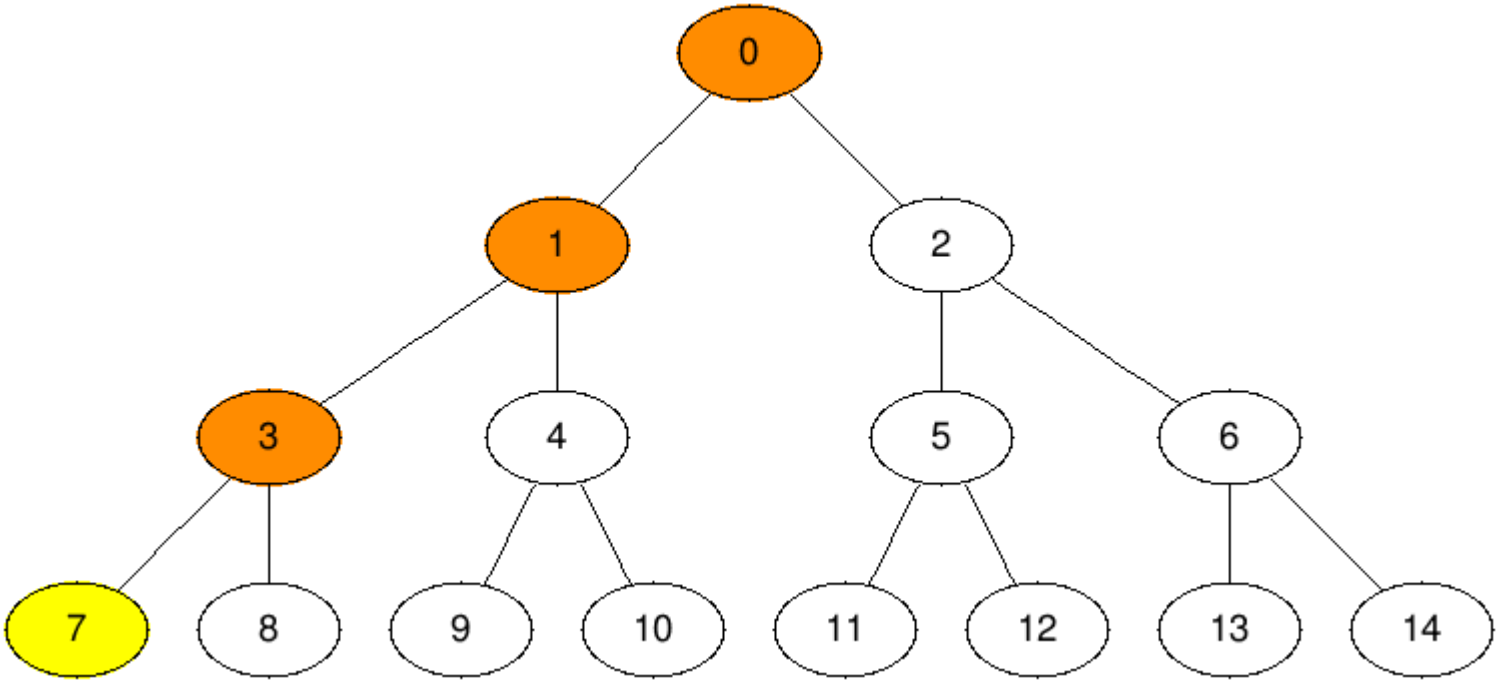
Een Tree is recursief



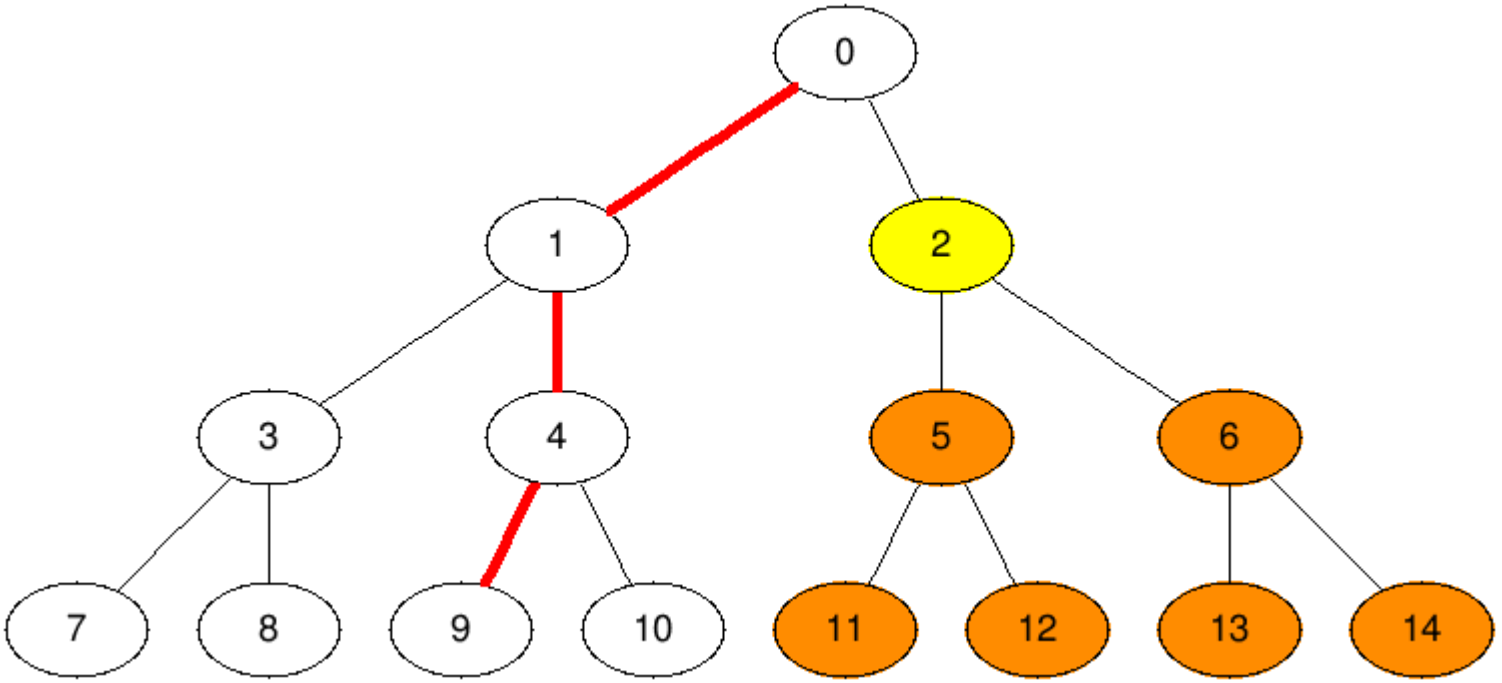
parent/children/siblings



Ancestors van 7

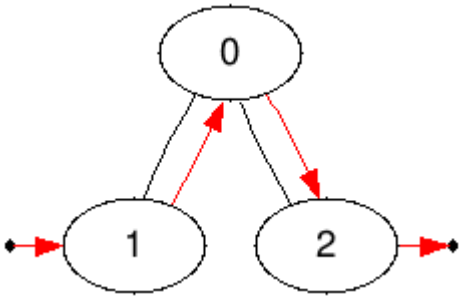


Een pad en de descendants van 2

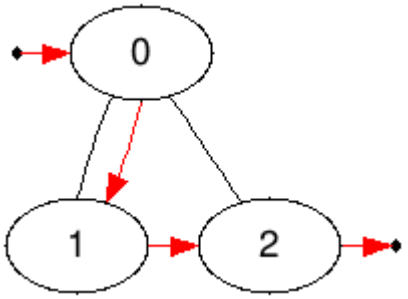


traverses

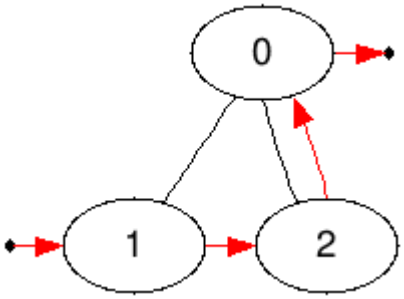
- Pre-orderer traversal
  - alle nodes
  - eerst de parent, dan de kinderen van links naar rechts
- Post-order traversal
  - eerst de children van links naar rechts
  - dan de parent
- In-order (binaire boom)
  - linker child, parent, rechter child



In-order traverse
Left child → parent → right child



pre-order traverse
parent → left child → right child

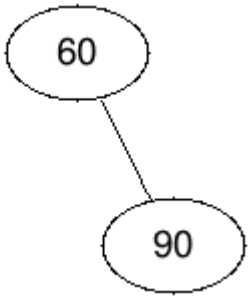


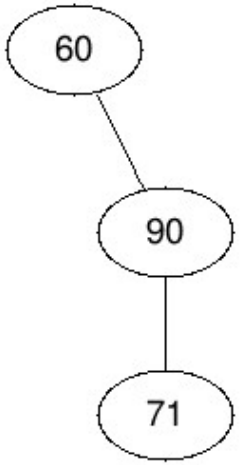
post-order traverse
Left child → right child → parent

