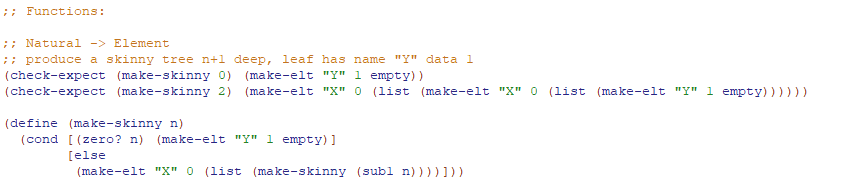
Local expressions can be used to avoid recomputing results, and in recursive programs this can have significant (even exponential) effects on program performance.

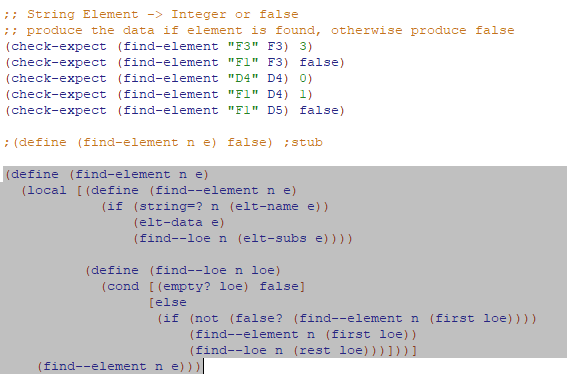
Exponential growth in performance

* Important category of performance issue that you need to deal with
* Local can help us deal with this performance issue

Let’s try for this 2 functions



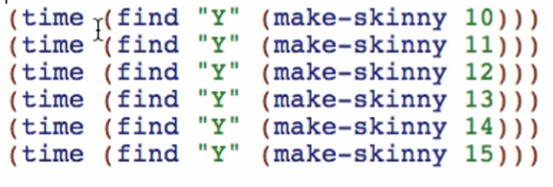
* Makes a skinny tree (1 lined tree with the n as the number of “X” as a parent)



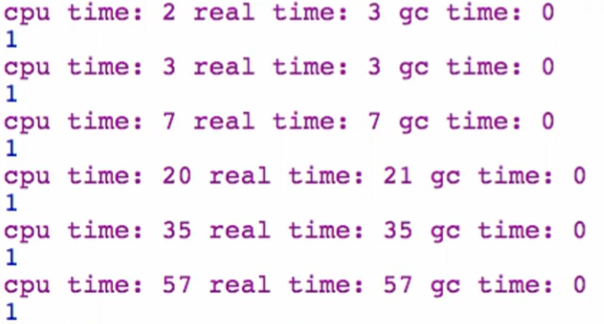
* Find the element in a tree

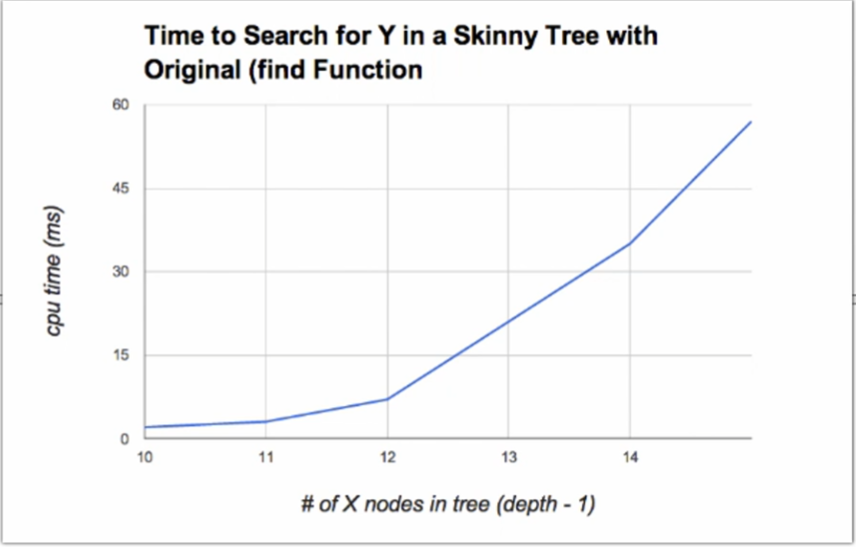
Using “time” primitive

* More like an if rather than a function
* Evaluates operand and returns the time it took to evaluate them



