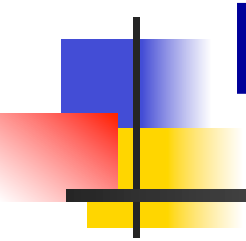
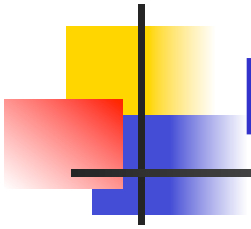


Lecture 3

Chapter 4. Induction and Recursion

4.3 Recursive Definitions and Structural Induction

4.4 Recursive Algorithms



Recursive Algorithms

- Recursive definitions can be used to describe functions and sets as well as *algorithms*.
- A *recursive procedure* is a procedure that invokes itself.
- A *recursive algorithm* is an algorithm that contains a recursive procedure.
- *An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.*

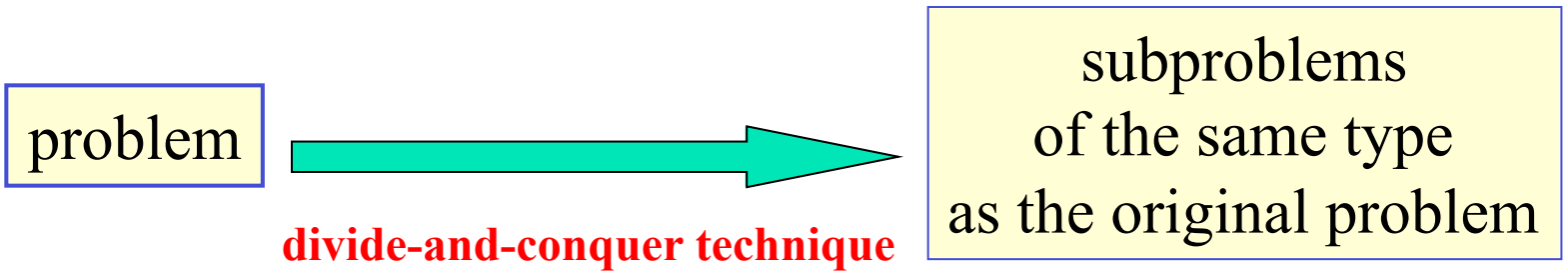
Example

- A procedure to compute a^n.

  procedure *power*($a \neq 0$: real, $n \in$ **N**)

  if $n = 0$ **then return** 1

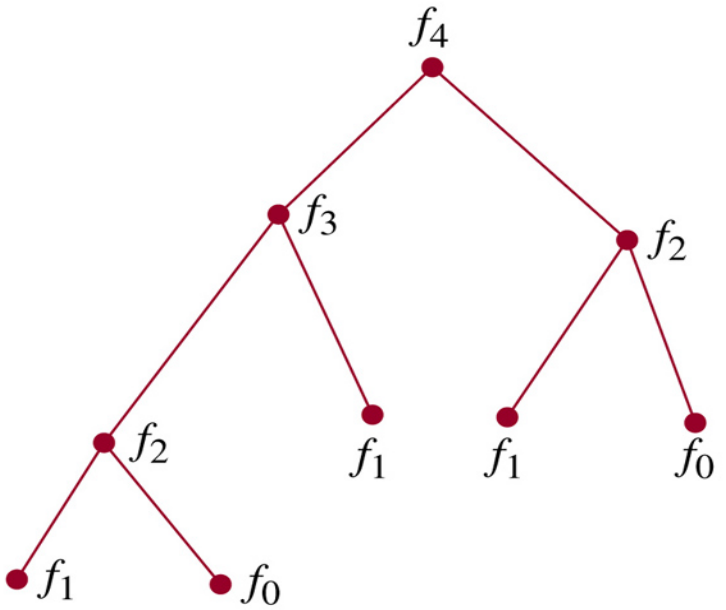
  else return $a \cdot power(a, n-1)$



Recursive Fibonacci Algorithm

procedure *fibonacci*($n \in$ **N**)
  if $n = 0$ **return** 0
  if $n = 1$ **return** 1
  return *fibonacci*($n - 1$) + *fibonacci*($n - 2$)

- Is this an efficient algorithm?
- How many additions are performed?



Iterative Fibonacci Algorithm

```
procedure iterativeFib(n ∈ N)
    if n = 0 then
        return 0
    else begin
        x := 0
        y := 1
        for i := 1 to n – 1 begin
            z := x + y
            x := y
            y := z
        end
    end
    return y    {the nth Fibonacci number}
```

Requires only $n - 1$ additions

Recursive Linear Search

{Finds *x* in series *a* at a location ≥*i* and ≤*j*

procedure *search*

(*a*: series; *i*, *j*: integer; *x*: item to find)

if a_i = *x* **return** *i* {At the right item? Return it!}

if *i* = *j* **return** 0 {No locations in range? Failure!}

return *search*(*a*, *i* +1, *j*, *x*) {Try rest of range}

- Note there is no real advantage to using recursion here over just looping

  for *loc* := *i* to *j*...

  recursion is slower because procedure call costs