<1>

(a)

Use quick sort or merge sort to sort the list of numbers first. The sorting process will take $O(n \log n)$

After sorting, set a variable smallest_difference, by looping through the sorted list comparing S[i] and S[i+1], we updata the smallest_difference variable. (Because there won't exist a S[j] that is not S[i+1]/S[i-1] that has a smaller difference to S[i] in a sorted list)

The loop through n elements takes $O(n)$.

$O(n) + O(n \log n) = O(n \log n)$

Done.

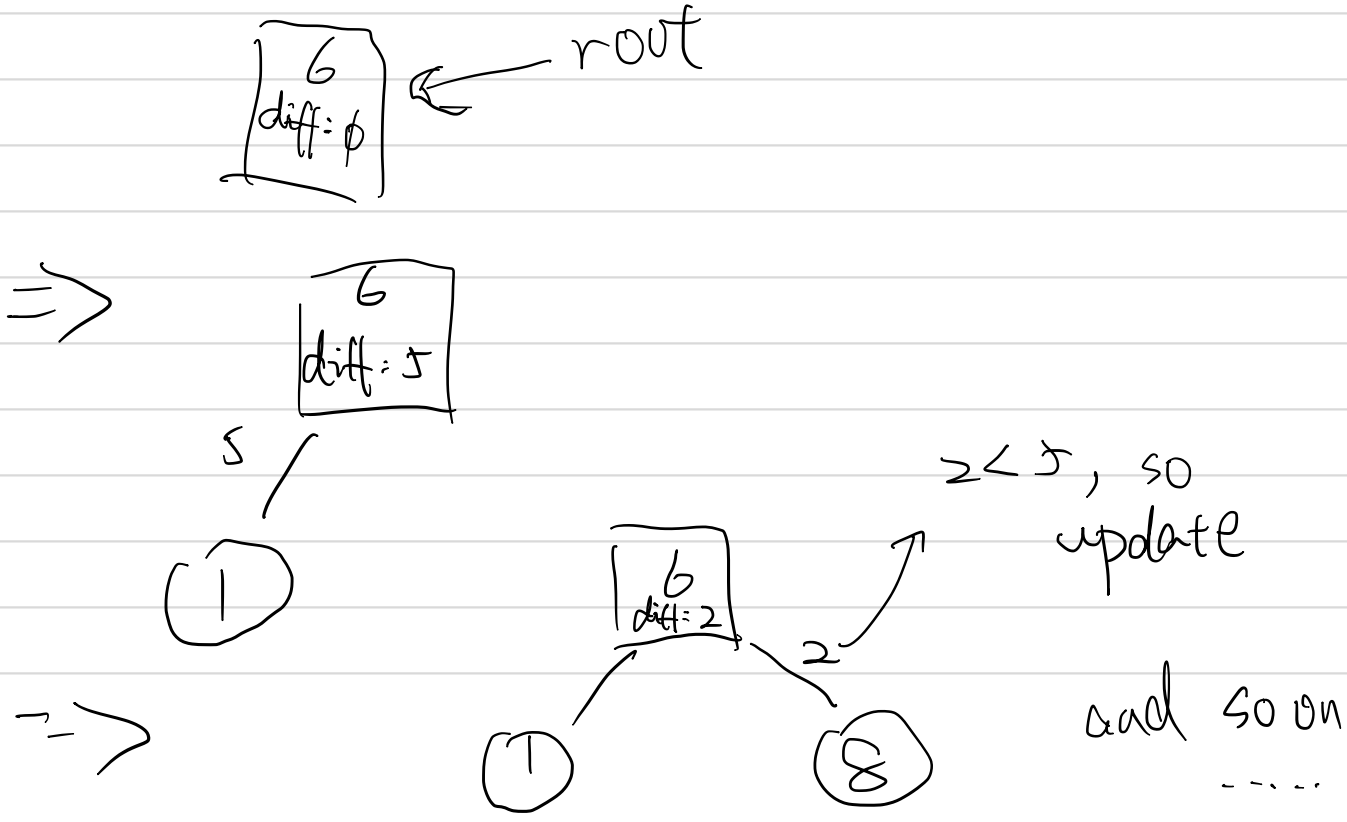
(b) The way to allow insertion while answering queries in constant time is:

① Build a red black tree with every insertion of data (Takes logn to insert every data in to Red Black tree (self-balanced struc))

② While inserting, due to property of binary tree, while comparing data to the node value, also calculate out the difference with each node on its way down. If the difference between the new data and one of the nodes smaller than the current smallest difference (can be stored in root), update it. ⇒

Because a new data at most will compares to O(logn) nodes in insertion. The calculation of difference takes constant time, so overall insert takes O(logn).

