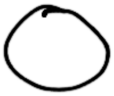
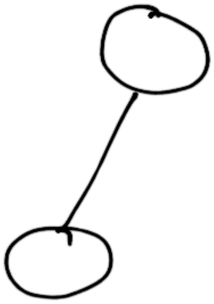
Binary Tree & its types

- Each node can have 'atmost' two children

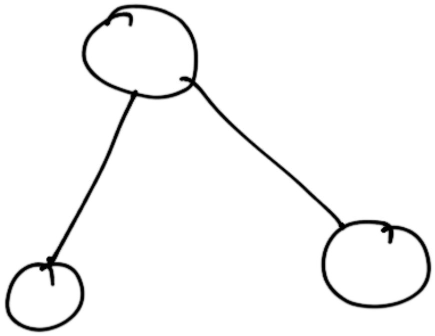
Children - 0, 1, 2



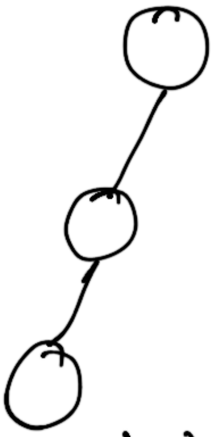
(i) ___



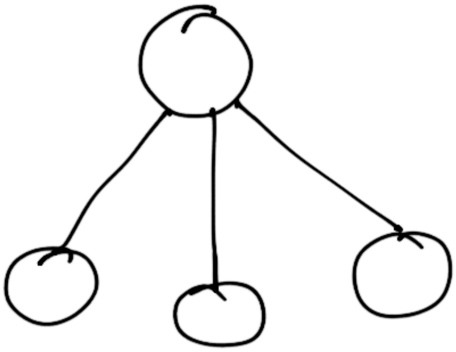
(ii) ___



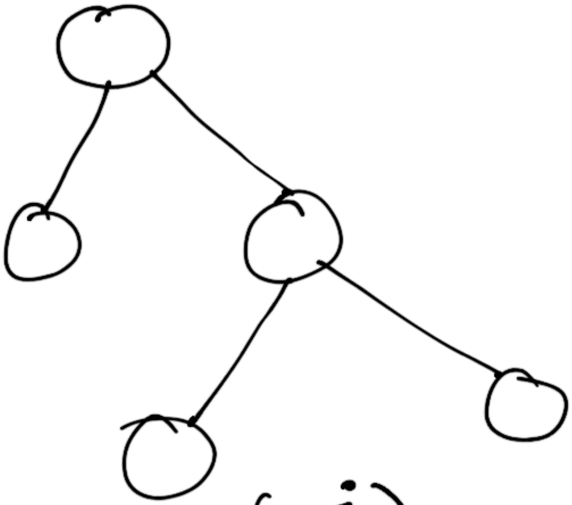
(iii) ___



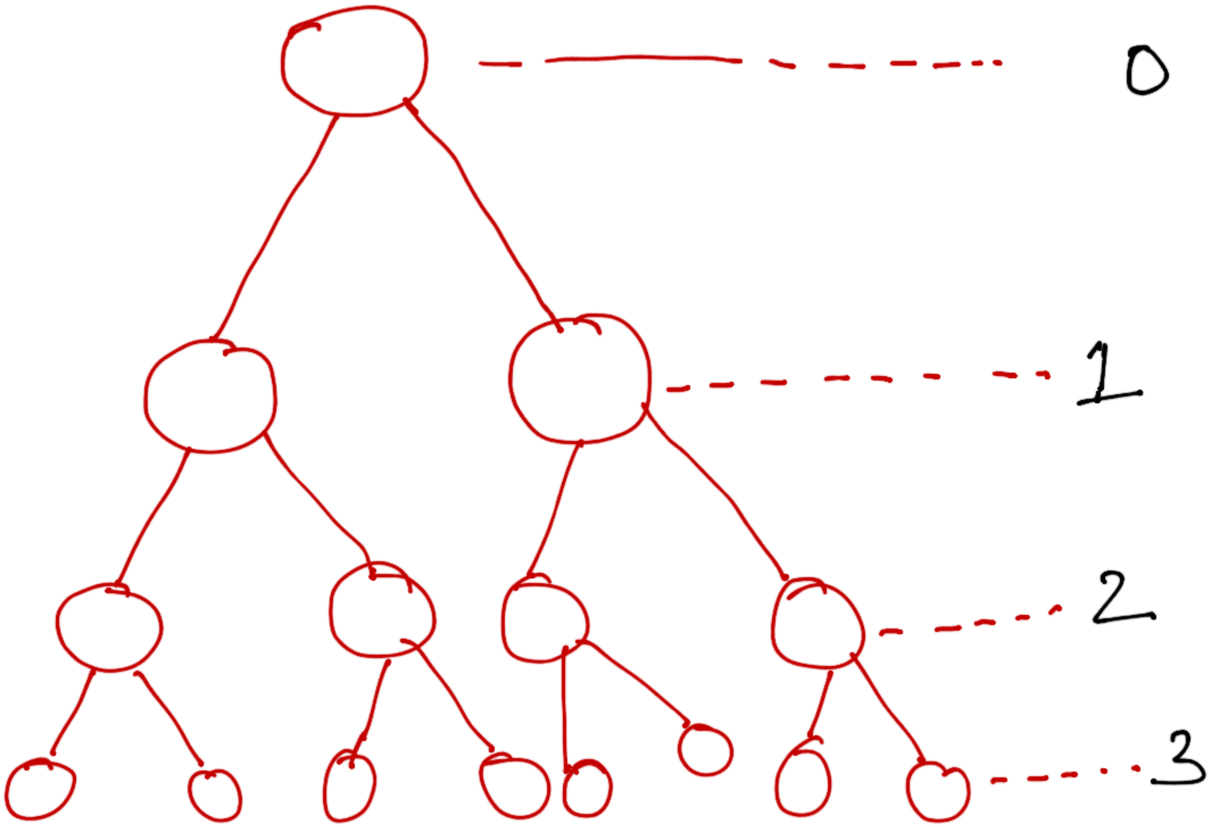
(iv) ___



