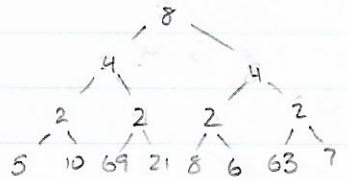
5) As a way of indexing the modified 2-3 tree, the internal nodes would need to store the number of children in a subtree. So in the example provided, the interal nodes would store:



This may not necessarily be all that each internal node stores; however, this will allow the tree to be indexed, so that operations may be easily performed given some k.

i. A list with 1 item does not need to rearrange the list. This is trivial

ii. Simply referring to the notes, a join operation would take at most (if not, much less as ordering of leaves would not matter in this tree) $O(\log n)$ as the internal indices would need to be updated

* iii. If $\lceil k \div 2 \rceil >$ the value stored at the root, then all items in the left subtree could be stored in L_1. Continuing this logic down the right subtree and setting $k = \lceil k/2 \rceil$ gets the path for which the split must be made on. The runtime for this would be $O(\log n)$

iv. The same logic applies as the previous question

Note: For a root (node) with 3 children, accessing all values of the left subtree without considering children would have the condition that $\lceil k/3 \rceil >$ value stored. For the middle child, it would be $\lceil 2k/3 \rceil >$ value stored.

* An alternative (probably more accurate) way would be to store k as a temp variable.

If k < value at current node and current node is not a leaf

    go to the left

else

    k − the value stored at current node (minus)

    go to the right (or middle)

↳ Repeat until k=0 to get L_1 and L_2

Merge remaining trees using join algo from class