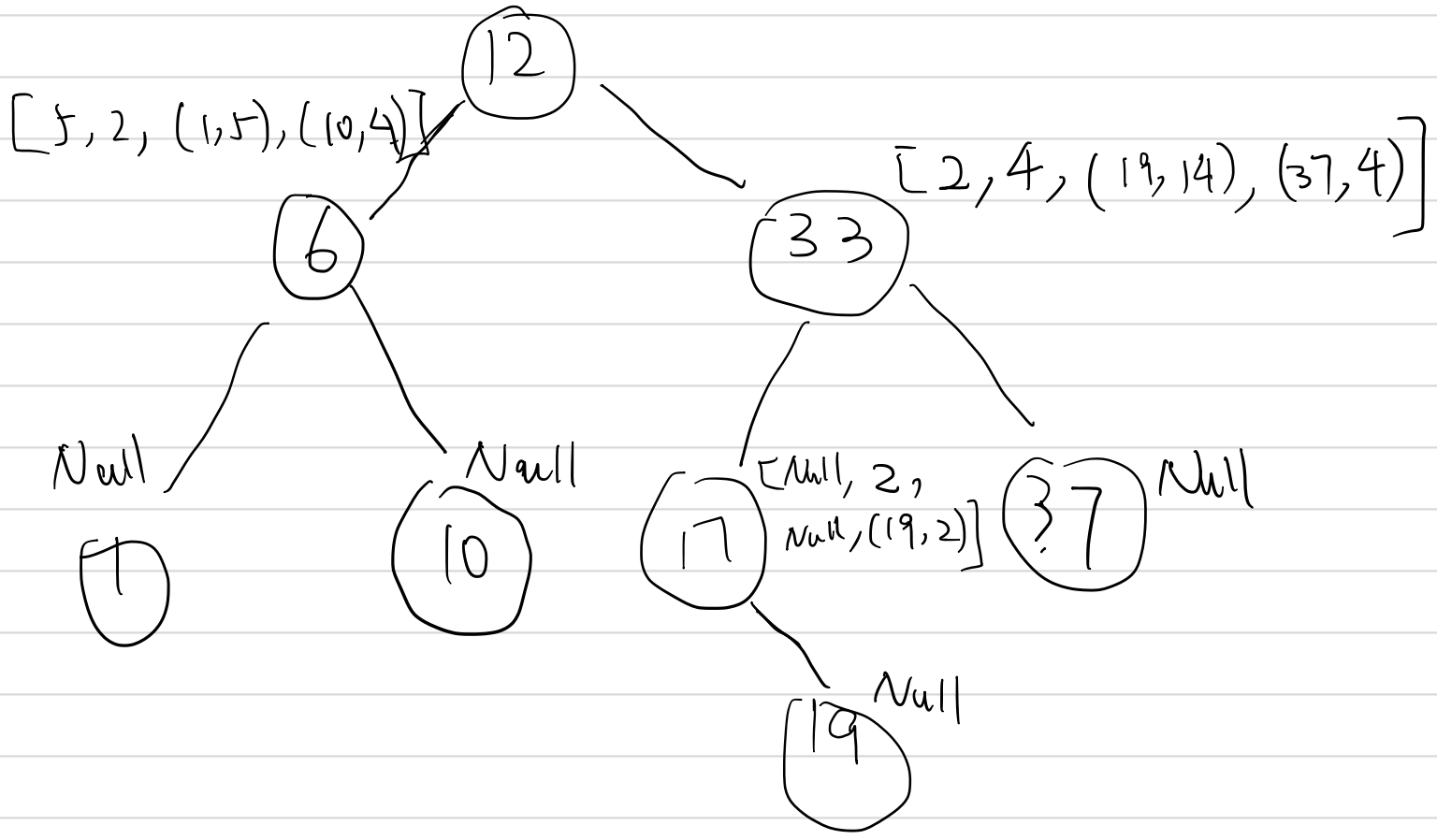
To extract the smallest difference, just simply extract the difference value stored in root. which takes O(1), constant time process as shown below:

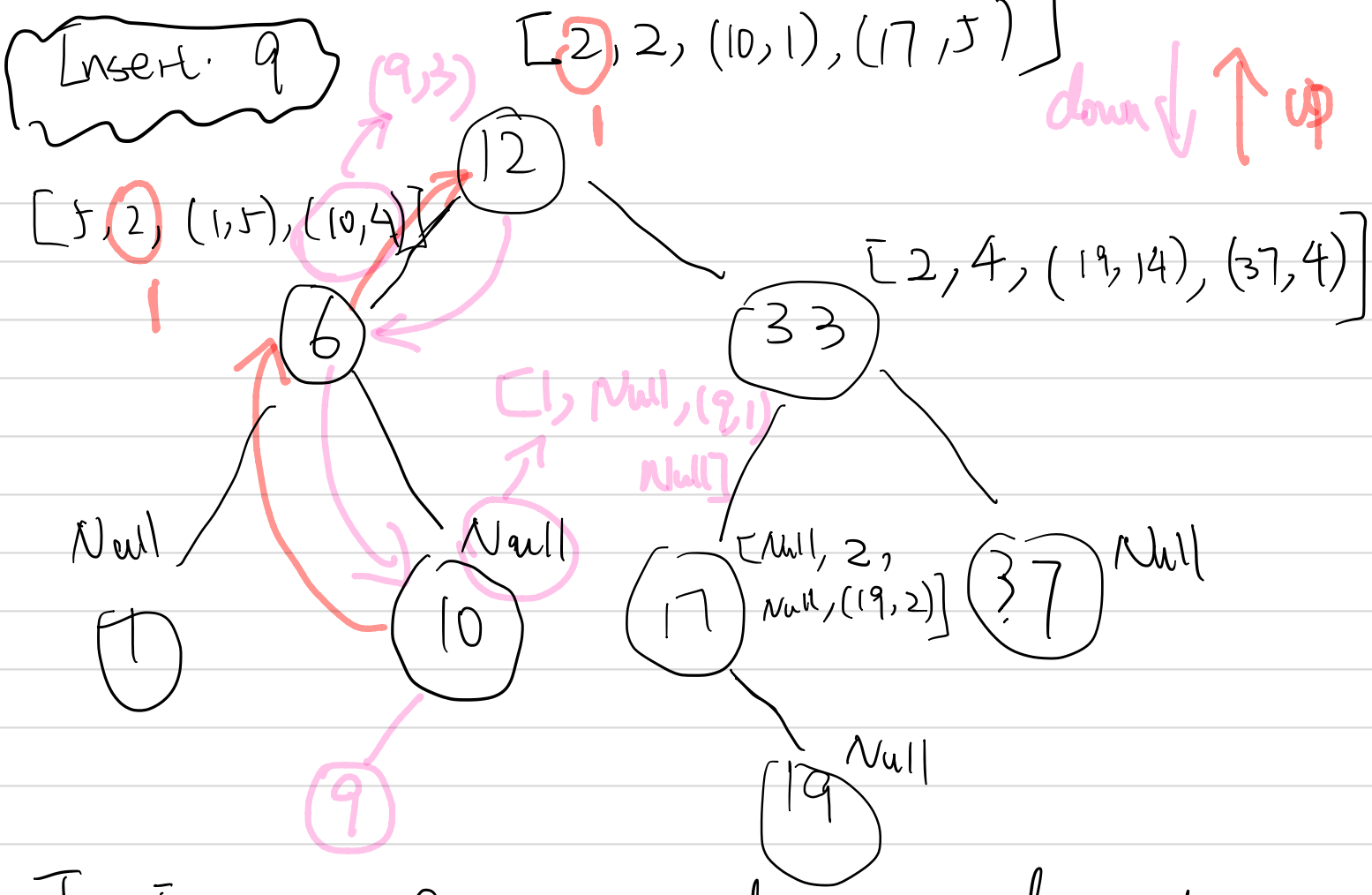


(3) To delete properly, extra informations stored in every node is

[smallest diff in left subtree, smallest diff in right subtree, (smallest value in left subtree compares to node itself, the difference), (smallest value in right subtree compares to node itself, the diff)]