(v) ___



(vi) ___



Level	Max. No. of Nodes
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$

Max. No. of nodes at any level 'i' $= 2^i$

Height of tree = Height of Root Node

= No. of edges in longest path from

Root to the leaf node

For this tree, height is 3

Max. no. of nodes in tree of height '3'

$= 2^0 + 2^1 + 2^2 + 2^3$

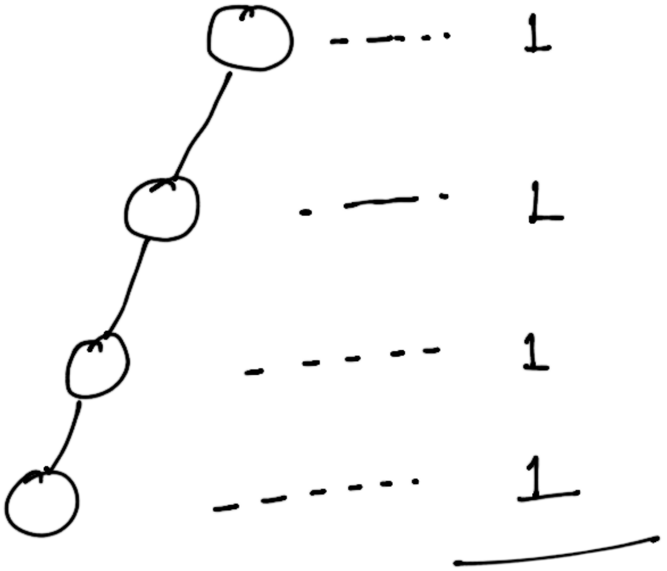
$= 1 + 2 + 4 + 8 = 15 \left(2^{h+1} - 1\right)$