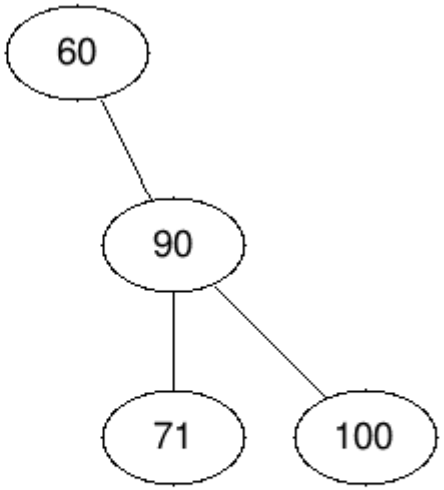
Binary search tree

Zoek eerste lege positie
    als waarde node > toe te voegen waarde naar rechts
    Anders naar links

Vb: [60, 90, 71, 100, 12, 71, 4, 5, 1, 69, 41, 84]





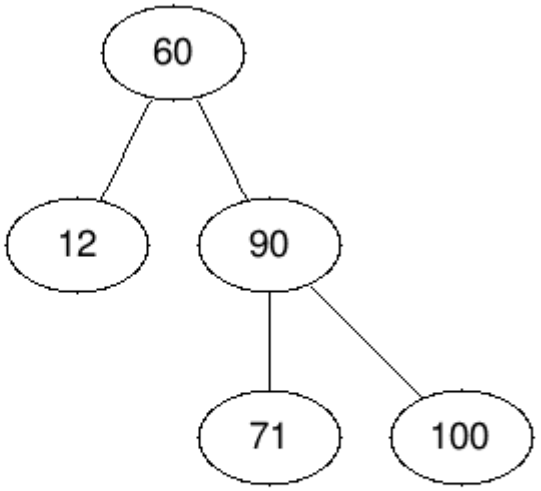


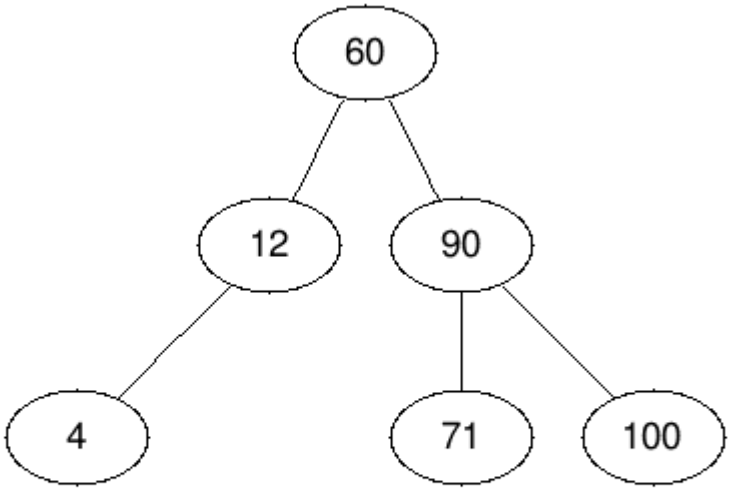
Zoek eerste lege positie

als waarde node > toe te voegen waarde naar rechts

Anders naar links

Vb: [60, 90, 71, 100, 12, 71, 4, 5, 1, 69, 41, 84]



