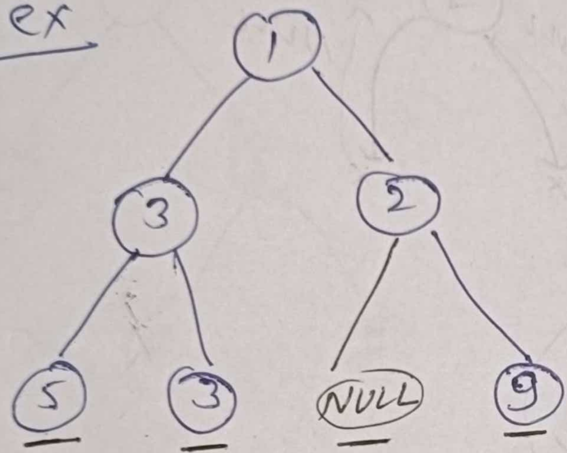
Maximum Width of Binary Tree

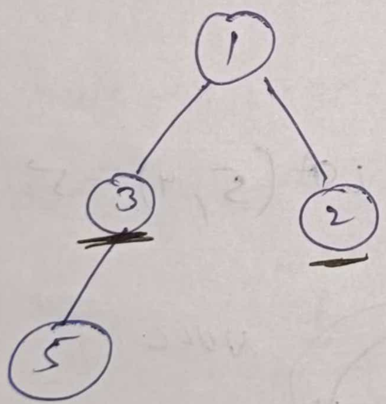
Width :- Maximum no. of nodes in any level b/w two nodes

ex



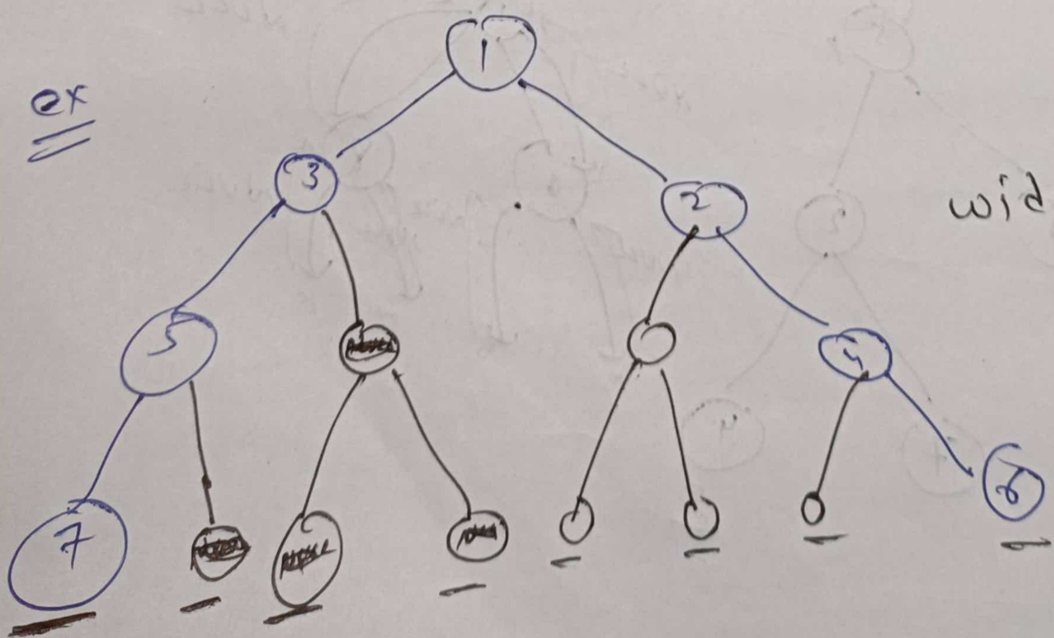
width = 4

ex



width = 2

ex



width = 8

Intuition :-

As we know that to solve BT problem we have to travers the tree & we go for traversal.