Max. no. of nodes in tree of height 'h'

$$= 2^0 + 2^1 + 2^2 + \ldots + 2^h$$

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

Min. no. of nodes in tree of ht. 'h' $= h+1$



$4 \, (h+1)$

Max. ht. = when min. no. of nodes

$= n - 1$

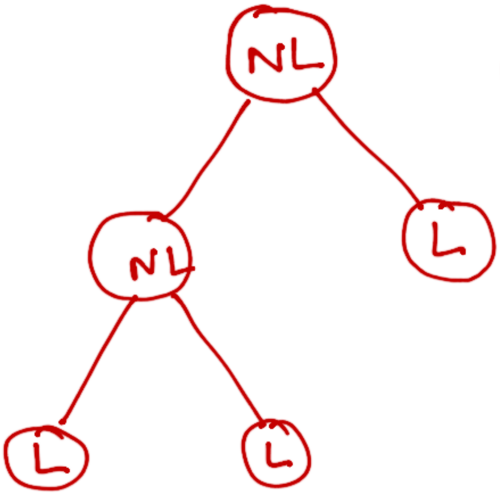
Min. ht = when max. no. of nodes

$= \lceil \log_2 (n+1) - 1 \rceil$

Types of Binary Tree

→ Full / Proper / Strict

- All nodes have 2 children, except the leaf nodes
- No. of leaf nodes = No. of non-leaf nodes + 1



