The way `partial-tree` works is that it will divide the given $n - 1$ in two (`left-size`), and then it makes a recursive call to form a balanced tree for the left branch (`left-result`). The root node will be the first leftover element of the recurisve left branch call (`this-entry`). The right branch recursive call will be the remaining half (`right-size`) of $n - 1$. The rest of the leftovers from the first recursive call (`left-result`), are the elements for the second one (`right-result`). Whatever is left over from that second call will be the list elements not included in the tree (`remaining-elts`). The `car` of the result will be the balanced binary tree with a root node of `this-entry` and left and right branches taken as the `car` of the recursive calls.

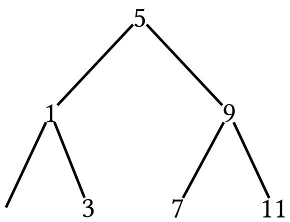
For the list `(1 3 5 7 9 11)`, `list->tree` will form the below tree. The first call to `partial-tree` is `(partial-tree '(1 3 5 7 9 11) 6)`.

The left recursive call will be `(partial-tree '(1 3 5 7 9 11) 2)`. This itself spawns a `left-result` of `(() (1 3 5 7 9 11))` because `left-size` is 0. The entry node will be `1` and the right branch will be the recursive call `(partial-tree '(3 5 7 9 11) 1)`. This itself yields a `left-result` of `(() (3 5 7 9 11))`, an entry of `3`, and a `right-result` of `(() (5 7 9 11))`. The remaining elements are thus `(5 7 9 11)`. The entire left recursive call referenced at the beginning of this paragraph thus yields `((1 () (3 () ())) 5 7 9 11)`.

The root node will be thus be `5`, the `car` of `non-left-elts` which are `(5 7 9 11)`.

The right recursive call will thus be `(partial-tree '(7 9 11) 3)`. The `left-result` will be the result of the recursive call `(partial-tree '(7 9 11) 1)` which itself has a `left-result` of `(() (7 9 11))`, entry node of `7`, and `right-result` of `(() (9 11)`. Therefore, the entry node of the previous recursive call will be `9`, and its `right-result` will be the recursive call `(partial-tree '(11) 1)`. This itself has a `left-result` of `(() (11))`, an entry node of `11`, and a `right-result` of `(() ())`. Therefore, there are no remaining elements and the entire right branch (first recursive call mentioned in this paragraph) will yield the pair `((9 (7 () ()) (11 () ())) ())`.

Thus the original call to `partial-tree` will return the pair `((5 (1 () (3 () ())) (9 (7 () ()) (11 () ()))) ())`, and `list->tree` will return the tree `(5 (1 () (3 () ())) (9 (7 () ()) (11 () ())))`, which is visualized below.



The order of growth in the number of steps required by `list->tree` to convert a list of n elements is $\Theta(n)$ because each node is traversed only once. Either a node is a leaf and its further recursive `partial-tree` calls immediately return the remaining elements, or it will spawn one or two recursive calls depending on how whether it has a left or right branch or both. This results in the number of meaningful (to our analysis) `partial-tree` calls being equal to n. Doubling the size of n might require only one more `partial-tree` call to be kept track of (when thinking of space), but still there will be twice as many `partial-tree` calls made, as there is one for each n.