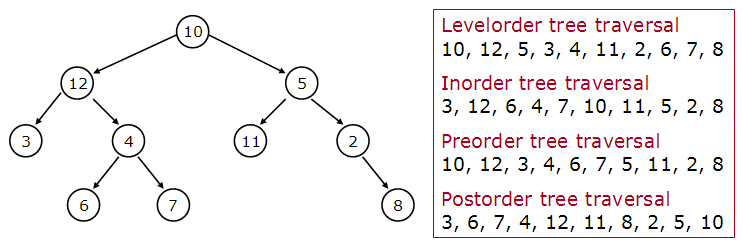
PROS:

**Recursion can reduce time complexity**. This was somewhat counter-intuitive to me since in my experience, recursion sometimes increased the time it took for a function to complete the task. An example of this is calculating fibonacci numbers. If you calculate the fibonacci sequence up to a number *n*using recursion rather than iteration, the time to complete the task when compared to that of the iterative approach was much greater. However, if you ***memoize*** the result (aka save the value of each calculation for further use in the recursive call) you can in fact reduce the time complexity (read a great answer response for more information about memoization [here](https://cs.stackexchange.com/questions/13055/time-complexity-and-space-complexity-in-recursive-algorithm)).

**Recursion adds clarity and reduces the time needed to write and debug code.** This one is valid to a point. If you know your input into a function is going to be small, then recursion is certainly a good choice if you want to de-clutter your code. If your input is sufficiently large however, the sacrifice of speed and memory for the sake of clarity becomes much less attractive and functional.

**Recursion is better at tree traversal.** This one is a little more advanced. An *extremely* simplified version of what this means is as follows: A tree is a collection objects that are linked to one another (imagine leaves on a tree connected by branches that are in turn connected to other branches all the way to the roots). One of the more efficient ways to traverse these trees when looking for a specific leaf (or *node*) is by recursively following a single branch until the end of that branch until you find the value you are looking for. Again, this is extremely abstracted and simplified for what is actually happening and I urge you to look further into what is actually happening in tree traversal.



Example of tree traversal

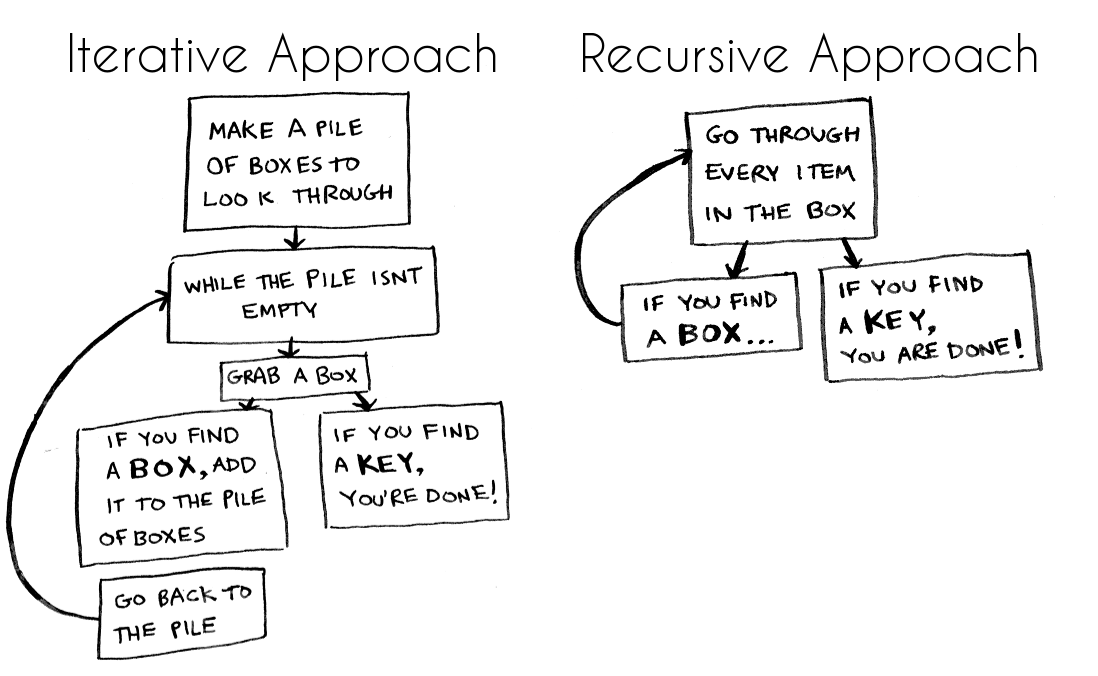
Recursion in the above tree diagram would be beneficial when used on preorder tree traversal.

CONS:

**Recursion uses more memory.** Because the function has to add to the stack with each recursive call and keep the values there until the call is finished, the memory allocation is greater than that of an iterative function.

**Recursion can be slow.**If not implemented correctly (as stated above with memoization) it can be much slower than iteration. It is actually pretty difficult to write a recursive function where the speed and memory will be less than that of an iterative function completing the same task. The reason that recursion is slow is that it requires the allocation of a new stack frame.

I know I mentioned a lot about recursion vs iteration above, so lets look more into that.



Iteration vs recursion, courtesy of [freecodecamp](https://medium.freecodecamp.org/how-recursion-works-explained-with-flowcharts-and-a-video-de61f40cb7f9" \t "_blank)

Both iteration and recursion are repetitive processes that repeat a certain process until a certain condition is met. They are both used in programming to complete tasks where a task has to be repeated in order to solve the problem.

**Iteration:** A function repeats a defined process until a condition fails. This is usually done through a loop, such as a for or while loop with a counter and comparative statement making up the condition that will fail. An infinite loop for iteration occurs when the condition never fails.

**Recursion:** Instead of executing a specific process within the function, the function calls itself repeatedly until a certain condition is met (this condition being the base case). The base case is explicitly stated to return a specific value when a certain condition is met. An infinite recursive loop occurs when the function does not reduce its input in a way that will converge on the base case.