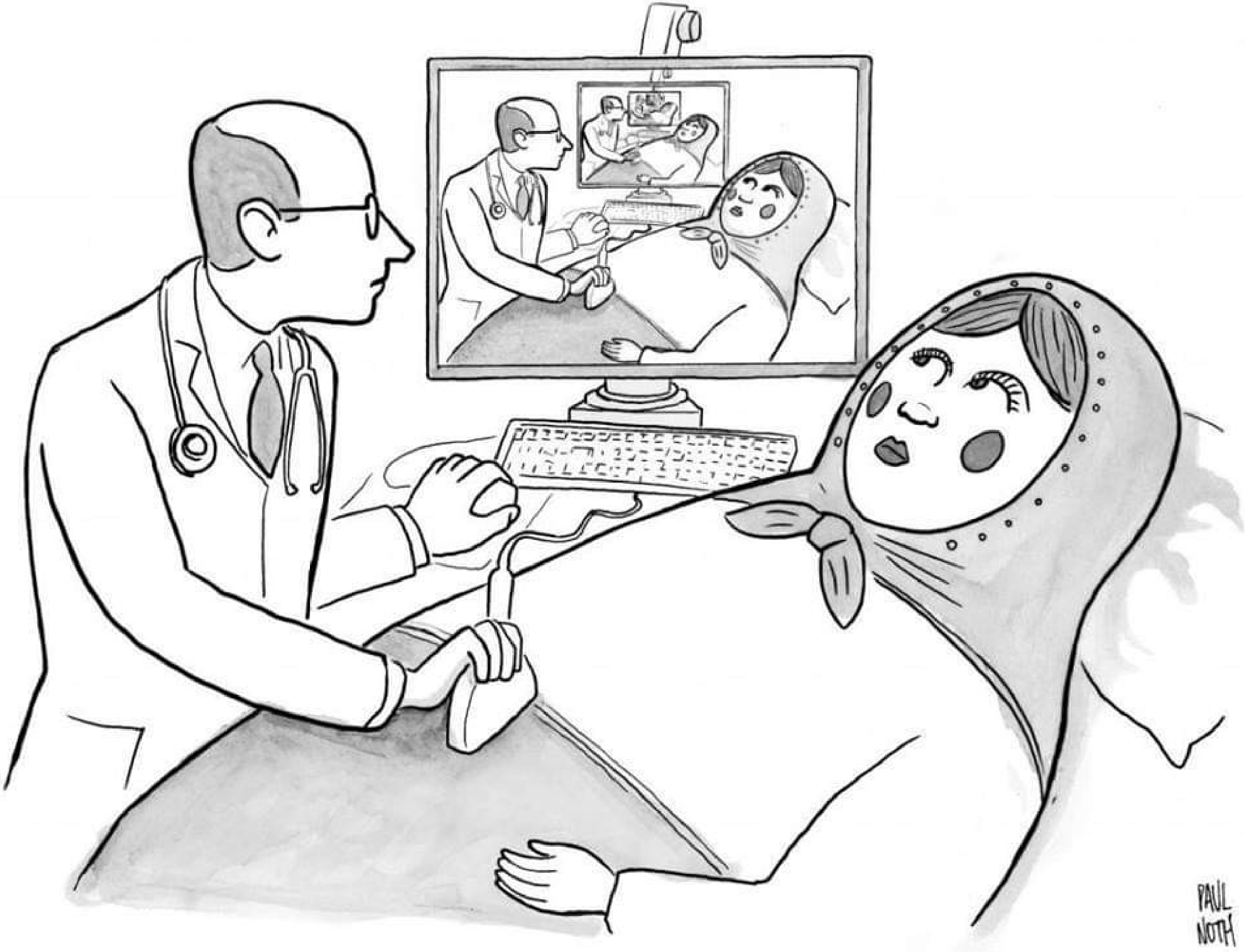
C01 Recursion

Recursive data structures
Recursive algorithms

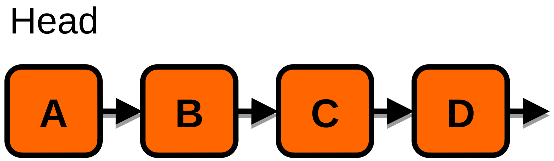
ANU – School of Computing – Structured Programming 1110 / 1140 / 6710


PAUL NOTH

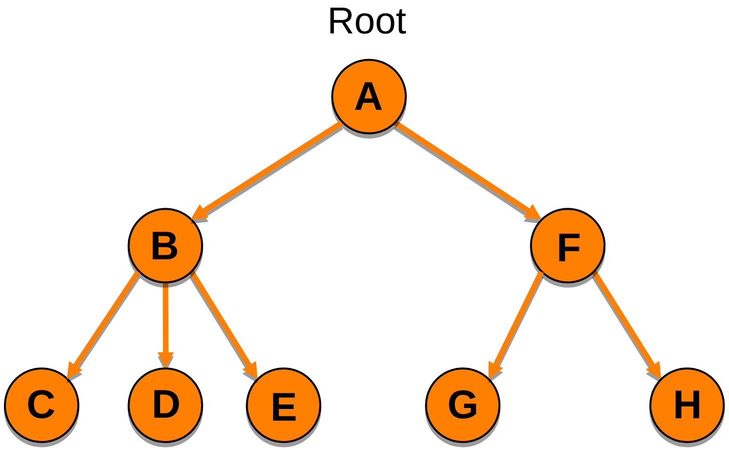
HAARLEM HOLLAND
Droste®
HOLLAND
Droste
cacao
cocoa
NET WT 250g ℮ / 8.8 oz.
Kosher Ⓤ

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 reduces a computational problem to one or more smaller instances of the same problem, and composes the solution from their solutions.

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

fib(0) = 0 (base case)

fib(1) = 1 (base case)

fib(n) = fib(n-1) + fib(n-2) (for n ≥ 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]
```

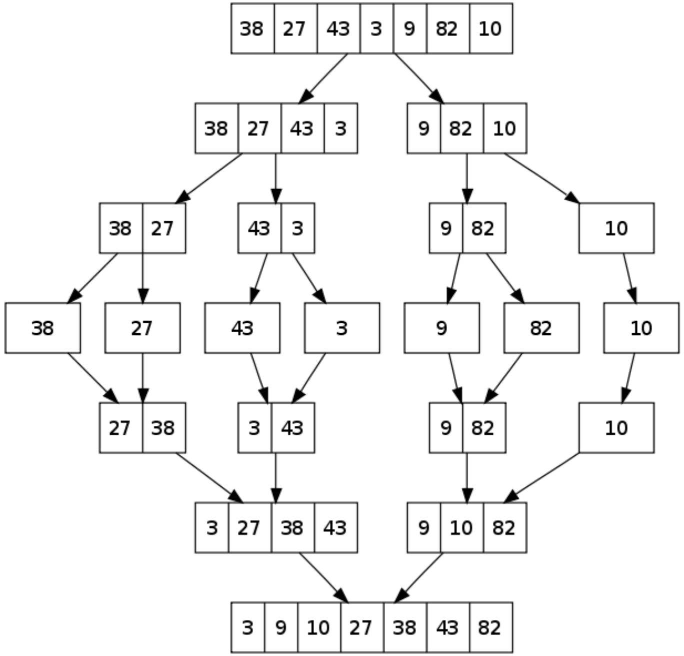
How does this compare to linear search?

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)



Animation: Visualizing Algorithms, Mike Bostock, bost.ocks.org/mike/algorithms

Recursion

- A recursive method (function) calls itself: this works because of the *call stack*.
- 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 straightforward.

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* are more limited in Java – but important cases still exist!