Q 3.1)

Since pre-order traversal has root->left->right policy, the first element of the array would always be the root of the tree.

By recreating a binary search tree from given pre-order traversal array then we could simply compute post-order traversal from the created tree using normal post-order traversal.

Using the first element of the array as a root and current node and iterate through the array to construct the tree as the next element. If the new element is less than current node's value then become its left child and current node. Otherwise, traversing back to its parent if the element is greater than the parent's value. Do that until the element is no greater than the parent then become that node's right child and current node (by having every node has a pointer to their parents, this would take constant time). Doing so traversing each node at most 3 times causing it to be $O(3n) \approx O(n)$.

After we have the tree then we do normal post-order traversal by recursively traversing to the left subtree then to the right subtree and end up at the root of the tree into an empty array. This would take $O(n)$ since we iterate through each node once.

Overall time complexity is $O(n) + O(n) = O(n)$ since both recreating tree and post-order traversal take $O(n)$ as mentioned above.

Q 3.2)

Since pre-order traversal has root->left->right policy, the first element of the array would always be the root of the tree. And in-order traversal has left->root->right policy, that means we could split that into 2 subtrees from knowing the position of the root.

By recreating a tree from given pre-order and in-order traversal arrays then we could simply compute post-order traversal from the created tree using normal post-order traversal.

Starting with the root from the first element of pre-order array. Then to find it's left and right subtrees, we would find the index of this root from in-order array (every element from the left of the index would be left subtree, would be right subtree otherwise) and recursively build its left and right subtrees from in-order array. Then using the next element of pre-order array as the new root and do the same. This takes $O(n)$ to search for the index of the root in the in-order array.

After we have the tree then we do normal post-order traversal by recursively traversing to the left subtree then to the right subtree and end up at the root of the tree into an empty array. This would take $O(n)$ since we iterate through each node once.

Overall time complexity is $O(n) + O(n) = O(n)$ since both recreating tree and post-order traversal take $O(n)$ as mentioned above and $O(n)$ beats $O(n \log n)$.