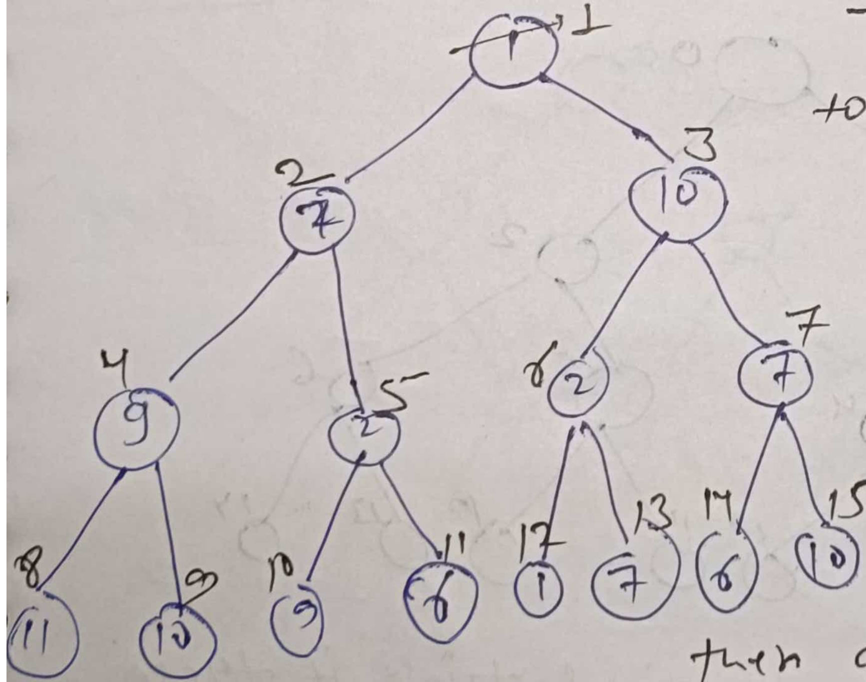
Types of traversa

BFS (level order)

DFS (inorder, preorder, postorder)

width is dependent on level (no. of nodes b/w 1st node & last node of the level.)

So we use level order traversal.

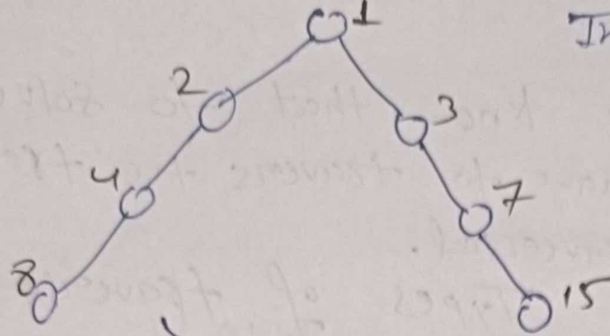


This is tree was given to yous where every node have different valuse. If we indexed the tree in such a way

Now if tree is indexed like this then question become very evey. Go to every node pickup the ~~first~~

$\boxed{|\text{first\_node index} - \text{last\_node\_idx}| + 1}$

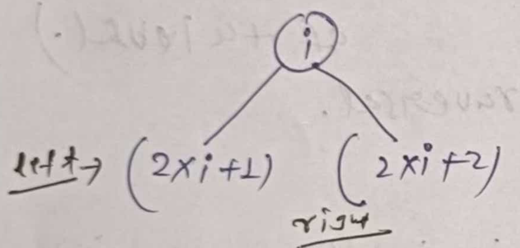
for every level the maximum of this is width.



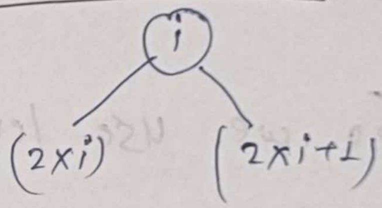
In these type of BT we are going to indexed like

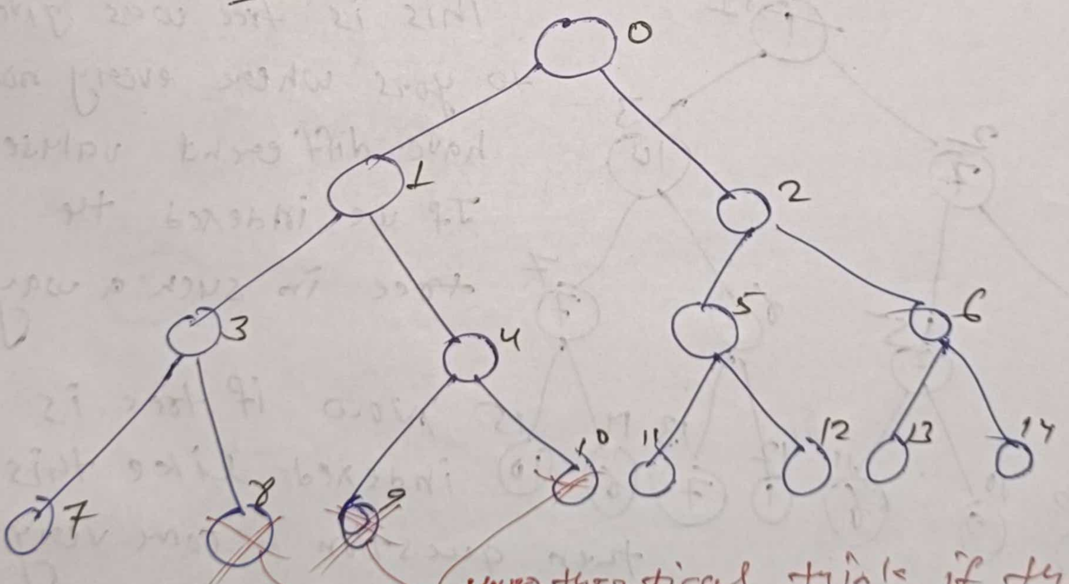
→ Hypothetical think their exist some node & indexed in such a way that every node hold correct indexing

