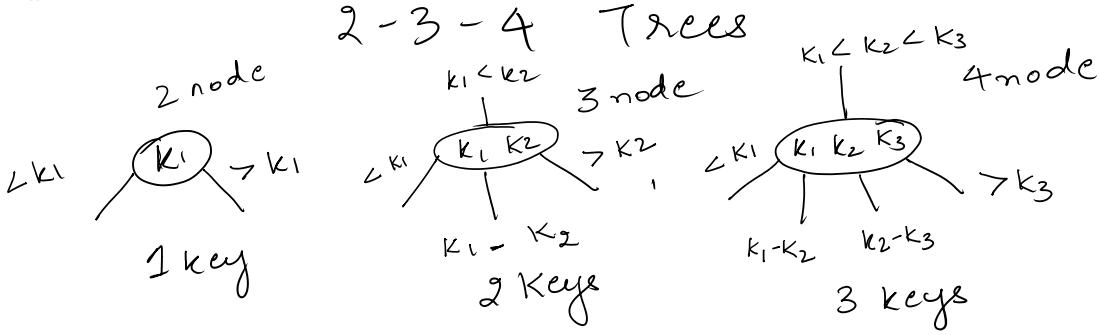
2-3-4 Trees



child nodes

any tree

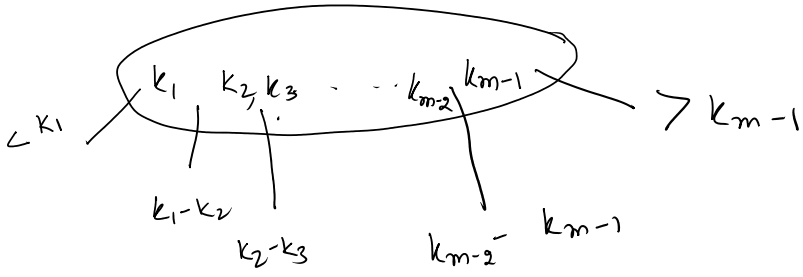
$m = 2 \rightarrow$ Binary tree

$3 \rightarrow$ 2-3 tree

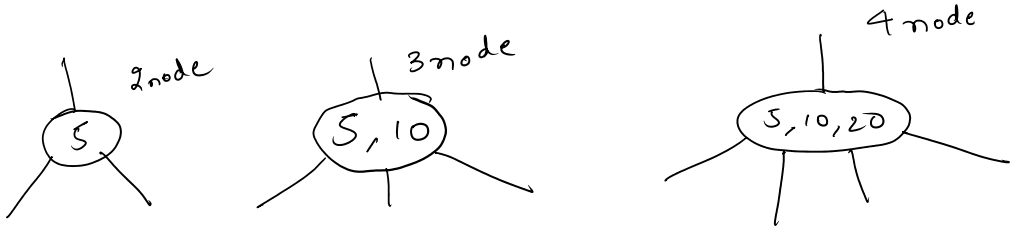
$4 \rightarrow$ 2-3-4 trees

m - Nary tree - B tree

m-ary → each node has upto (m-1) keys



#q



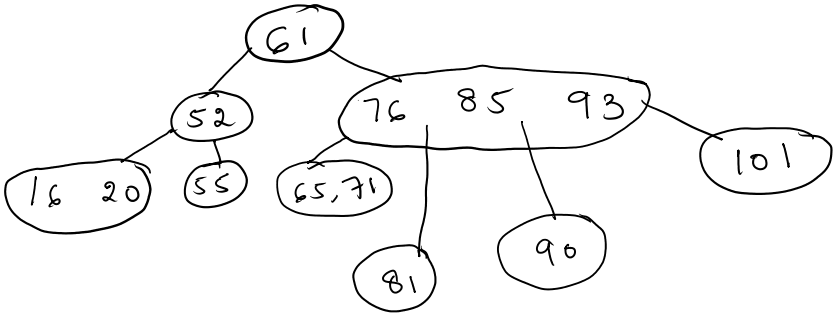
— Similar to 2-3 trees, 2-3-4 trees are always balanced

— Balancing Algo - Splitting nodes

↳ Top Down Approach

* If a node has more than 1 key, the keys must be placed in a sorted order

#g



1. INSERTION

- Insert at leaf position
- While going down the tree to insert a key, if encounter any full nodes (3 keys / 4 child node / 4-node)

Split it.

