Multi-way Trees

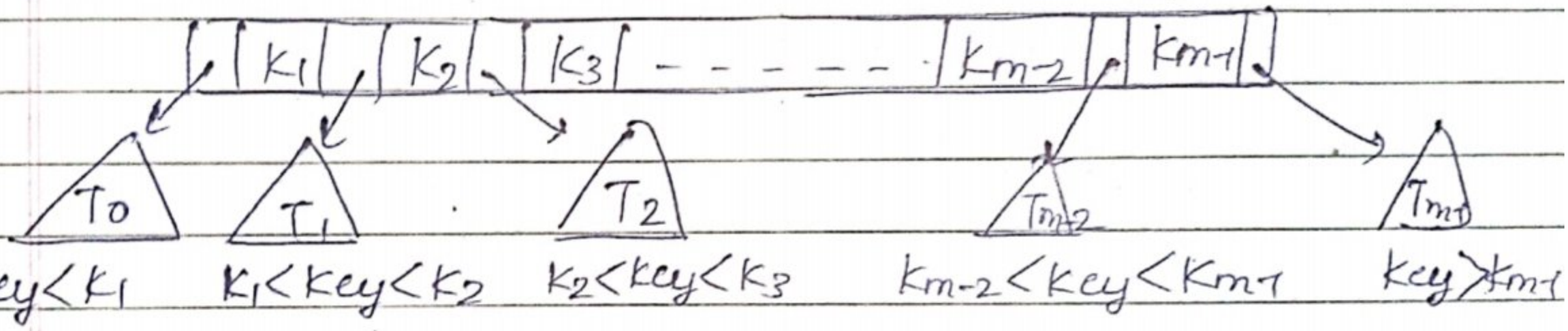
order = m

Each node → atmost m subtrees

(Subtrees may be empty

Atleast 1 & atmost m-1 dist. keys

→ Keys in each node are sorted



→ Order of keys & subtrees

T0, K1, T1, K2, T2, ..., Km-1, Tm-1

such that

→ All keys in T0 are less than K1

→ All keys in subtree Ti, 1<=m<=m-2, are greater than Ki but less than Ki+1

Not ⇒ The leaf nodes need not be at the same level

→ A non-leaf node with n-keys may contain less than n+1 non-empty subtrees

B-tree

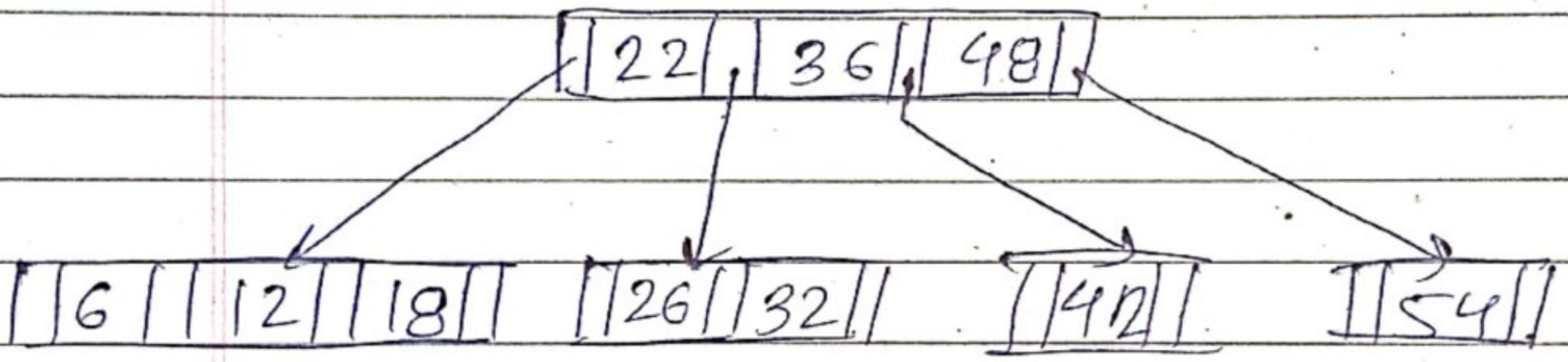
"Order" or "Branching factor" = m

→ Non-root nodes = atleast (m-1)/2 keys

→ Each node : max. m-1 keys & m subtrees

E.g.

order = 4



⇒ For representing huge tables on second-memory

⇒ But for large branching factor m, ht. of B-tree is low resulting in fewer disk accesses.

Note: As m inc. amount of ~~comp~~ computation of each node inc., however this cost is negligible as ~~comp.~~ to hard-drive accesses compared

⇒ Or B.F. is chosen such that block corr. to block of second-memory

⇒ Most comm. data struct. used for DB indices is B-tree.

Insertion

Overflow condition:

A root node or a non-root node of a B-tree of order m overflows if, after a key ins., it contains m-keys.

Algo.