No. of non-leaf nodes = 2

Leaf nodes = 2 + 1 = 3

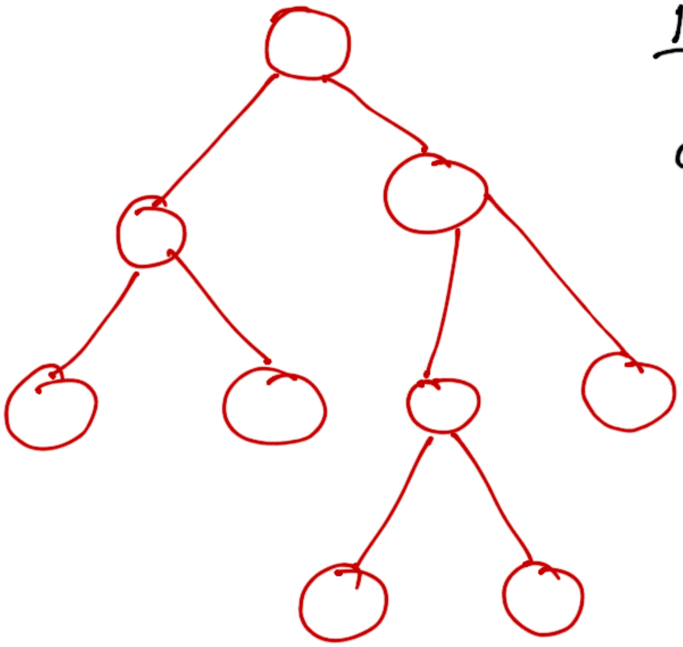
Max. no. of nodes = $2^{h+1} - 1$

Min no of nodes = $2h + 1$

→ Min ht = $\lceil \log_2(n+1) - 1 \rceil$

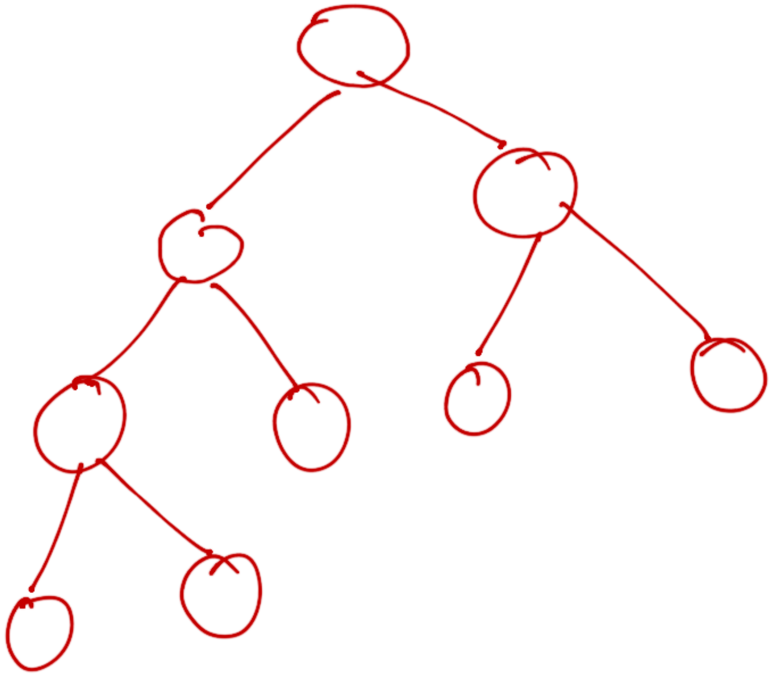
Max ht = $(n-1)/2$

Complete Binary Tree



1) All levels completely filled except the last

