



C01 Recursion

Recursive data structures
Recursive algorithms



PAUL
NOTH



Dröste®

HOLLAND



COCOA

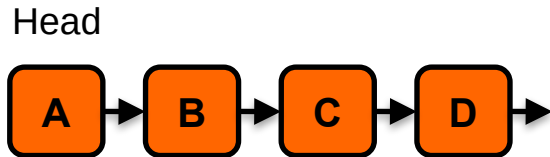
NET WT 250g e 8.8 oz.

Kosher ①

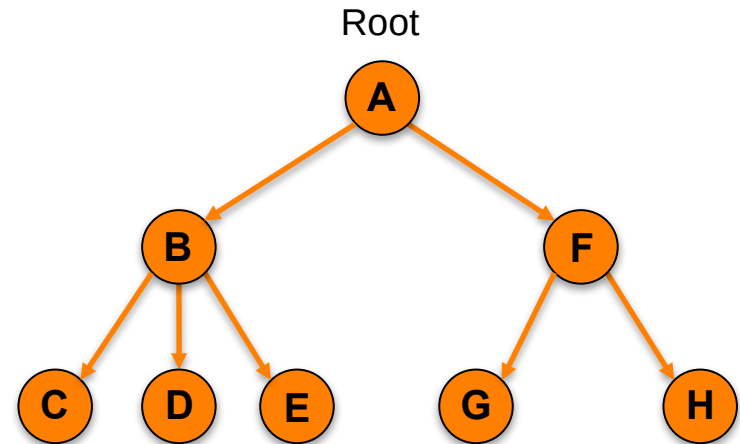


Recursive Data Structures

A recursive data structure is comprised of components that reference other components of the same type.



Linked list



Tree

Recursive Algorithms

A recursive algorithm references itself.

A recursive algorithm is comprised of:

- Base case(s) that terminate the recursion
- Recursive call(s) that reduces towards the base case(s)

Example: Fibonacci Sequence

$\text{fib}(0) = 0$ (base case)

$\text{fib}(1) = 1$ (base case)

$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ (for $n \geq 2$)



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377...

Example: Binary Search

Ordered list and a target value to find.

[1, 4, 5, 7, 9, 11, 15, 20, 25]	find 11	
[1, 4, 5, 7, 9 , 11, 15, 20, 25]	9 > 11?	right half
[9, 11, 15 , 20, 25]	15 > 11?	left half
[9 , 11]	9 > 11?	right half
[11]		

How does this compare to linear search?

What might the base case(s) be?

Example: Mergesort (von Neumann, 1945)

Sort a list

- List of size 1 (base case)
 - Already sorted
- List of size > 1
 - Split into two sub lists
 - Sort each sub list (recursion)
 - Merge the two sorted sub lists into one sorted list (by iteratively picking the lower of the two least elements)

38	27	43	3	9	82	10
----	----	----	---	---	----	----



Recursion

- A recursive method can always be rewritten into an iterative one and vice-versa (consequence of *Church-Turing thesis*).
- When to use **recursion** vs when to use **iteration** (**for** and **while** loops)?
 - The problem at hand might be more naturally written and read in one form (once you understand recursion!).
 - Converting between approaches not always straight-forward.



Recursion and Java

- Overhead of calling calling methods often higher than iterating
- *Stack overflow* on larger problems
- Compilers in many other languages perform *tail-call elimination* for certain forms of recursion – Java doesn't
- More functional languages (scheme, lisp, ocaml, haskell, f#, scala) make recursion more convenient
- Situations where recursion is *best* is more limited in Java – but important cases still exist!