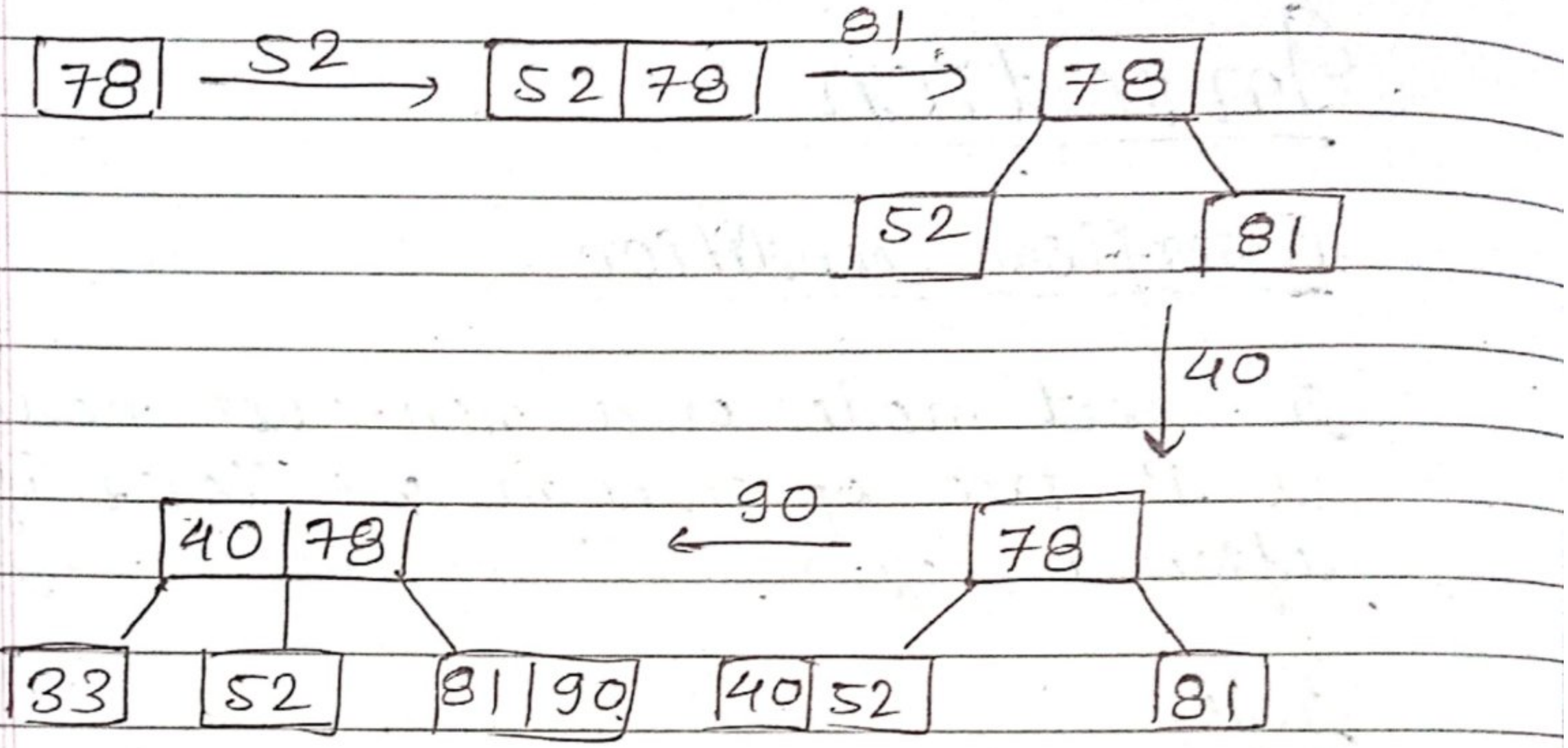
If a node overflows, split it into two, propogate the ~~middle~~ "middle" key to the parent of the node. If the parent overflows the process prop. upwards. If the node has no parent, create a new root node.

Note: Insertion of a key always starts at leaf.

order = 3 Max. = P-1 = 3-1 = 2

Min. key = ceil[3/2] -1 = 2-1 = 1

78, 52, 81, 40, 33, 90, 85, 20, 38



Ans. In a B-tree of even order

Right-bias :