2) In the last level, node must be as left as possible



Max nodes $= 2^{h+1}-1$

Min nodes $= 2^h$

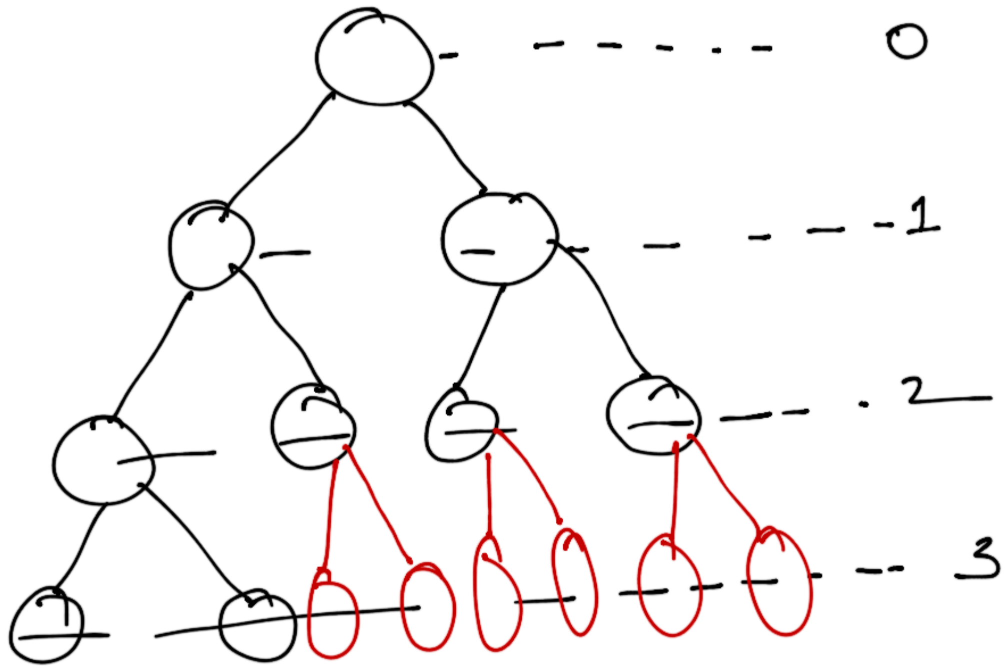
Max ht $= \lceil \log_2(n+1)-1 \rceil$

Min ht $= \log_2 n$

Perfect Binary Tree

→ All internal nodes have 2 childuen

→ All leaves are at the same level

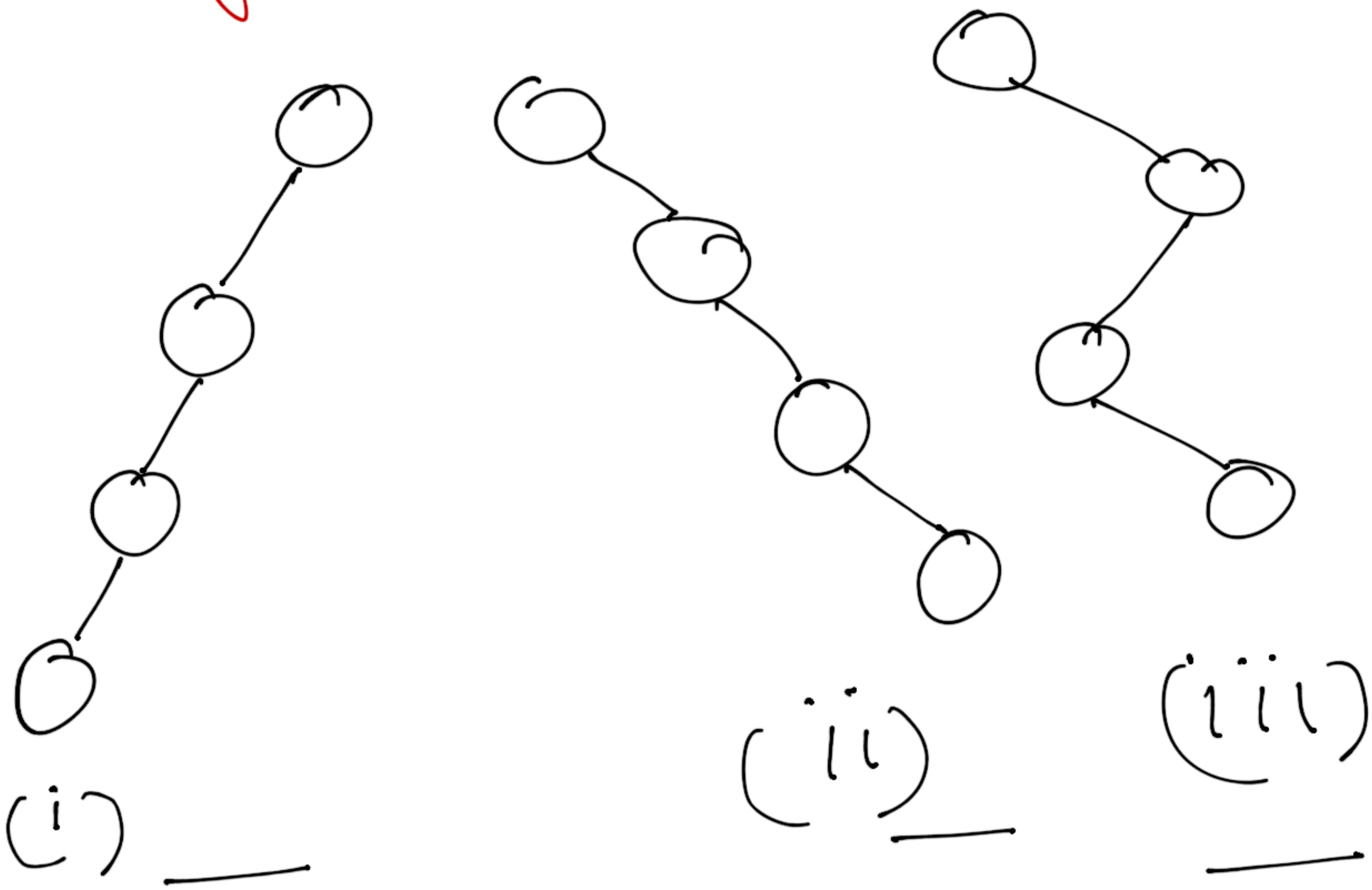


