**Complexity of recursion on trees**

Consider the following generic recursion algorithm on trees

```plaintext
function Foo(T)
    if T == NULL then
        return something
    end if
    do_smth1
    for C in T.children do
        call Foo(C) and do_smth2
    end for
    do_smth3
end function
```

**Statement.** Let tree $T$ consist of $n$ nodes. Then

1. $\text{Foo}(T)$ will invoke $n - 1$ recursive calls;

2. $\text{do_smth1}$ and $\text{do_smth3}$ will be performed $n$ times;

3. $\text{do_smth2}$ will be performed $n - 1$ times.

**Proof.**

1. For each non-root node $C$, there will be exactly one recursive call $\text{Foo}(C)$ invoked by the unique parent of $C$. There are $n - 1$ non-root nodes.

2. $\text{do_smth1}$ and $\text{do_smth3}$ will be called exactly once for each call of $\text{Foo}$. There will be one initial call and, as we have shown, $n - 1$ recursive calls, so the total count is $n$.

3. For every node $C$, $\text{do_smth2}$ will be performed in $\text{Foo}(C)$ the number of times equal to the number of children of $C$. Hence

$$
\text{the number of do_smth2} = \sum_{C \text{ in the tree}} \text{the number of children of } C
$$

Since every non-root node has exactly one parent, every non-root node is counted in the right-hand side of the above equality exactly once. Thus

$$
\sum_{C \text{ in the tree}} \text{the number of children of } C = \text{the number of non-root nodes} = n - 1.
$$