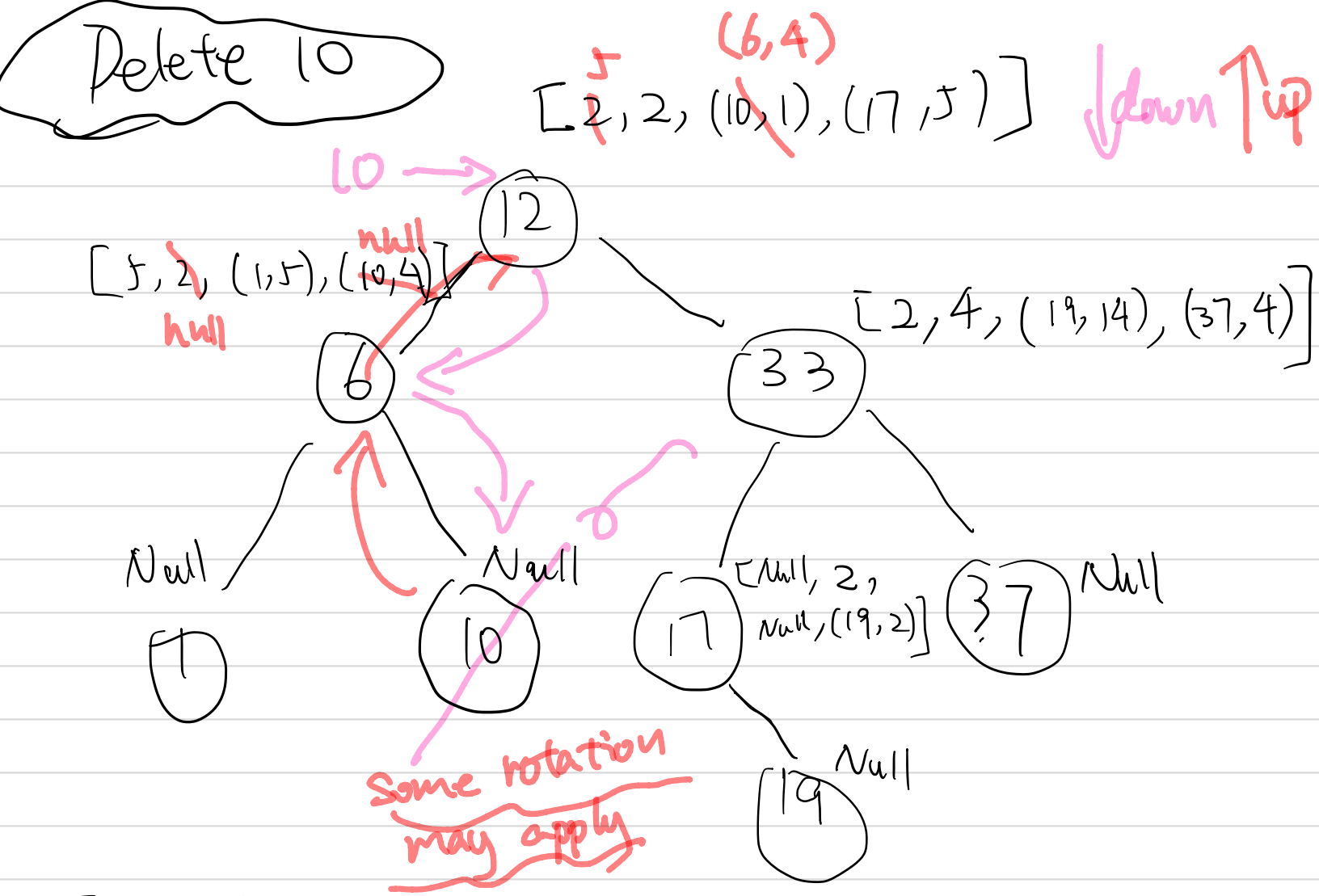
Example : [2, 2, (10, 1), (17, 5)]