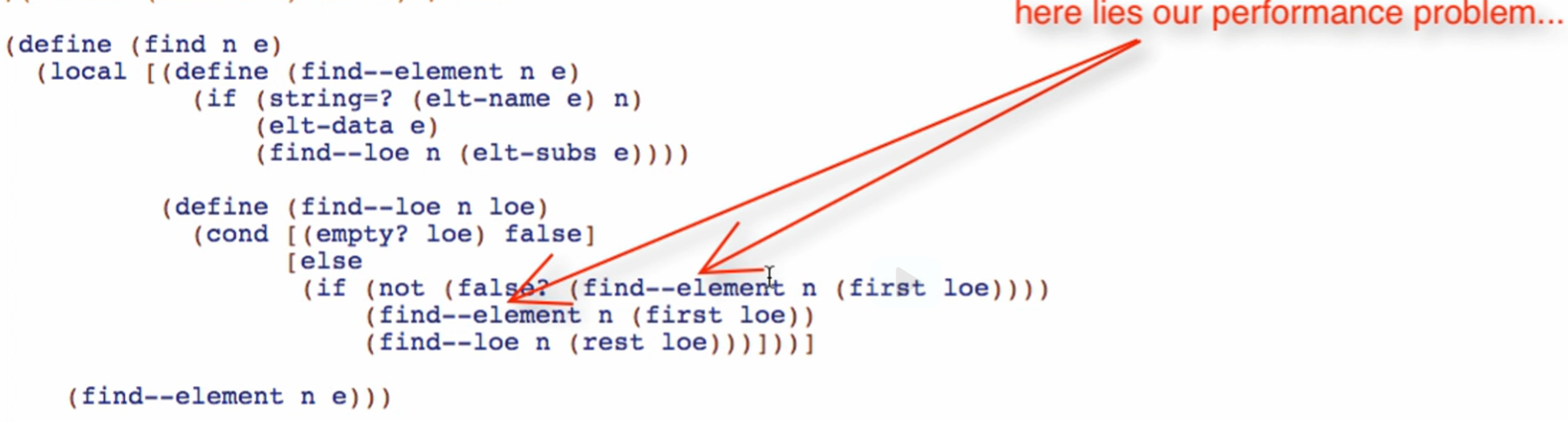
Results:

- time in milliseconds



* This is a bad graph and a serious performance problem! As we make the tree deeper, it increases the time it takes to find the element
* 1 deeper = twice longer to run

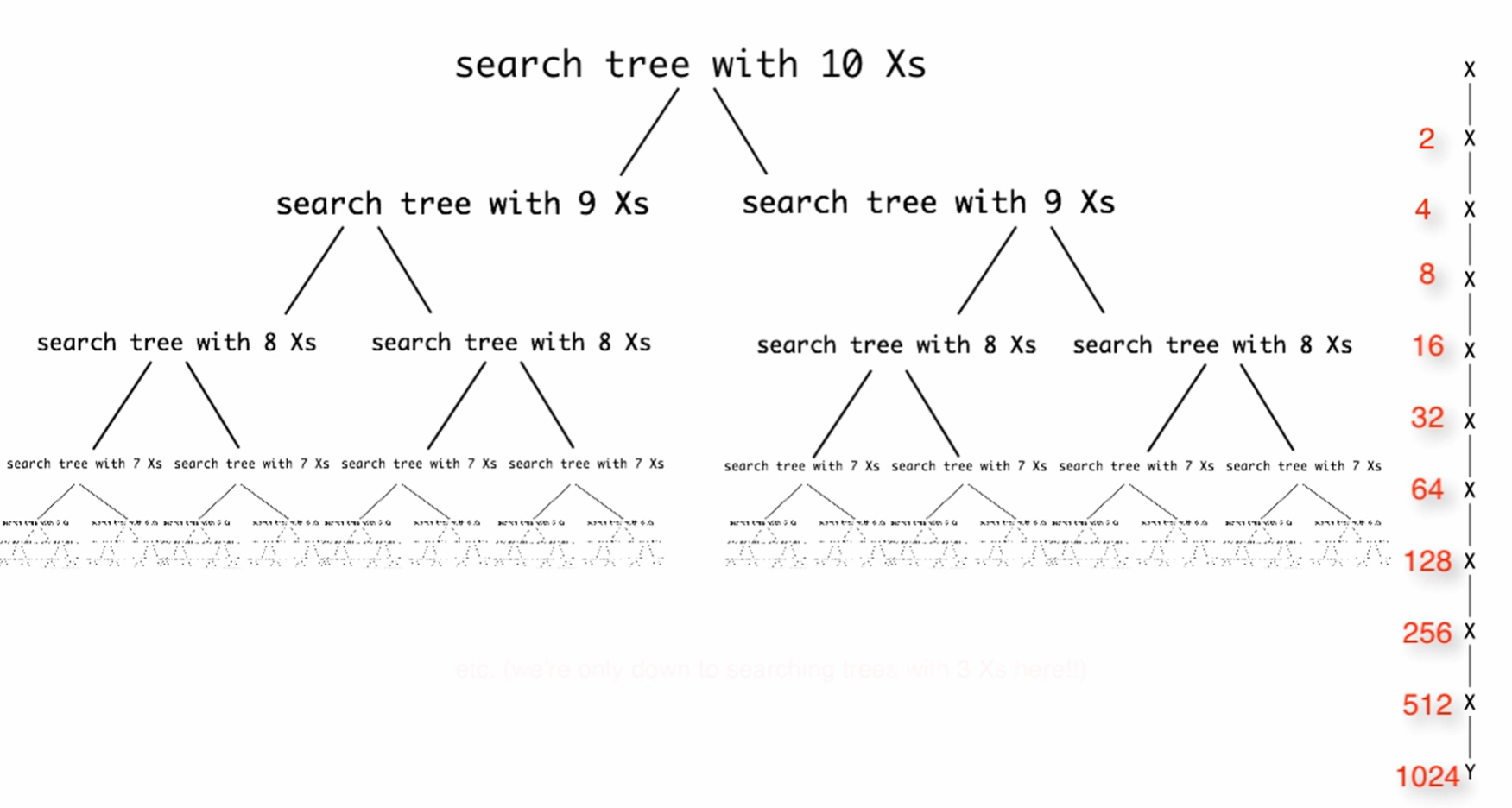
Why is it happening in the find function:



* It searches in that branch twice here!
* At the very top we search the next level down twice then twice again under that!

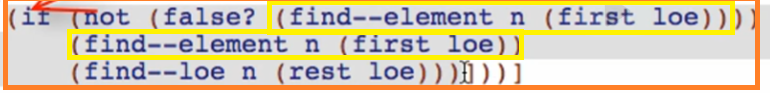
 -> 

Overview:

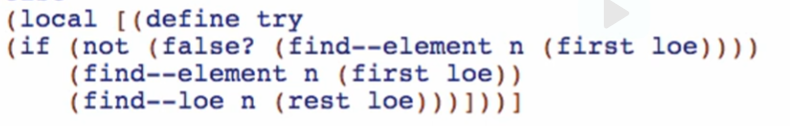


So how do we fix that?

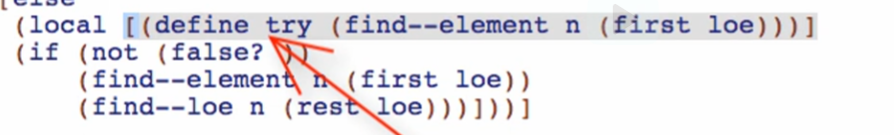
1. Find the nearest enclosing expression containing the multiple computations



1. Wrap it in local and define a function

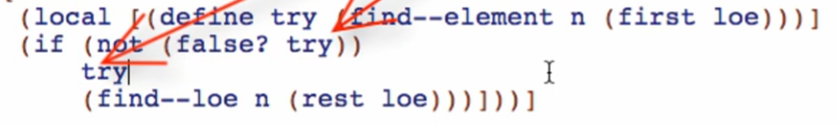


1. Take the value that is being recomputed



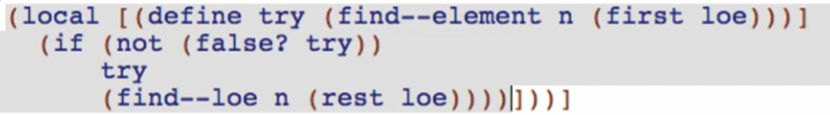
* + We define the value that is being recomputed thus it will only be **evaluated/computed in that definition**

1. Replace each expression in the body

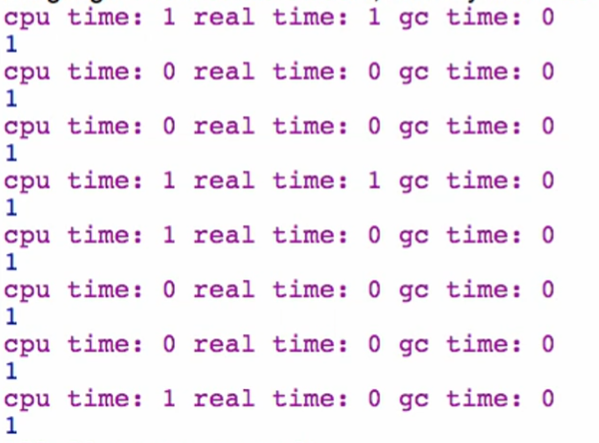


* + We then call that computed value in the code body (without recomputing it since it is only computed in the definition

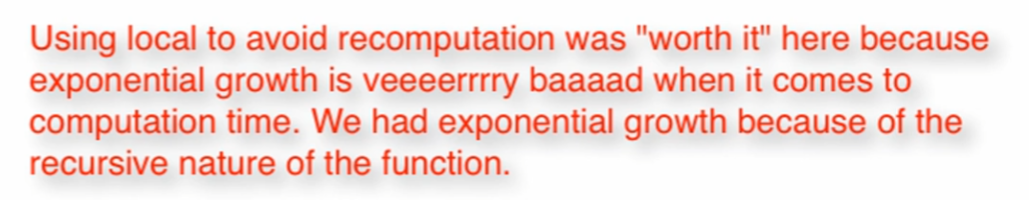
1. Close the local



Try running again:



There are no exponential growth already!



Not all instances that you use local is worth it especially for functions with non-recursive calls