W2.

a)

```
TOTAL(T, n)
INPUT: A tree T and the ID of a node in the tree n.
OUTPUT: The sum of the weights of all nodes in the subtree rooted at the node with
ID n
1 Sum <- NODE_WEIGHT(T, n)
2 if IS_LEAF(T, n)
3         return Sum
4 else
5         Child_List <- CHILDREN(T, n)
6         for each Child in Child_List
7                 Sum <- Sum + TOTAL(T, Child)
8         return Sum
```

b) The pseudocode above illustrates a divide and conquer algorithm for calculating the sum of a subtree as it follows the three steps: divide, conquer, and combine. The algorithm starts by dividing the instance into a smaller instance by isolating the children of a node (Line 5). It then conquers the smaller instance by summing the weights of each of those children in a for loop (Lines 6-7). Lastly, it combines this sum with that of each of the children's children until a leaf is reached (Line 7).

c) The base case of the algorithm is represented by Line 2 and 3 where we verify whether *n* is a leaf. This can be executed in constant time where the value *Sum* is returned. Thus, $T(1) \in \theta(1)$.

For the recursive case, the cost can be viewed as the sum of the divide step, the conquer step, and the combine step.

- (Line 5) Dividing the instance into 3 subtrees excluding the root $\frac{n-1}{3}$
- (Line 7) The conquer and combine step consists of adding the weights of the subtree's nodes until a leaf is reached. The cost of calculating the weight of a specific node (conquer step) is constant.

Since there are three subtrees per node, the cost of the recursive case is $T(\frac{n-1}{3})$.

Thus the recurrence of the algorithm above is $T(n) \leq 3T\left(\frac{n-1}{3}\right) + O(1)$.