Lesson 7:

Mathematical Induction & Recursion

- Recursive Definitions
- Mathematical Induction
- Variations on Induction
- Strong Induction
- Recursive Functions
- Recursive Sets

Chapter 3.3-3.4

Recursive Definitions of Functions

$$f : \mathcal{N} \rightarrow \mathcal{N}$$

$$f(n) = 2^n$$

Can also be defined in terms of f:

$$f(n) = 2 * f(n-1)$$

Note: Needs stopping criteria: $f(0) = 1$

$$g(n) = n!$$

$$g(n) = n * g(n-1)$$

Mathematical Induction

Prove that if $f(n) = 2 * f(n-1)$, $f(0)=1$

then $f(n) = 2^n$

Prove that if $g(n) = n * g(n-1)$, $g(0)=1$

then $g(n) = n!$

Prove that the sum of the first n odd positive integers is n^2.

Prove that $n < 2^n$ for all positive integers n.

Mathematical Induction (אינדוקציה מתימטית)

An important proof technique for problems of this sort.

Used in wide variety of areas:

- Complexity of algorithms
- Program run-times
- Correctness of programs
- Theorems about graphs and trees

Main issue: equivalence exists only needs proof.

Well-Ordering Property (סדור היטב)

Axiom:

The *Well-Ordering Property* ('תכונת 'סדור היטב)
of Natural Numbers (or positive Integers) :
Every nonempty subset of $\mathcal{N}$ has a least member.

Intuitively means that the set $\mathcal{N}$ can be linearly ordered.

Examples: $\{x \mid 0 < x < 10\}$ $x \in \mathcal{N}$

Has a least member: 1

$\{x \mid 0 < x < 10\}$ $x \in \mathcal{R}$

Does NOT have a least member.

$\mathcal{R}$ is not Well-Ordered.

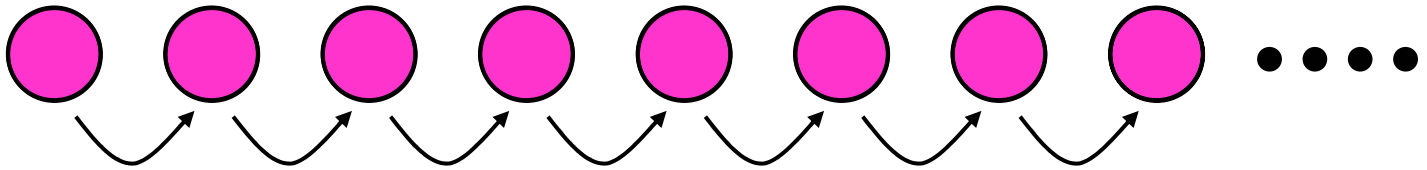
Mathematical Induction (אינדוקציה מתימטית)

Principle of Induction:

To prove a statement on a well-ordered set:

- order the set linearly
- prove specifically for first element
- prove for all elements: if statement is true for an element then is true for following one.

Statement is thus proved for all elements in the set.



Mathematical Induction (אינדוקציה מתימטית)

Induction is used to prove Theorems of the form:

$\forall n\ P(n) \quad n \in \mathcal{Z}^+$

Proving $\forall n\ P(n)$ by induction:

Step 1: *Basis Step* (בסיס האינדוקציה)

prove $P(1) \Leftrightarrow \mathbf{T}$

Step 2: *Inductive Step* (צעד אינדוקטיבי)

prove $\forall k\ P(k) \rightarrow P(k+1) \quad k \in \mathcal{Z}^+$

P(n) is called the *Induction Hypothesis* (הנחת האינדוקציה)

$[\ P(1) \wedge\ \forall n\ (P(n) \rightarrow P(n+1))\ ] \rightarrow \forall n\ P(n) \quad n \in \mathcal{Z}^+$

Mathematical Induction (אינדוקציה מתימטית)

$\forall n\ P(n) \quad n \in \mathcal{Z}^+$ is proven because:

P(1) = **T** Proven in Induction Basis

P(2) = **T** because P(1) = **T** and
P(1)→P(2) = **T** by Induction Step.

P(3) = **T** because P(2) = **T** and
P(2)→P(3) = **T** by Induction Step.

⋮

P(k) = **T** because P(k-1) = **T** and
P(k-1)→P(k) = **T** by Induction Step.

For all $k \in \mathcal{Z}^+$

Mathematical Induction (אינדוקציה מתימטית)

The First Theorem proven by induction :

Francesco Maurolico (1494-1575)

Theorem:

The sum of the first n odd positive integers is n^2.

$1 = 1$ $\quad$ $1 + 3 = 4$ $\quad$ $1 + 3 + 5 = 9$

$1 + 3 + 5 + 7 = 16$ $\quad$ $1 + 3 + 5 + 7 + 8 = 25$

Mathematical Induction (אינדוקציה מתימטית)

Proof: By induction:

P(n) = "The sum of the first n odd integers equals n^2".