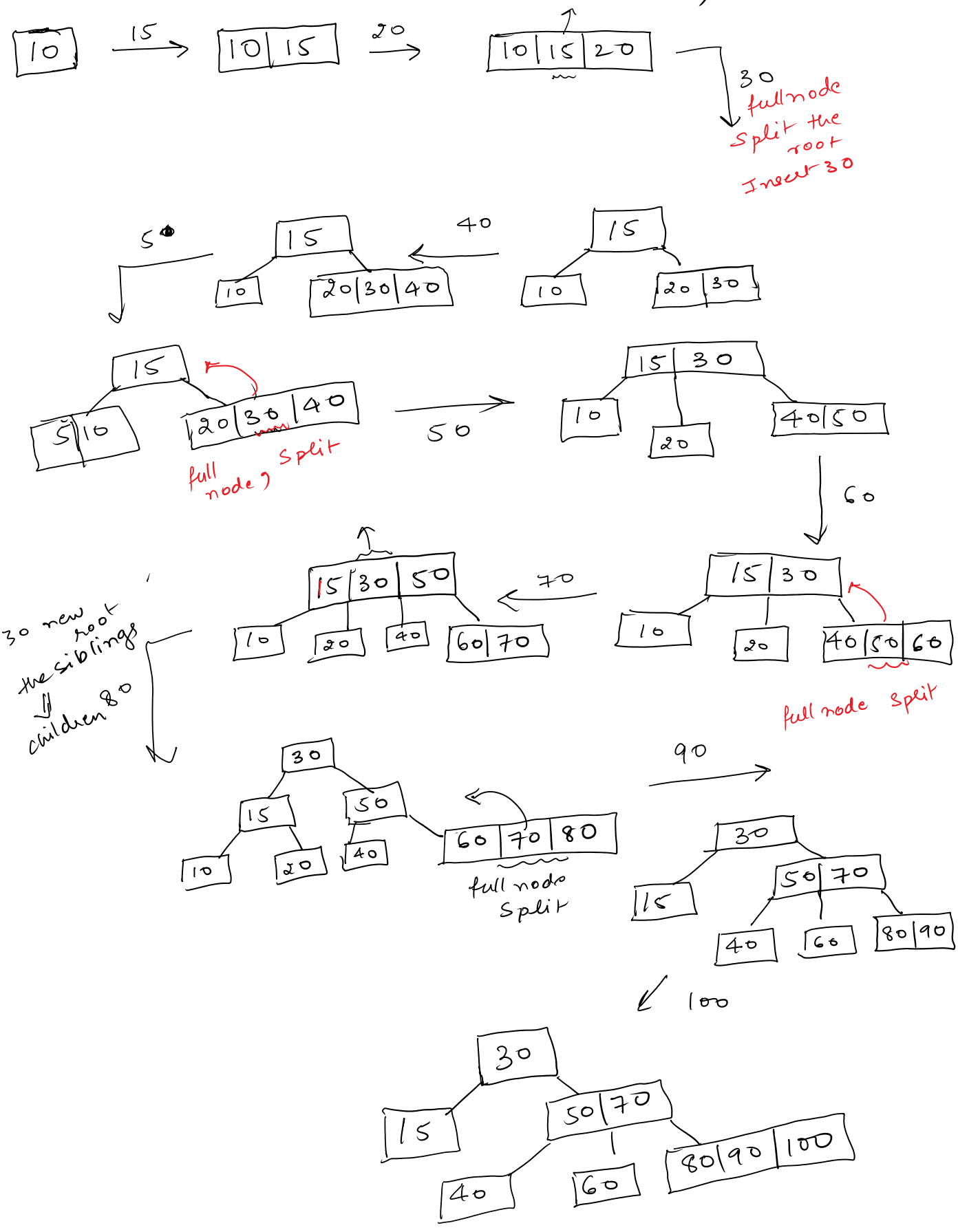
↳ Pick middle element & move it to the parent

* If its the root node, it has no parent. Split and create a new root.