Right-subtree has more keys than

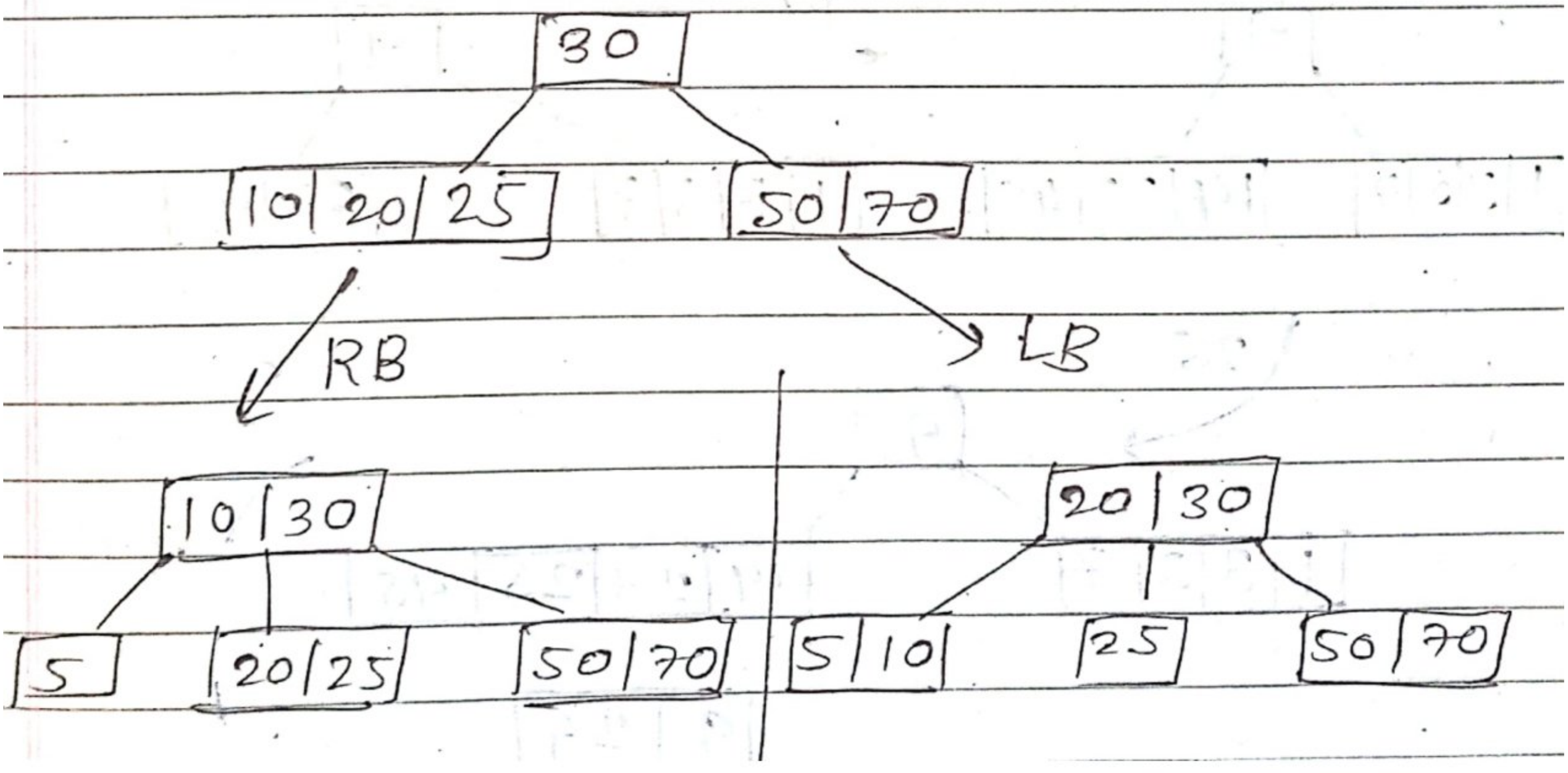
Keys in RST > Keys in LST

Left-bias

Keys in LST > keys in RST

E.g.

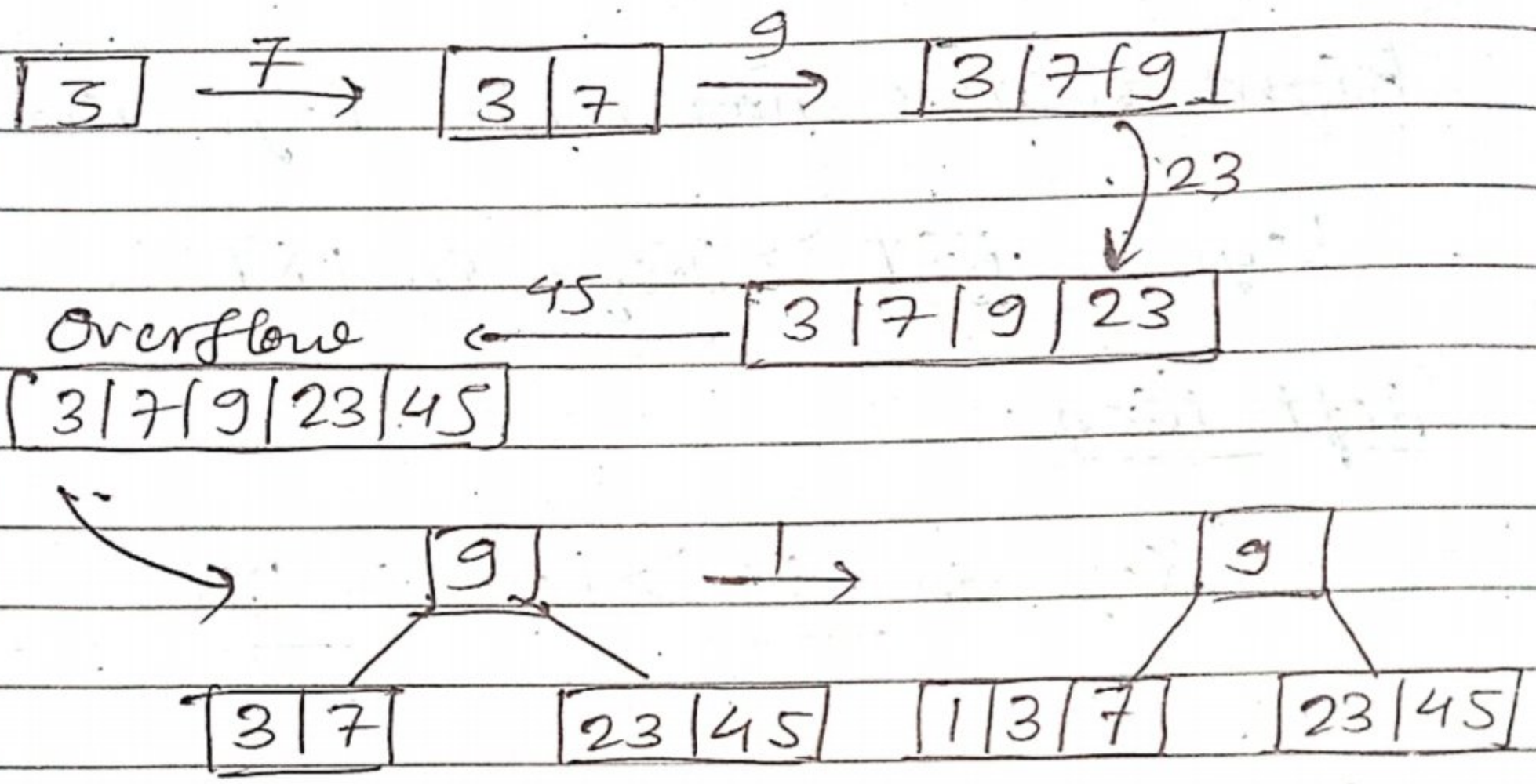
Insert 5 in foll. B-tree of ~~order~~ order 4 :



