Q1.

```
merge(A, B):
A: AVL tree
B: AVL tree
where A < B
returns merged AVL tree
    // Find and delete smallest element from B. Complexity O(B.height)
    // Modified B becomes the right child of this element on line 12
 1. smallestB ← delete(B, smallest(B))
    // Find appropriate node that can become the left child of smallestB.
Complexity O(A.height)
 2. node ← A
 3. height ← A.height
 4. while height > B.height + 1 do
 5.     if node.balance = -1 then
 6.         height ← height - 2
 7.     else
 8.         height ← height - 1
 9.     node ← node.right
    // Find parent of the appropriate node
10. nodeParent ← parent(A, node)
11. smallestB.left ← node
12. smallestB.right ← B
    // Make smallestB the right child of the parent node
13. nodeParent.right ← smallestB
    // Fix nodeParent, from M4 slide 19. Complexity O(nodeParent.height)
14. fix(nodeParent)
15. return nodeParent
```

On line 11, we assign a subset of A as the left child of an element from B. This works as `A < B`.

On line 12, we assign the modified B as the right child of the smallest element from B. This works as `smallest element < modified B`.

On line 13, we assign the smallest element as the right child of the parent node from line 11 assignment. This works as `right child (previous right child + B) > parent node`.

On line 14, we fix any imbalances on the parent node.

The complexity of the algorithm is `O(max(A.height, B.height))`.