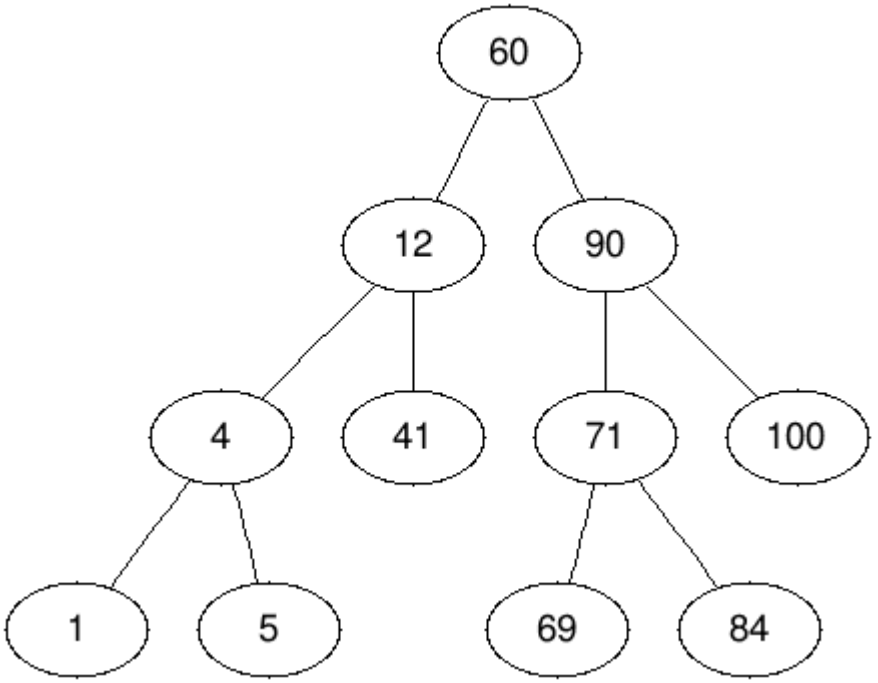


Zoek eerste lege positie

als waarde node > toe te voegen waarde naar rechts

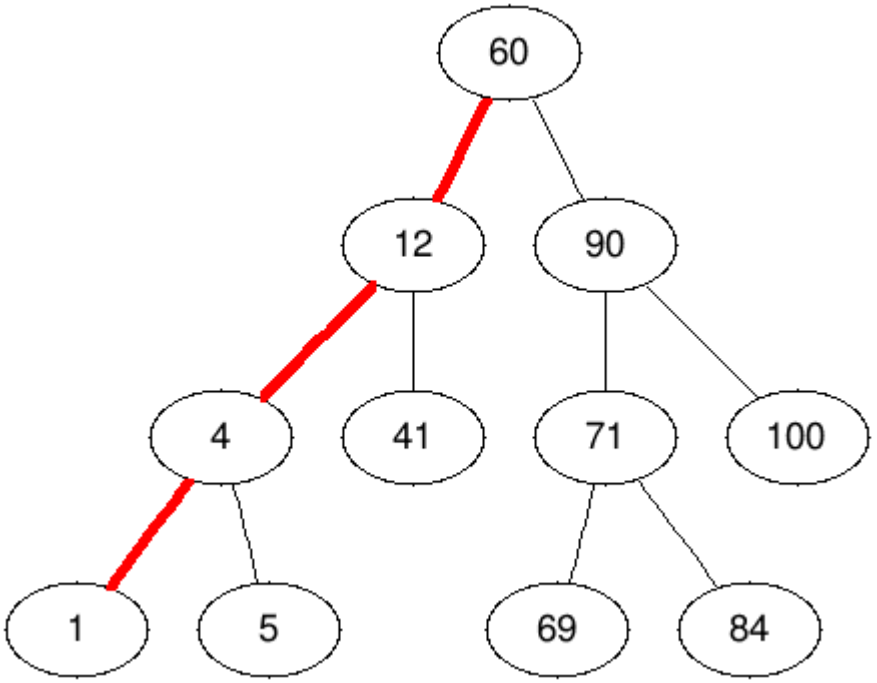
Anders naar links

Vb: [60, 90, 71, 100, 12, 71, 4, 5, 1, 69, 41, 84]