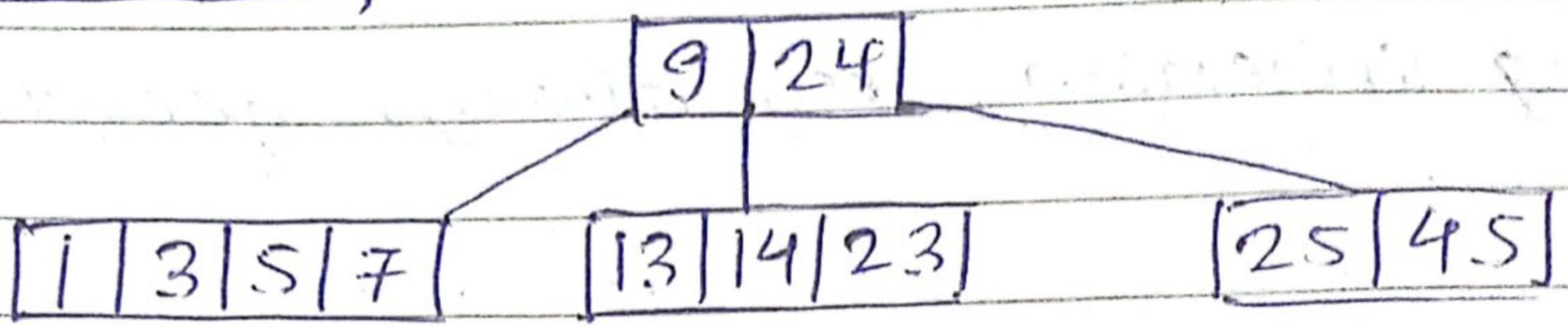
Q. Atleast = 2 Atmost = 4

5 - way B-tree

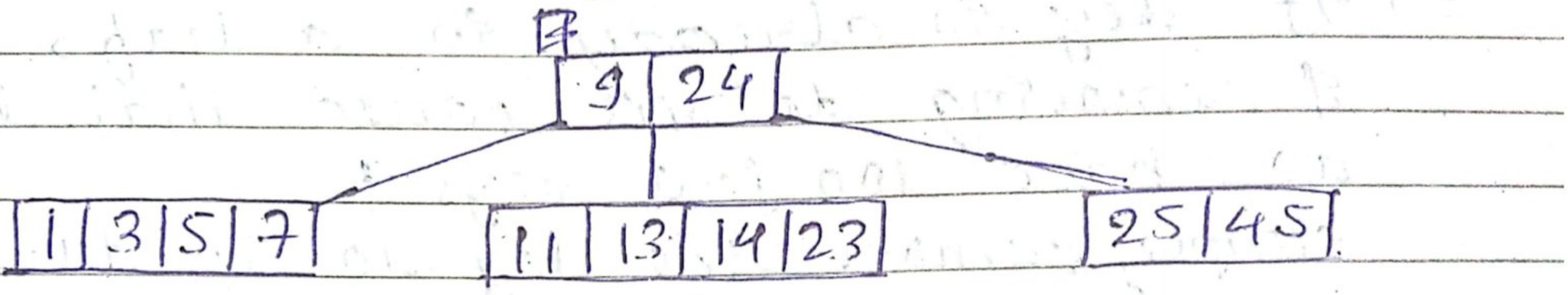
3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8



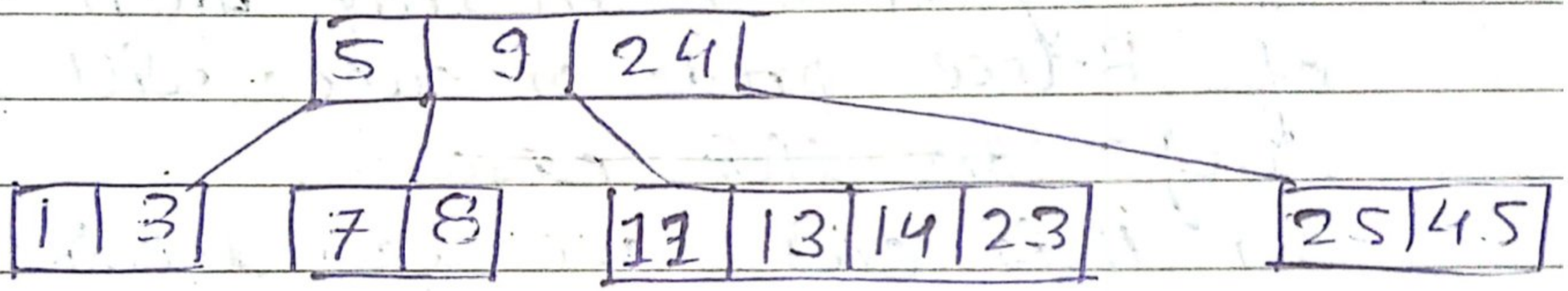
13 →


9 24
1 3 5 7
13 14 23
25 45

↓ 11


9 24
1 3 5 7
11 13 14 23
25 45

↓ 8


5 9 24
1 3
7 8
11 13 14 23
25 45

Deletion of from B-tree:

During insertion, key always goes to leaf

⇒ of key

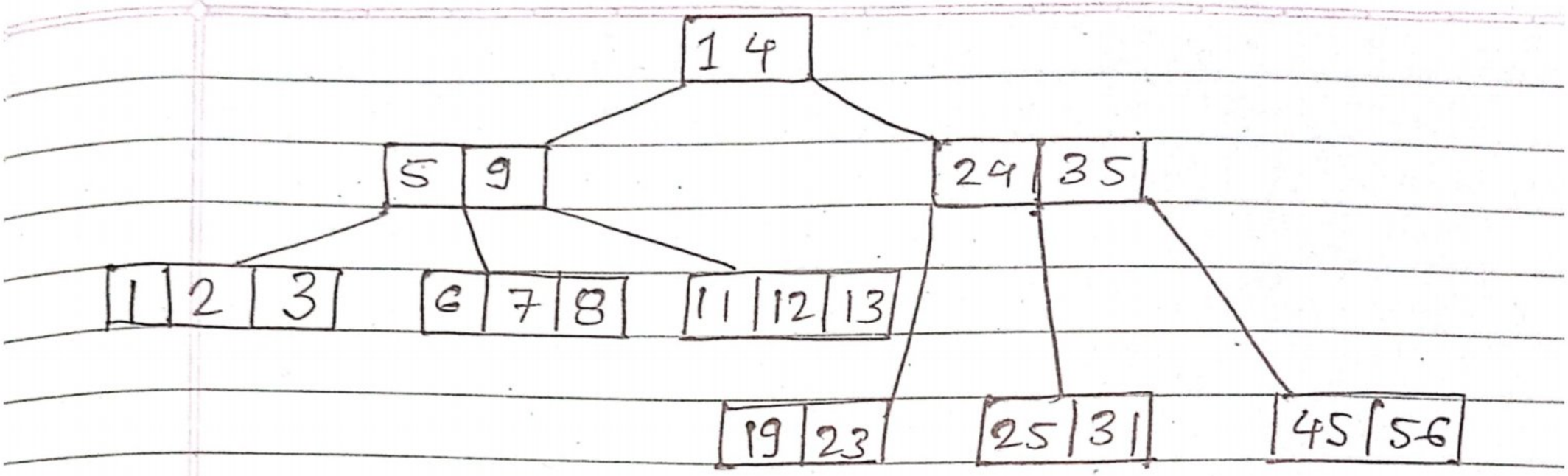
⇒ If key is already in a leaf, & removing doesn't cause that leaf to have too few keys & simply remove the key to be del.