#g 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100

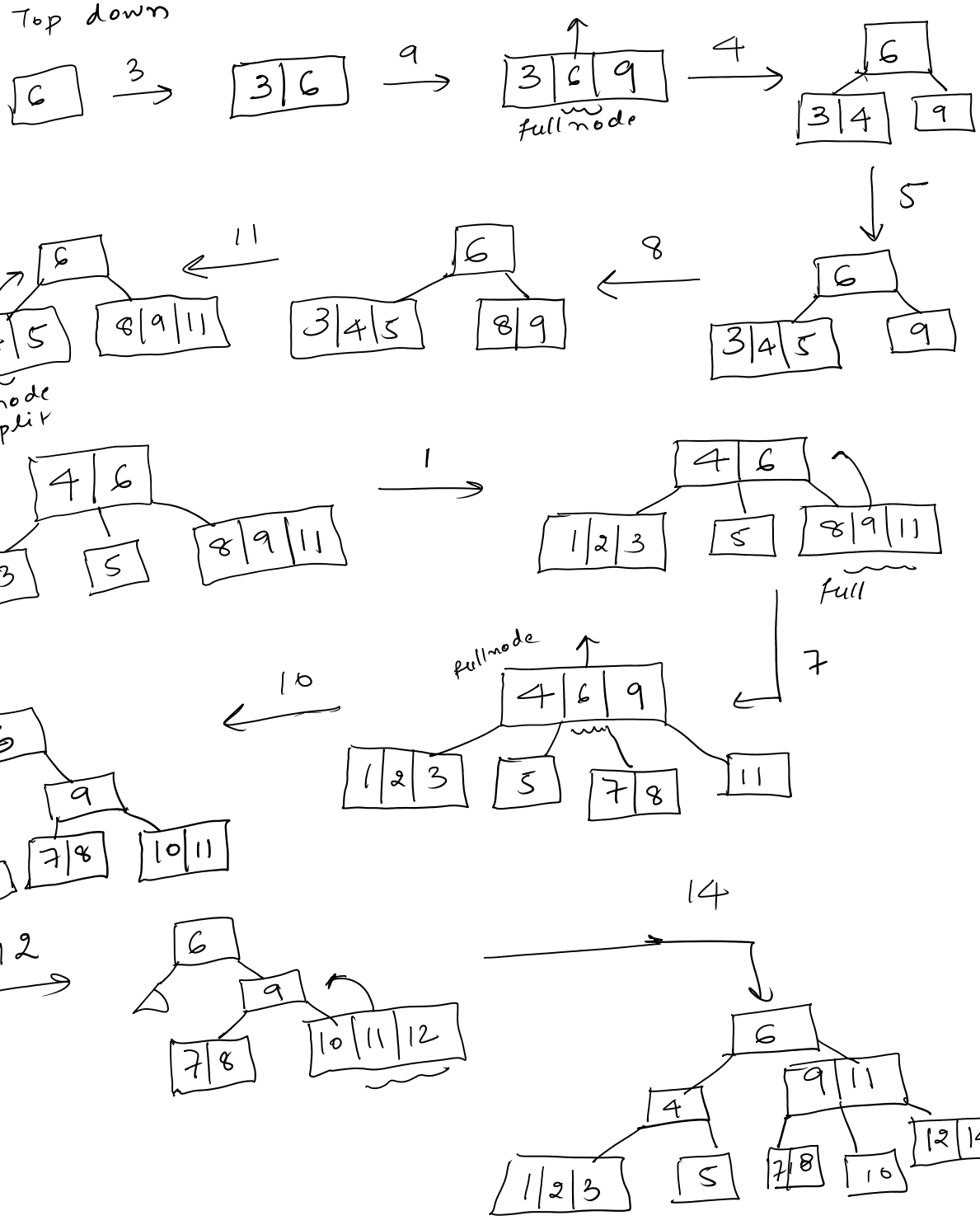
[10] —15→ [10|15] —20→ [10|15|20] —

#g


10
15
10 15
20
10 15 20
30
full node
Split the root
Insert 30
5
15
10
20 30 40
40
15
10
20 30
15
5 10
20 30 40
full node
Split
50
15 30
10
20
40 50
60
15 30
10
20
40 50 60
full node Split
70
15 30 50
10
20
40
60 70
30 new root
the siblings
children 80
30
15
50
10
20
40
60 70 80
full node
Split
90
30
15
50 70
40
60
80 90
100
30
15
50 70
40
60
80 90 100

#g. 6, 3, 9, 4, 5, 8, 11, 2, 1, 7, 10, 12, 14

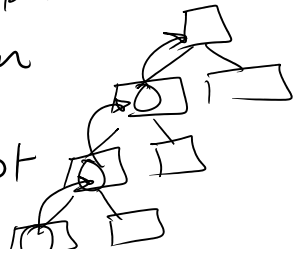
Top down



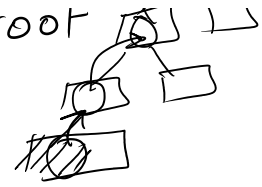
Bottom up - worst case - $O(\log N)$ - splits

↳ split at the leaf position

propogate up till the root



propogate up till the root



→ Top down approach

- less number of splits

in comparison to bottom up approach