This problem can be solved using dynamic programming in linear time because when we know optimal solution for each subtree of the binary tree we only have to traverse the whole tree to get the desired order of vertices.

It's obvious that the optimal memory usage for tree or subtree made only out of one vertex is one. This means we can always get the optimal memory usage when the subtree consists of an leaf.

Thus, we must consider only two cases for each inner vertex – when the vertex is a parent of only one subtree and when the vertex is a parent of two subtrees.

For the first case, the smallest amount of used memory is always 2 because we have to still hold the result of the subtree after processing it. However, when the optimal memory usage m is bigger than 2, the optimal memory usage is m. As a result, the optimal memory consumption of the whole subtree is $\min(2, m)$.

In the second case, the smallest amount of used memory is always 3. However, now the optimal memory consumption also depends on equality of the optimal memory usage of each tree m, n. If $m = n$, then the optimal memory usage of the whole subtree is $m + 1$ because after processing one of the subtrees, we have to hold its result in the memory. On the other hand, if $m \neq n$, then the optimal memory usage of the whole subtree is $\max(m, n)$ since after processing the more memory intensive subtree we are able to process the rest of the whole subtree without increasing the memory usage above $\max(m, n)$. Hence, the optimal memory consumption of the whole subtree is $\min(3, m + 1)$ when $m = n$, else it is $\min(3, \max(m, n))$.

Now we can recursively calculate the optimal memory usage for each subtree in the tree. Then when traversing the tree we have to traverse the more memory intensive subtrees first and add the visited vertices into the computation plan. This will take $\mathcal{O}(n)$ time and use $\mathcal{O}(n)$ space.