Is this CBT = ______

Is this FBT = ______

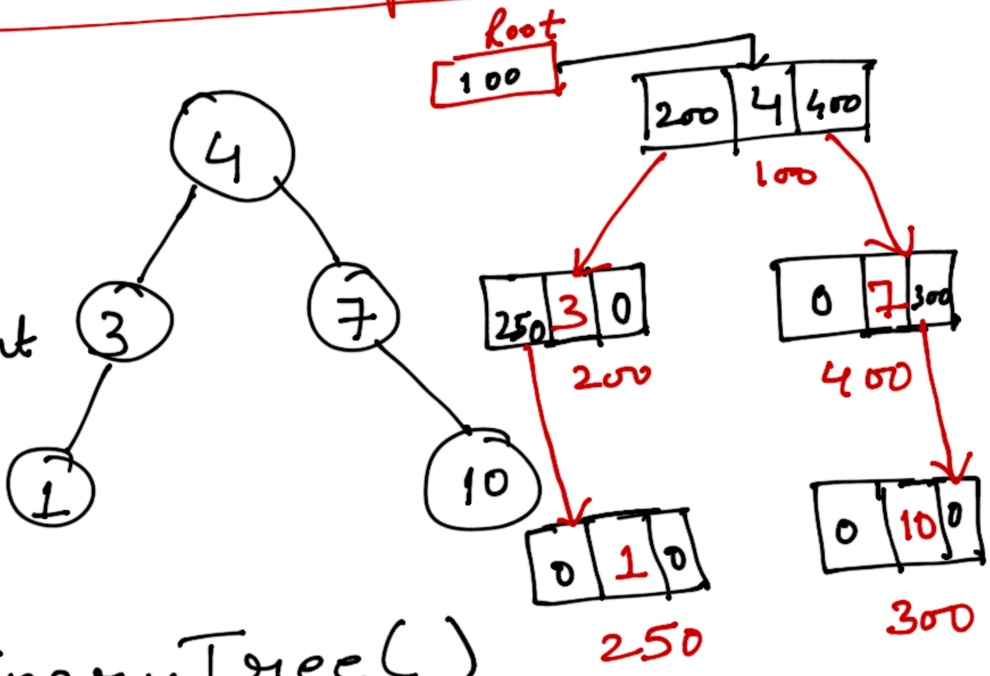
Degenerate Binary Tree

→ All internal nodes have only 1 child

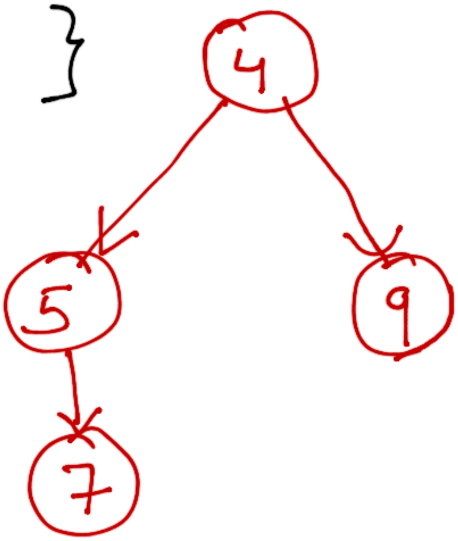


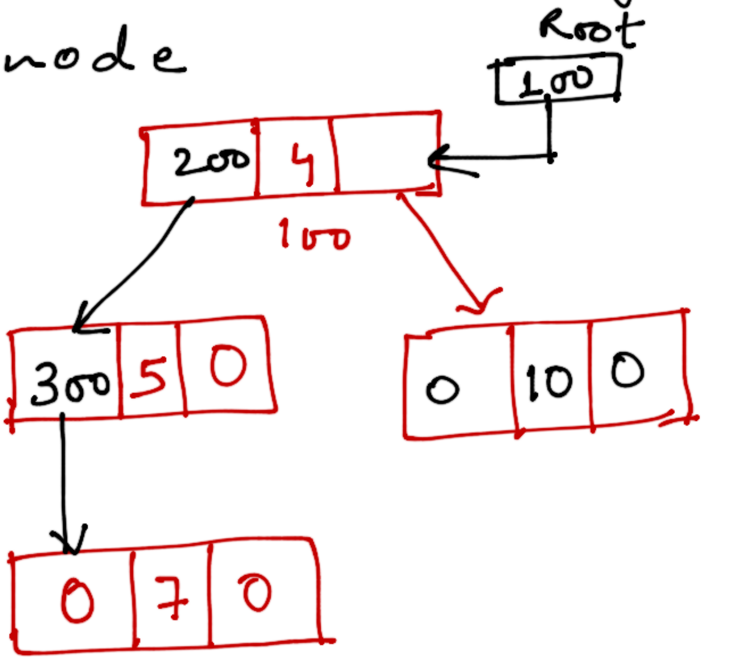
Binary Tree Implementation

```
Node
{ int data
  Node Left,Right
}
```