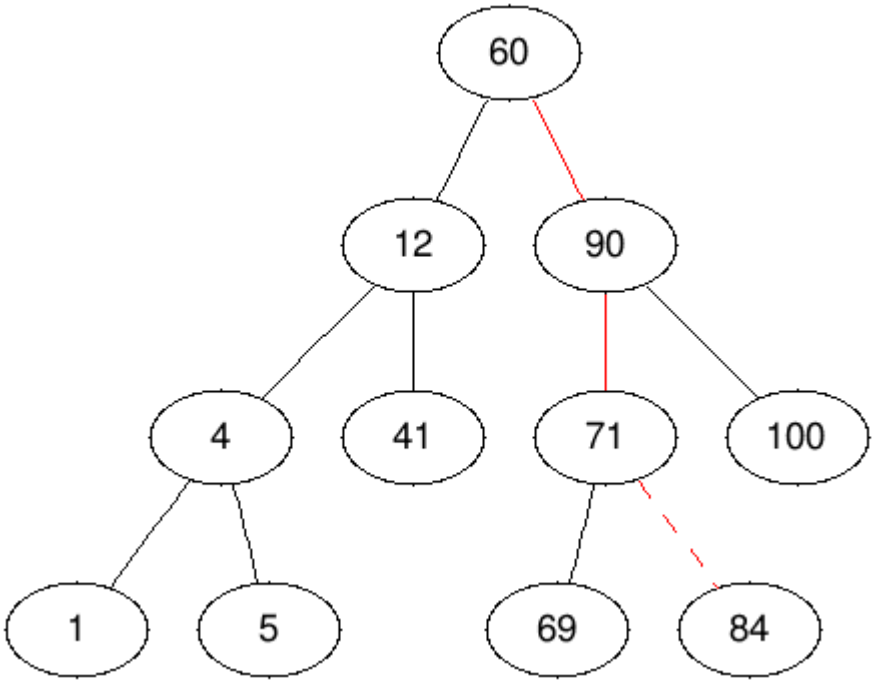


zoeken

Zit 1 in de tree?
Volgens dezelfde regels

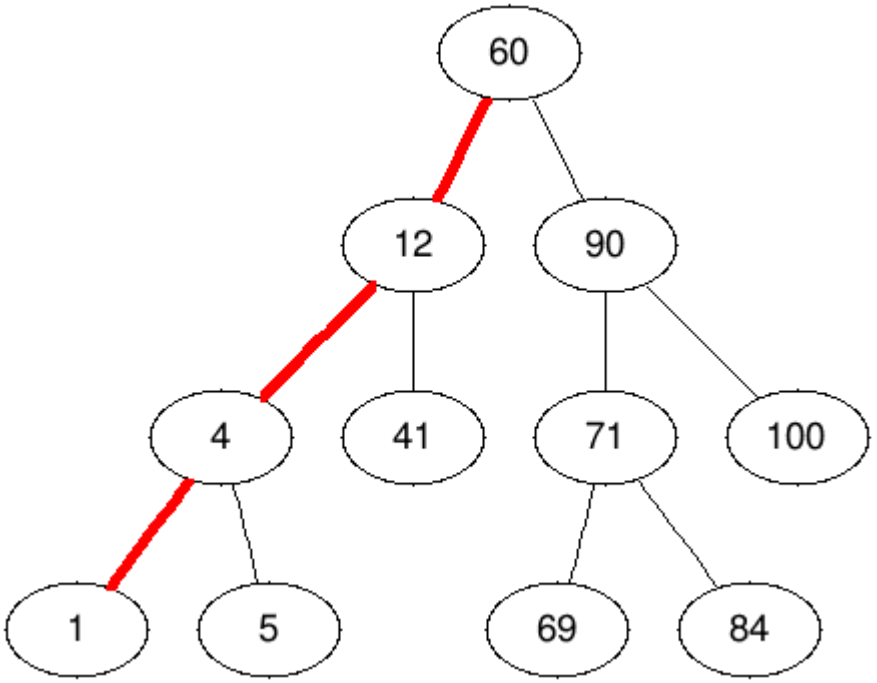


...en 72?



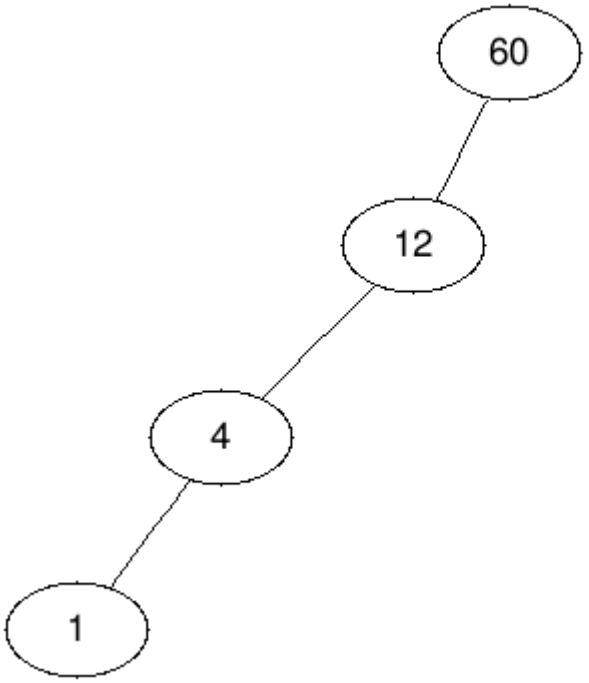
zoeken

Tijdscomplexiteit wordt bepaald door de hoogte van de boom
O(log2(n))