⇒ If key is not in leaf then by prop. of B-tree pred. or succ. will be in leaf - ~~In this case~~ In this case we can delete the key & promote the predecessor or ~~succes~~ successor key to non-leaf deleted key's pos.

⇒ If one of them has more than the min. no. of keys then we can promote one of it's keys to the parent & take the parent key into our lacking leaf.

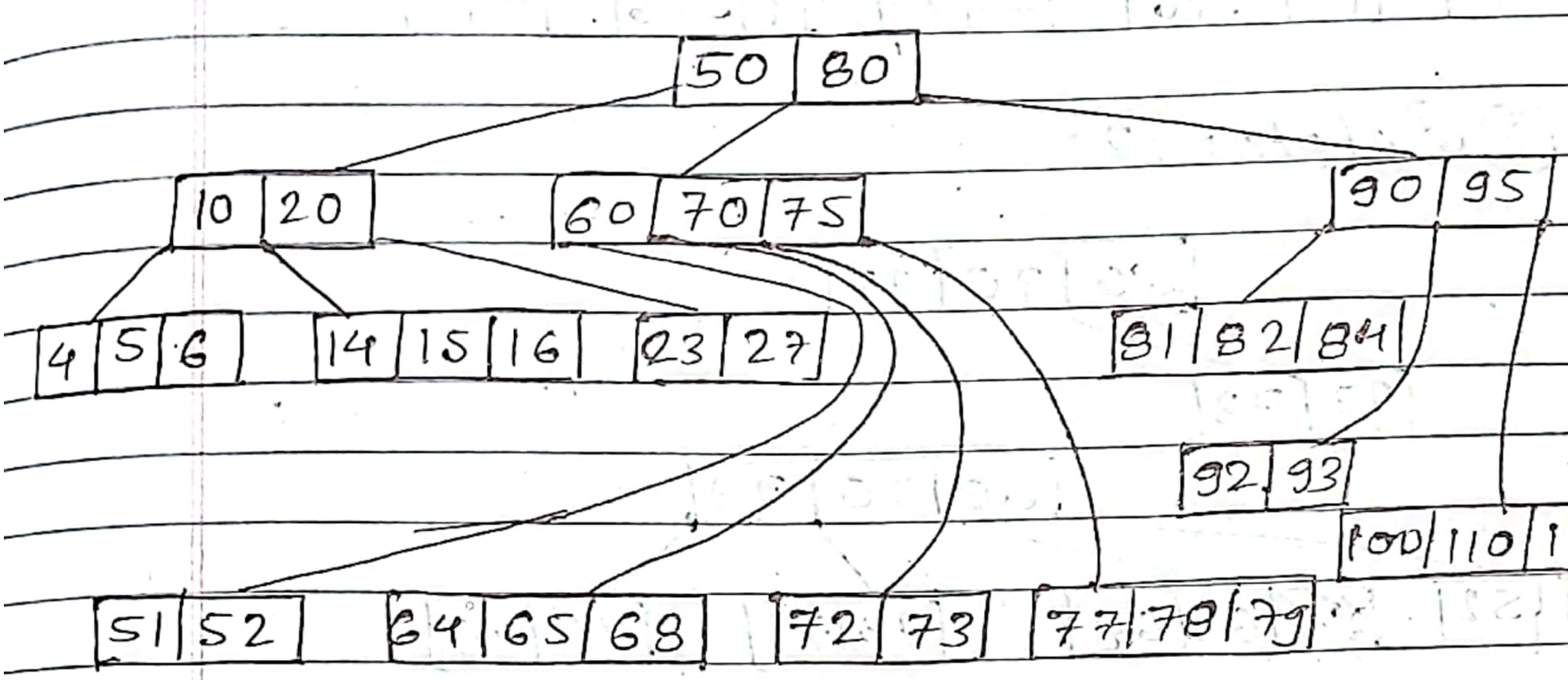
Successor of key(k): Smallest key greater than k.

Predesor of Key (k): Largest key smaller than k.



⇒ If neither of them has more than the min. no. of keys then locking leaf & one of its neighbours can be combined with their shared parent & the new leaf would have correct no. of keys;
If this step leaves parent with too few keys, then repeat the process upto root if reqd.