0 Base indexing



1 Base indexing



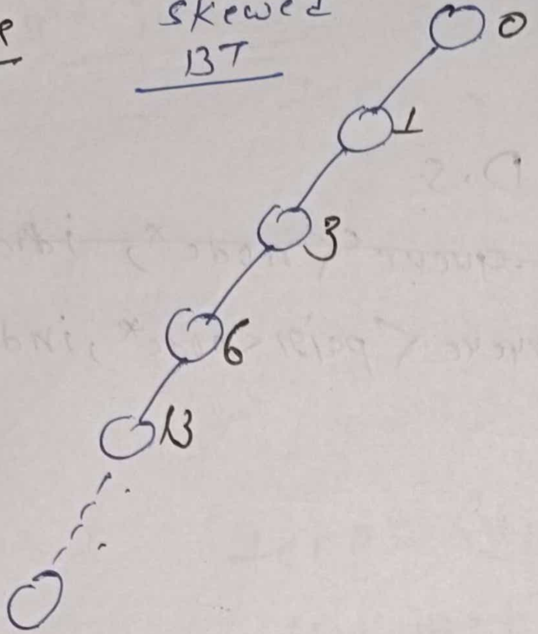


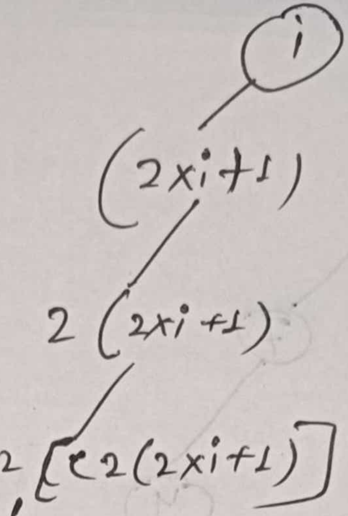
→ Hypothetical think if these nodes are not present the indexing is also maintained

IMP → their is a case where this logic of indexing is failed.

case

skewed BT

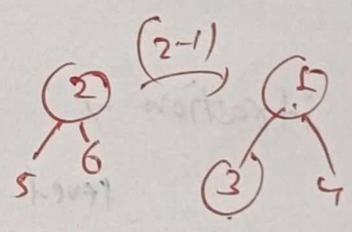


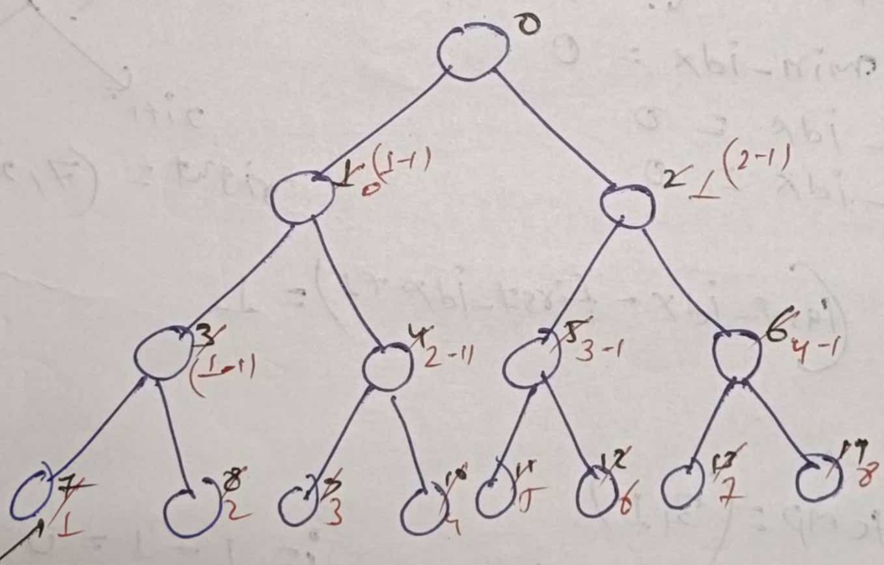


this increment is going in power of two

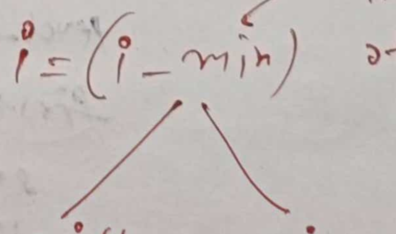
if this skewed tree contain 10^5 some elements then this will lead to overflow.