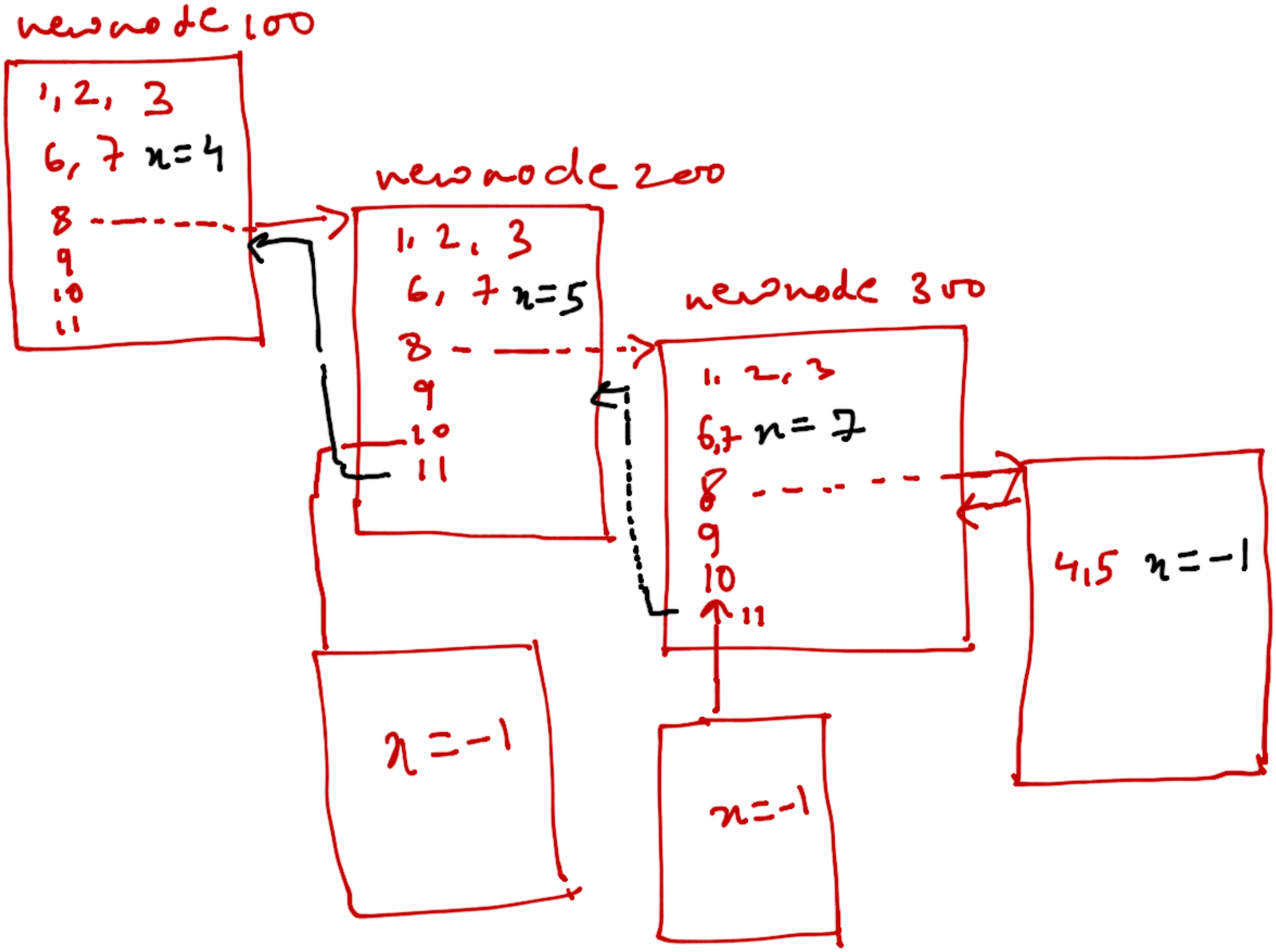


```
Node createBinaryTree( )
  {
1   newnode = Create a newnode
2    Print("Enter data")
3     x = Value to insert in node
4     if (x==-1)
5       { return 0; }
6      newnode→ data = x
7       Print("Enter left child")
8      newnode → left = createBinaryTree()
9       Print("Enter right child")
10  newnode → Right = createBinaryTree
11  return newnode
}
```





Root
100
200
4
100
300
5
0
0
10
0
0
7
0



newnode 100
1, 2, 3
6, 7 n=4
8
9
10
11
newnode 200
1, 2, 3
6, 7 n=5
8
9
10
11
newnode 300
1, 2, 3
6,7 n= 7
8
9
10
11
4,5 n=-1
n=-1
n=-1