[Q1]

Soluion:

[Algorithm]

We can use dynamic programming to solve the problem in $O(n)$ time. First pick any (DFS). Store the sorted nodes in array N. Due to the definition of DFS, a parent node appears earlier than all of its children in N.

For each node v, define $A(v)$, the best cost of a placement in the subtree rooted at v if v is included, and $B(v)$, the best cost of a placement in the subtree rooted at v if v is not included. The following recursion equations can be developed for $A()$ and $B()$:

If v is a leaf, then $A(v) = p_v$ and $B(v) = 0$.

If v is not a leaf, then

$$A(v) = p_v + \sum_{u \in v.children} B(u)$$

$$B(v) = \sum_{u \in v.children} \max(A(u), B(u))$$

For each node v in N, in the reverse order, compute $A(v)$ and $B(v)$. Finally the $\max(A(u_0), B(u_0))$ is the maximum profit.

The placement achieving this maximum profit can be derived by recursively comparing $A()$ and $B()$ starting from the root. The root u_0 should be included if $A(u_0) > B(u_0)$ and excluded otherwise. If u_0 is excluded, we go to all of u_0's children and repeat the step; if u_0 is included, we go to all of u_0's grandchildren and repeat the step. This algorithm outputs an optimal placement by making one pass of the tree.

[Correctness]

In the base case where v is a leaf node, the algorithm outputs the optimal placement which is to include the node.

In an optimal placement, a node v is either included, which removes all its children, or not, which adds no constraints. By induction, if all the children of v have correct $A()$ and $B()$ values, then $A(v)$ and $B(v)$ will also be correct and the maximum profit at v is derived. Since the array N is sorted using DFS and processed in the reverse order, child nodes are guaranteed to be processed before their parents.

[Timing Analysis]

Sorting all nodes using DFS takes $O(n)$ time. The time to compute $A(v)$ and $B(v)$, given the values for the children, is proportional to $degree(v)$. So the total time is the sum of all the degrees of all the nodes, which is $O(|E|)$ where $|E|$ is the total number of edges. For a tree structure, $|E| = n - 1$ so the time for finding all $A()$ and $B()$ is $O(n)$. Finally, using the derived $A()$ and $B()$ to find the optimal placement visits each node once and thus is $O(n)$.

Overall, the algorithm has $O(n)$ complexity.