How to prevent overflow





ideally this should have been 7 but somehow we convert this into 1.



this the minimum of the level

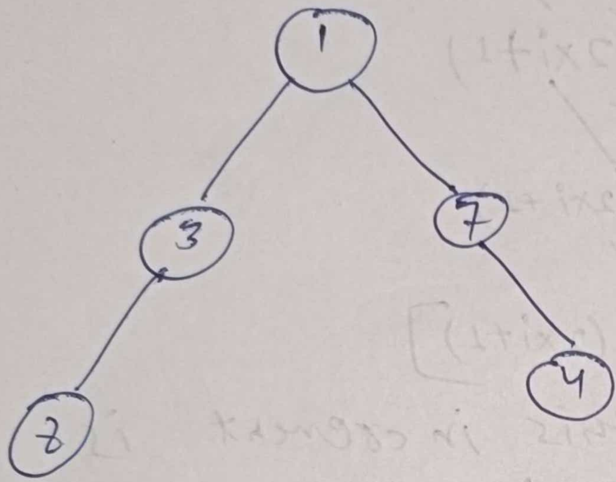
idea

Dry Run

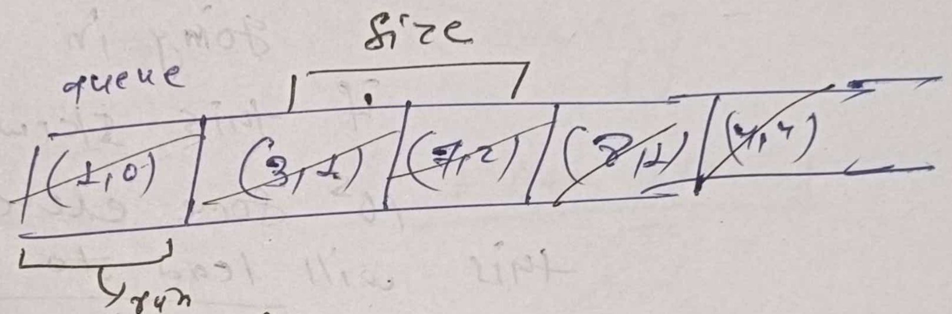
D.S.

~~queue<node*, index>~~

queue<pair<node*, index>>;



initially



~~while (!q.empty~~ loop for this size

left child (3,1)

= i = (i - min_idx)

2i+1   2i+2

1^{st} itration → tem = (1, 0)

level_min_idx = 0

first_idx = 0

last_idx = 0

right = (7,2)

width = (last_idx - first_idx + 1) = 1

2^{nd} itration → temp = (3,1)

level_min_idx = 1

first_idx = 1

left = (8, 1)

right = X

i = 1 - 1 = 0

temp = (7,2)

last_idx = 2

right = (4, 4)

i = 2

i = 2 - 1 = 1

width = 2 - 1 + 1

= 3

3rd iteration

temp = (8,1)