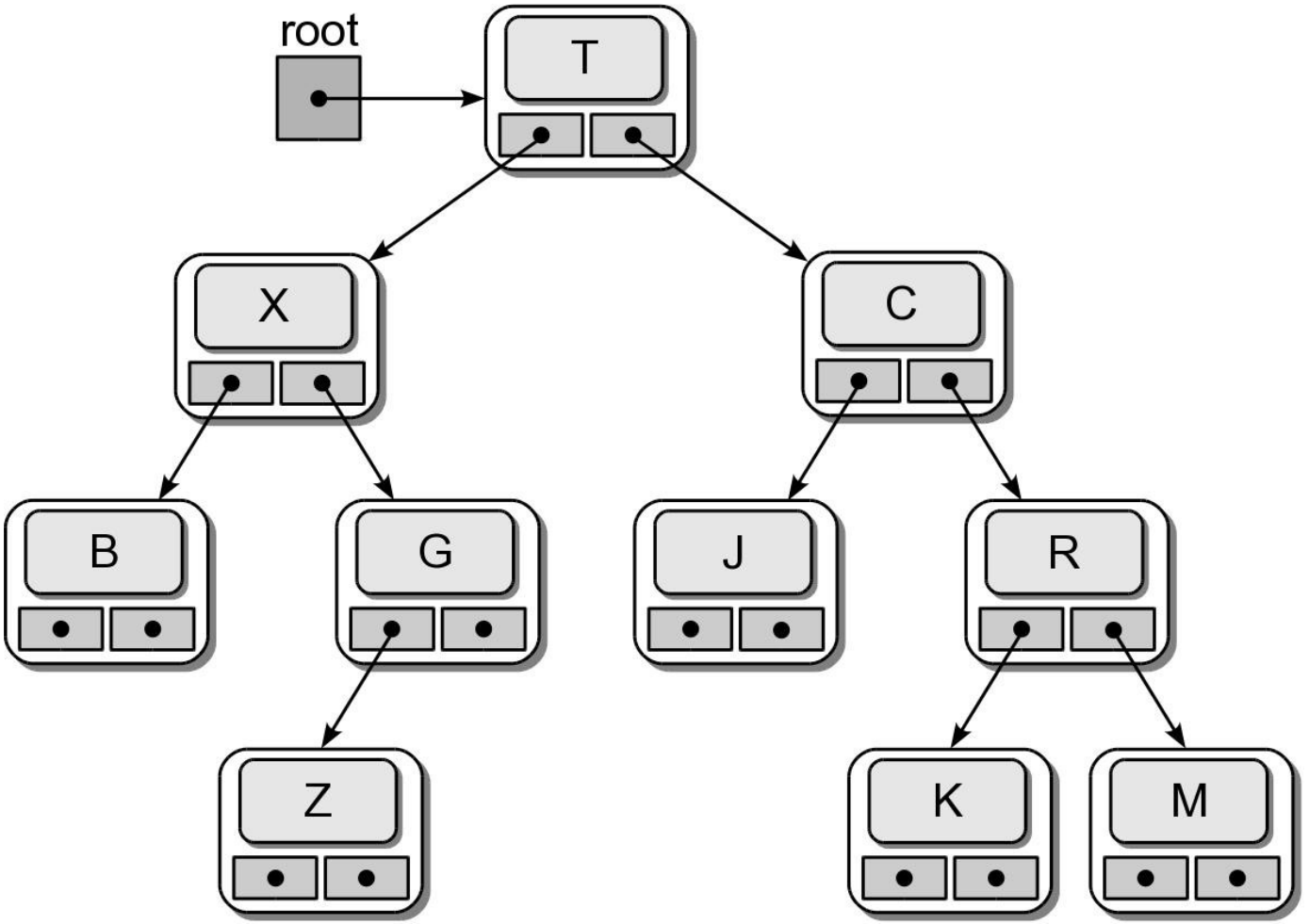


Maak een binary search tree van de volgende array [60, 12, 4, 1]
Zit 0 hier in?
Wat is de tijdscomplexiteit om dit te bepalen

0 zit er niet in
Dit loopt in O(n)!



- Probeer bomen ‘in evenwicht’ te houden
- Verschillende manieren voor


root
T
X
C
B
G
J
R
Z
K
M

Hoogte boom: log2(n)

Op bovenste laag (laag 0): 2^0 = 1 nodes
Op 2e laag (laag 1): 2^1 = 2 nodes
Op 3e laag (laag 2): 2^2 = 4 nodes
...

0 | 1 | 2 | 3 | ...