To Insert: Go from root down and update the value in left/right tuple value if necessary. Then go up to update the smallest left/right subtree value,

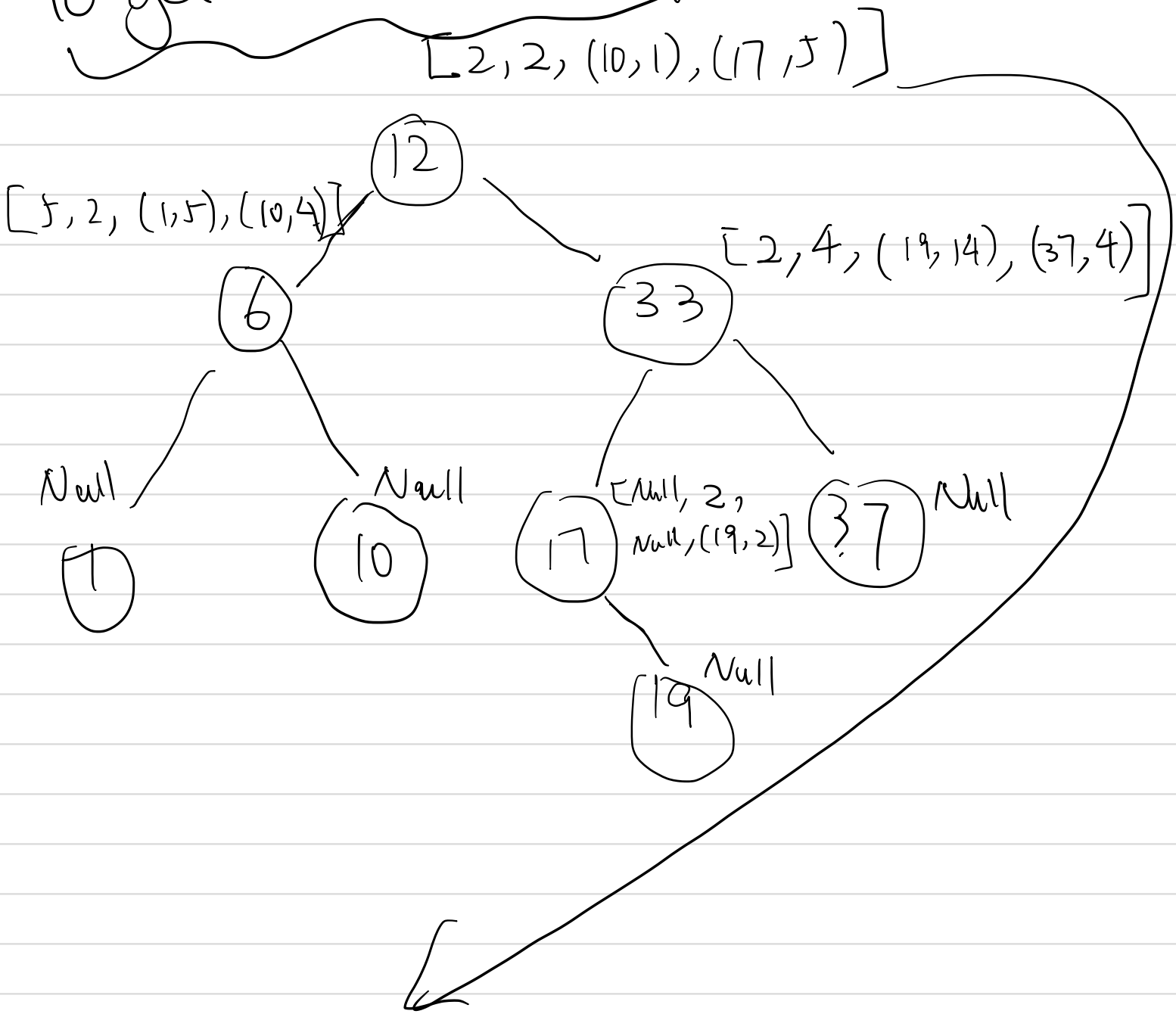
$O(\log n)$



To delete:

Go down to delete, and then rotate corresponding and recalculate. pass up the value. O(log n)

To get the smallest difference:



Go to the root, compare the first two number and the second numbers of last two tuples. done, O(1)