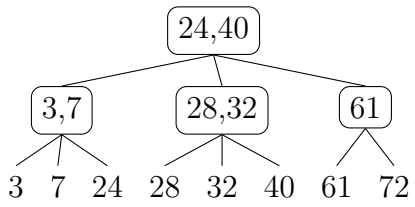
Homework 5, additional problems, solution set

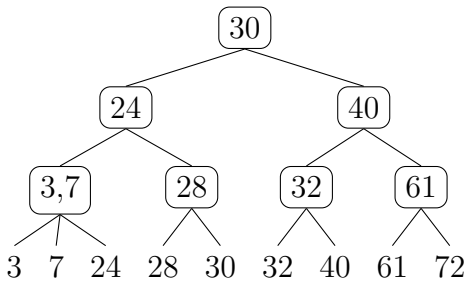
1. It's (iii), the depth can decrease but it does not have to.

If the initial tree stores 3 items, then it begins with depth 1. The insertion causes the root node to split, making its depth 2. Now the deletion restores the initial situation.

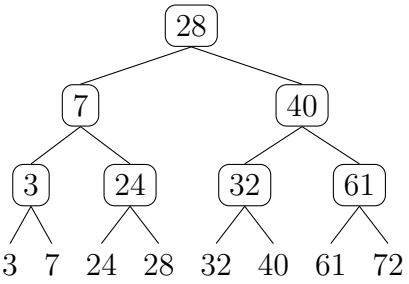
Next, we show an example in which the depth does not decrease after the increase. The initial tree is:



We perform Insert(30), yielding:



Now we perform Delete(30), yielding:



2. We store a record for each account in a 2–3 tree, using the owner's name as the key; each record has two fields: the owner's name and the balance. For each subtree, we keep an additional field, the value of the maximum balance for the accounts in the subtree, and store this value at the parent of the subtree.

Creating an account is an insertion into the 2–3 tree. Closing an account is a deletion. Changing the balance is implemented as follows: search for the record, and when it is found at a leaf ℓ, update the balance stored at this record. We then need to follow the path back to the root, updating the maximum balance for the subtrees which include leaf ℓ. To report the account with the maximum balance, one performs a search from the root, at each step continuing the search in the subtree with the maximum balance (ties, if any, can be broken

arbitrarily). When a leaf node is reached, simply report the balance for the account stored at that node. Reporting a balance to an owner is done by means of a search to find the record for that owner, and then returning the value in the balance field.

The one additional detail to explain is how to maintain the maximum balances when nodes change due to being split or joined, or losing or gaining a subtree. We maintain the invariant that the maximum values are always correct after a change to a node. Note that each maximum for the subtree rooted at v is simply the maximum of the maximum values for v's subtrees. A node change involves transfers of maximum values corresponding to the transfers of subtrees; in addition, the parent p of v may need to gain a maximum value (in the case of a split), lose a maximum value (for a join), or change one or two maximum values (in the case of a subtree transfer to v). But the new values at p are all maxima of values stored at v and its sibling, so can be updated in $O(1)$ time. This ensures the invariant is maintained following the update at node v.

All the operations run in $O(\log n)$ time as they all involve traversing a path from the root to a leaf and back again.

3. We will show that there are at most $2n-1$ internal nodes in a 2–3 tree storing $2n$ items. Now, after the n insertions, the 2–3 tree will be storing $2n$ items, and so has at most $2n-1$ internal nodes. Each node splitting increases the number of internal nodes by one. As there are no decreases in node numbers during this process, there can be at most $2n-1$ nodes splittings. (This is actually an overestimate.)

To see the bound on the number of internal nodes, we argue as follows. Let the depth of the 2–3 tree storing $2n$ items be h. There are at most n internal nodes at depth $h-1$, $n/2 = n/2^2$ at depth $h-2$, ..., at most $n/2^{i+1}$ at depth $h-i$, ..., and 1 at depth 0. This is a total of at most $n + n/2 + n/4 + \ldots + 1 = 2n - 1$ internal nodes.