min-idx = 1

first-idx = 1

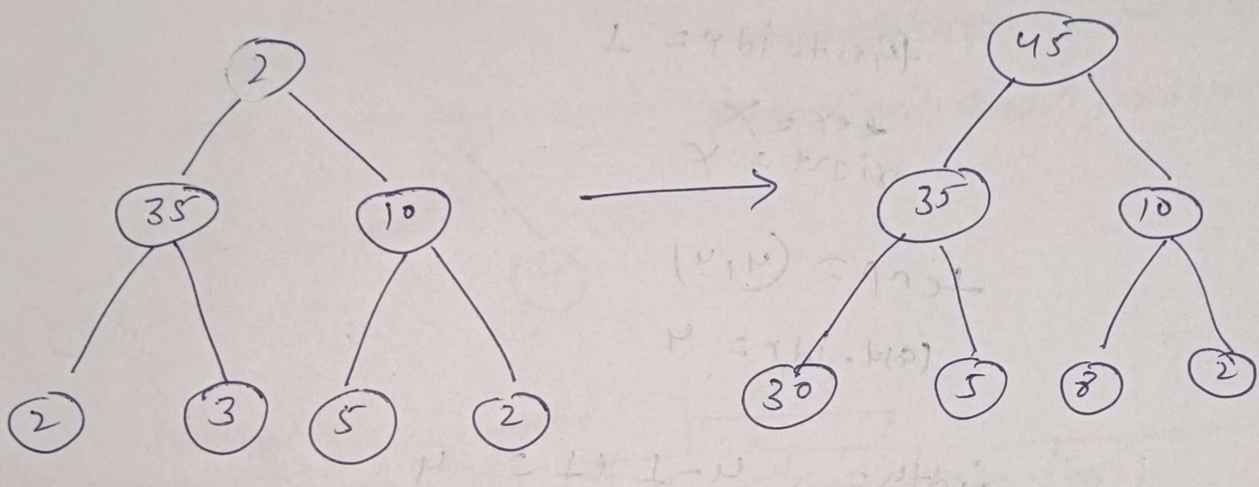
left = X
right = X

temp = (4,4)

last-idx = 4

width = 4 - 1 + 1 = 4

Children Sum Property

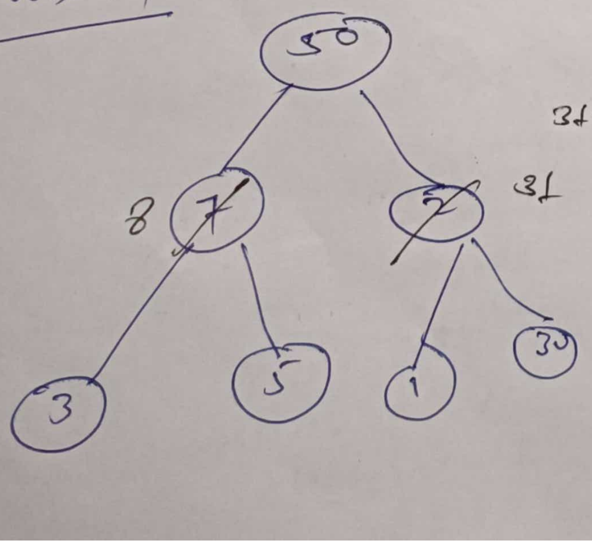


Que:- if BT doesn't follow the children sum property then convert it in such a way that it follow the children sum property

how we can convert → we can increase node's value by +1 as many times as we want.

$$\boxed{node \rightarrow val = left \rightarrow val + right \rightarrow val}$$

Problem

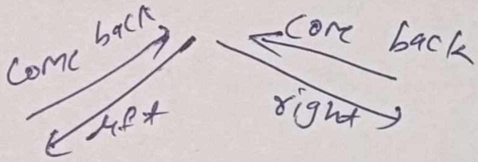


31+8 = 39 but root = 50 that means you can't decrease the value of ~~root~~ root.

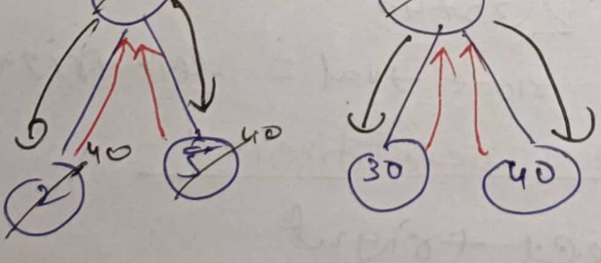
Imp:- Take the advantage that we can increase the node's value by +1 as many times as we want.

Solution

To solve this problem we are using recursive traversal in which we go left then come back from left then right & come back from right



So while going left we do something & while going right we do something & after comming back from both we do something



at root = 40
left = 10   right = 20
10+20 = 30 < 40
so if (l+r < root) then we change the l = root & r = root

Now call for root → left
val = 40
l = 2   r = 5
2+5 = 7 < 40
so if (l+r < root(40))
then l = 40
r = 40

30+40 = 70 >= 40

Now call for left that means 40 (2 initi gives)
~~So~~ & this is a ~~last~~ leaf so
come back

call for right this is also a leaf
come back

Now we are at root → left = 40 come back
from left & right recursive call.

so root → left = l + r = 40 + 40 = 80

~~Node~~
Note :-

(i) while going in recursive call

case - 1 → node → val > left + right
40 > 10 + 20
then left = node → val = 40
right = node → val = 40

~~10~~ 40
~~20~~ 40

case - 2 → node → val < left + right
40 < 30 + 40
then node → val = left + right

(ii) while coming back from recursive call

node → val = left + right