Deletion from B-tree



Order = 5 (p) Min. = $\lceil p/2 \rceil = 3$

Min. children : ceil $(p/2)$ or $\lceil p/2 \rceil = 3$

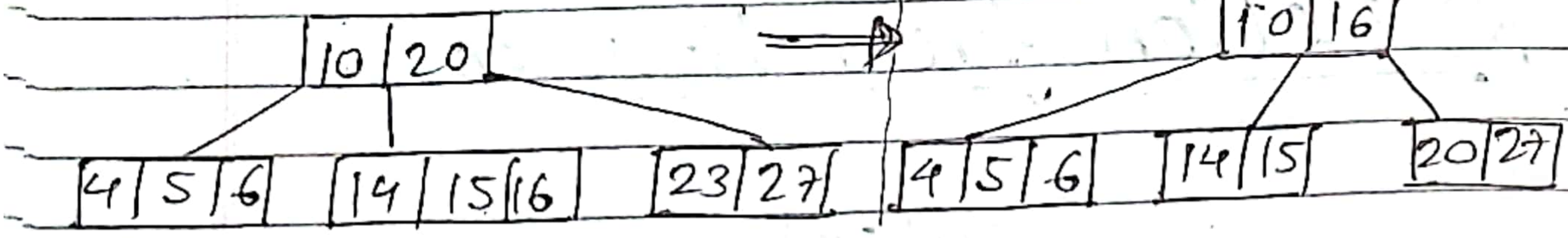
Max. children : $p = 5$

Min. keys : $\lceil p/2 \rceil - 1 = 2$

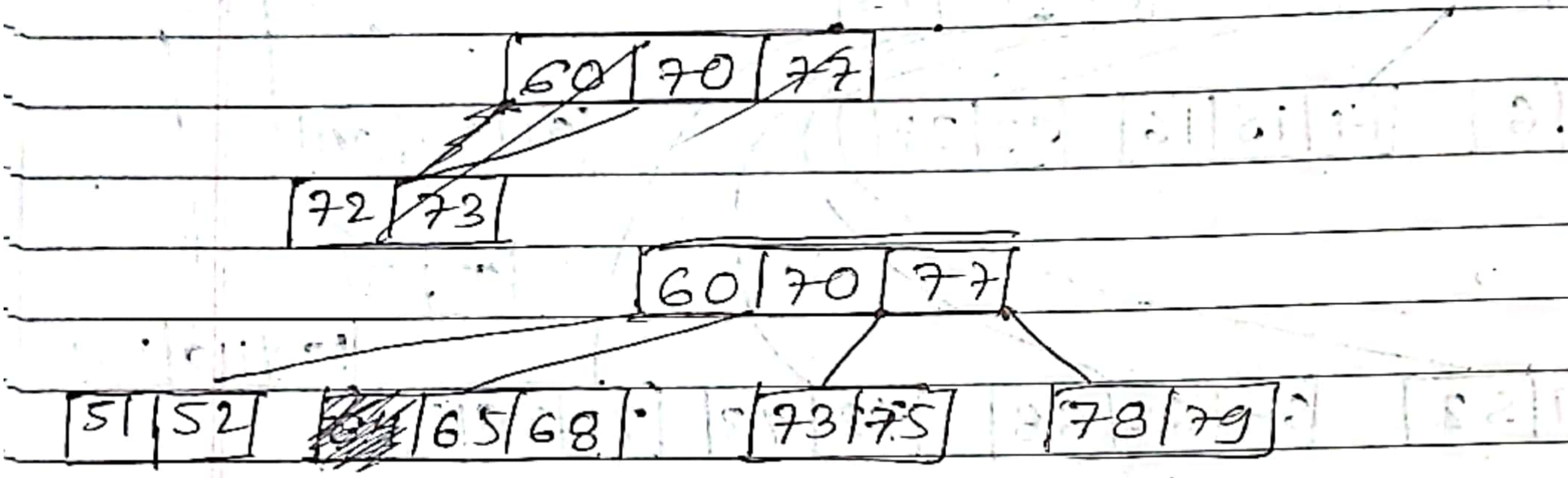
Max. keys : $p - 1 = 4$

① Delete 64 : Delete directly

② Delete 23 :
Change parent to 16 & make 20 as child

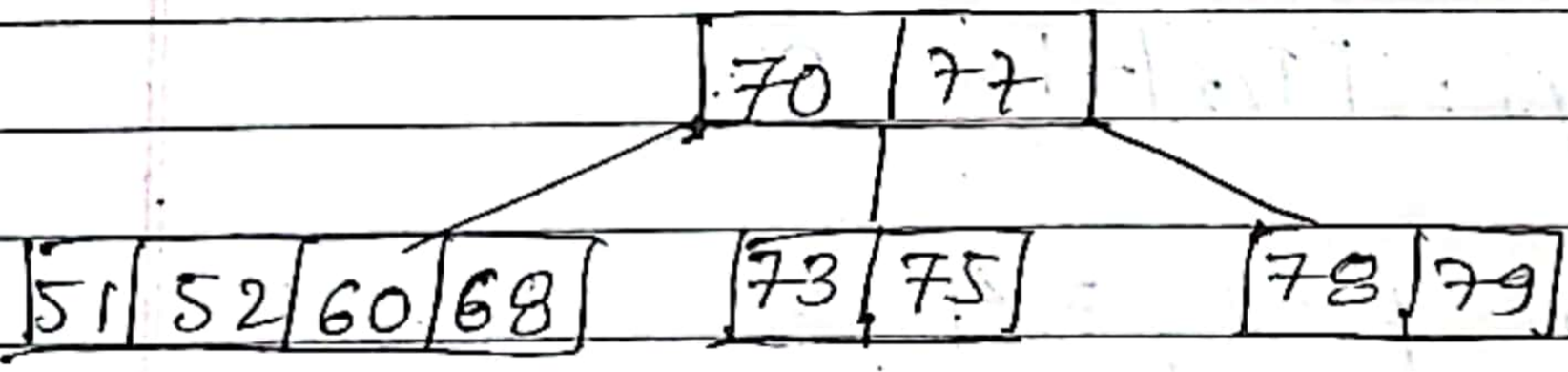


③ Delete : 72:

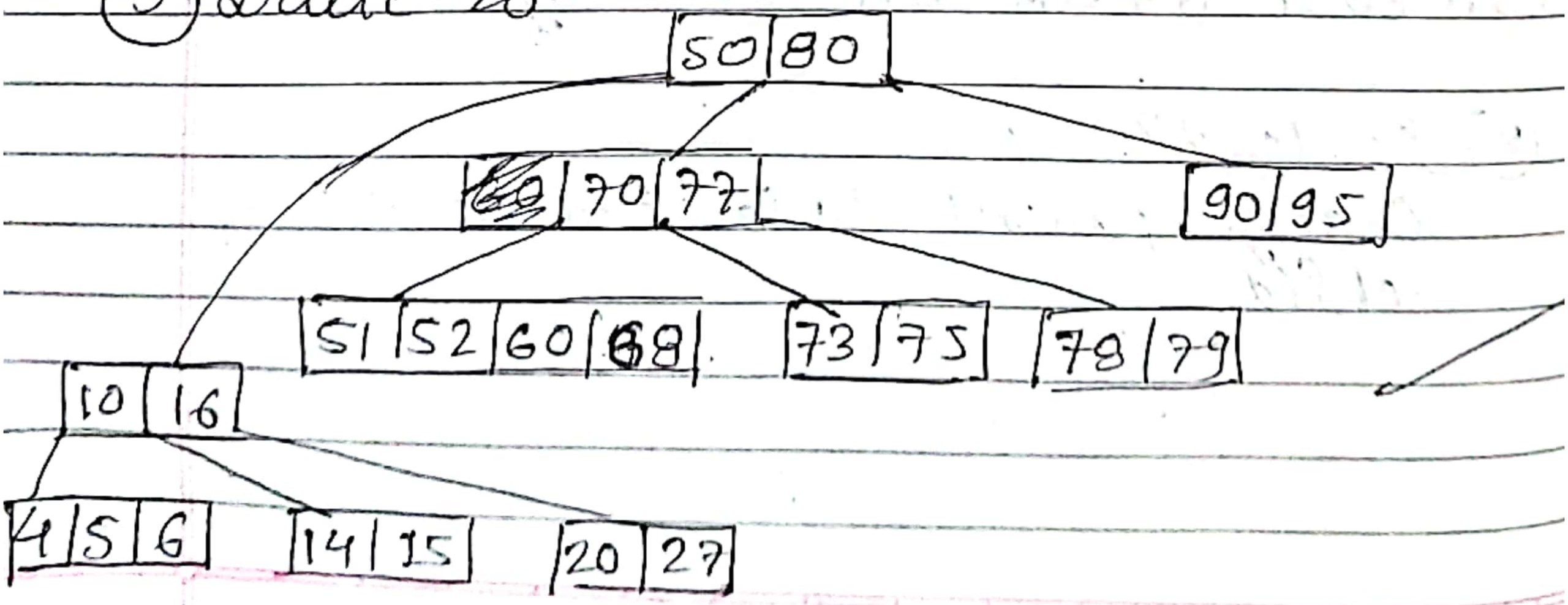


④ Delete 65

Cannot delete directly; cannot borrow so, combine delete it then combine two childsen & bring one parent down



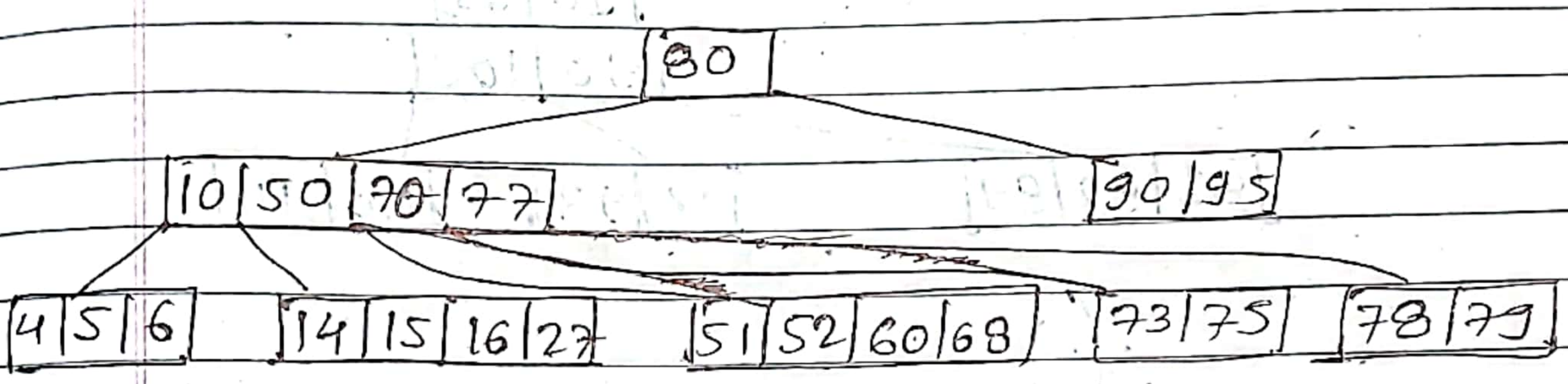
⑤ Delete 20



S-1 : Delete 20 & Bring 16 down to make the child as ~~14|15|6~~ [14|15|16|27]

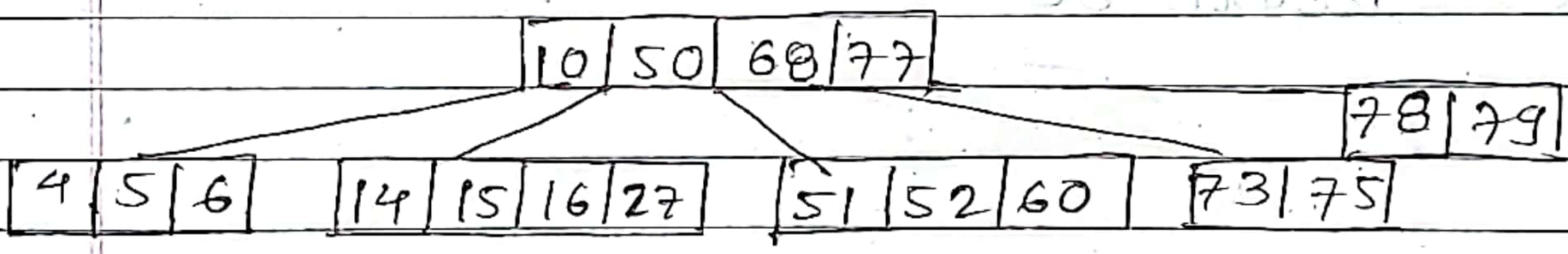
→ Now Min. key prop. is voited at node 10

→ So, bring 50 down & combine with sibling [70|77]

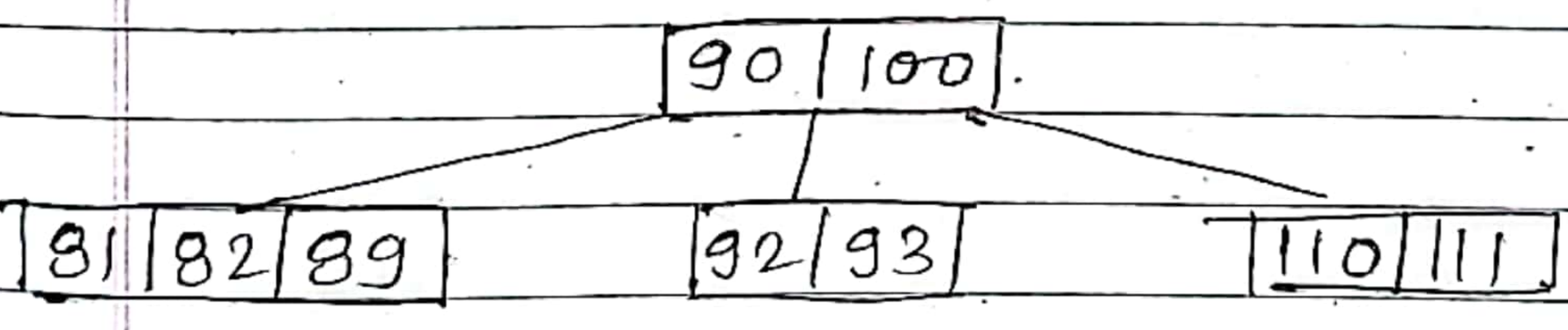


⑥ Delete : 70

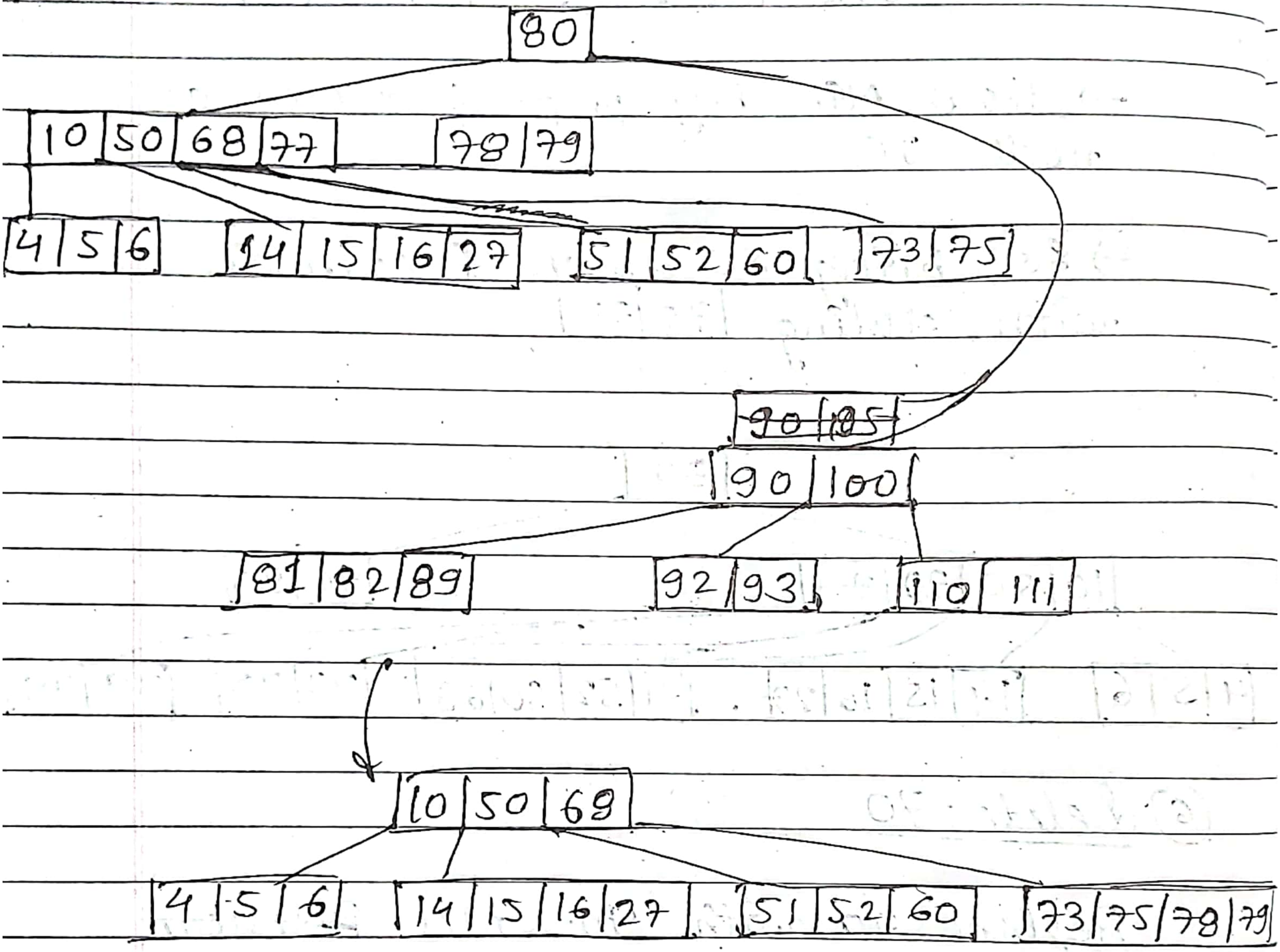
Replace 70 with In-order ~~Predessor~~ Predessor 68



⑦ Delete 95



(8) Delete 77


80
10 50 68 77
78 79
4 5 6
14 15 16 27
51 52 60
73 75
90 95
90 100
81 82 89
92 93
110 111
10 50 69
4 5 6
14 15 16 27
51 52 60
73 75 78 79