1) Basis step: P(1) = **T** because $1 = 1^2$.

2) Inductive Step: prove that for all k $P(k) \rightarrow P(k+1)$

Assume P(k) = **T** and prove P(k+1) = **T**

$1+ 3 + 5 + 7 + ... + (2k-1) = k^2$ (Induction Hypothesis)

$1+ 3 + 5 + 7 + ... + (2k-1) + (2k+1) = k^2 + (2k + 1)$

$= k^2 + 2k + 1 = (k+1)^2$ (math)

Thus P(k+1) = **T**.

Thus $\forall n$ P(n) = **T** for $n \in \mathcal{Z}^+$

Induction Proof for Inequality.

Theorem: $n < 2^n$ for all $n \in \mathcal{Z}^+$.

Proof: By induction:

$P(n) =$ "$n < 2^n$ ".

1) Basis step: $P(1) =$ **T** because $1 < 2^1 = 2$.

2) Inductive Step: prove that for all k $P(k) \rightarrow P(k+1)$

Assume $P(k) =$ **T** and prove $P(k+1) =$ **T**

$k < 2^k$ (Induction Hypothesis)

$k + 1 < 2^k + 1$ (math)

$k + 1 < 2^k + 1 \leq 2^k + 2^k = 2^{k+1}$ (math and that $1 \leq 2^k$)

Thus $P(k+1) =$ **T**.

Thus $\forall n\ P(n) =$ **T** for $n \in \mathcal{Z}^+$

Induction Proofs for $n \geq n_0$

Proof of statements of the form $P(n)$ for $n \geq n_0$, $n \in \mathcal{Z}^+$.

Since $\mathcal{Z}^+$ is well-ordered, any subset of $\mathcal{Z}^+$ is well-ordered.

Thus $\{n \mid n \geq n_0\}$ is well-ordered and induction can be applied.

Proving $P(n)$ for $n \geq n_0$, $n \in \mathcal{Z}^+$ by induction:

Step 1: *Basis Step* (בסיס האינדוקציה)

prove $P(n_0) \Leftrightarrow \mathbf{T}$

Step 2: *Inductive Step* (צעד אינדוקטיבי)

prove $\forall k\ P(k) \rightarrow P(k+1)$ for $k \geq n_0$

Induction Proofs for $n \geq n_0$

Theorem: $2^n < n!$ for $n \geq 4$.

Proof: By induction:

$P(n) =$ "$2^n < n!$"

1) Basis step: $P(4) =$ **T** because $2^4 = 16 < 1*2*3*4 = 24$.

2) Inductive Step: prove that for all $k \geq 4$ $P(k) \rightarrow P(k+1)$

Assume $P(k) =$ **T** and prove $P(k+1) =$ **T**

$2^k < k!$	(Induction Hypothesis)
$2 * 2^k < 2 * k!$	(math)
$2 * 2^k < (k+1) * k!$	(math & $2 \leq (k+1)$ for $k \geq 4$)
$2^{k+1} < (k+1)!$	(math)

Thus $P(k+1) =$ **T** and $\forall n\ P(n) =$ **T** for $n \geq 4$

Induction Proofs on Countable Sets

Proof of statements on countable sets.

Since $\mathcal{Z}^+$ is well-ordered, any subset of $\mathcal{Z}^+$ is well-ordered.

Thus $\{n \mid n \geq n_0\}$ is well-ordered and induction can be applied.

Proving P(n) for $n \geq n_0$, $n \in \mathcal{Z}^+$ by induction:

Step 1: *Basis Step* (בסיס האינדוקציה)

prove $P(n_0) \Leftrightarrow \mathbf{T}$

Step 2: *Inductive Step* (צעד אינדוקטיבי)

prove $\forall k\ P(k) \rightarrow P(k+1)$ for $k \geq n_0$

Induction Proofs on Countable Sets

Theorem: If S is a finite set with n elements then S has 2^n subsets. I.e. P(S) (the Power Set of S) has 2^n elements.

$$|P(S)| = 2^{|S|}$$

Proof: By induction:

P(n) = "Sets of cardinality n have 2^n subsets.

1) Basis step: P(0) = **T** because for n = 0, S = ∅ and P(S) = {∅}, then $|P(S)| = 1 = 2^0 = 2^{|S|}$

2) Inductive Step: prove that $P(k) \rightarrow P(k+1)$

Assume |S| = k and S has 2^k subsets.

Prove that |T| = k+1 has 2^{k+1} subsets.

Induction Proofs on Countable Sets

Proof cont:

2) Inductive Step:

Let T be a set, $|T| = k+1$ $(k \geq 0)$

$T \neq \emptyset$ thus there exists an element $a \in T$.

Define $S = T - \{a\}$ (i.e. $T = S \cup \{a\}$)

For every subset $X \subseteq S$ there are exactly 2 subsets of T:

X and $X \cup \{a\}$

There are 2^k elements in S (Induction Hypothesis)

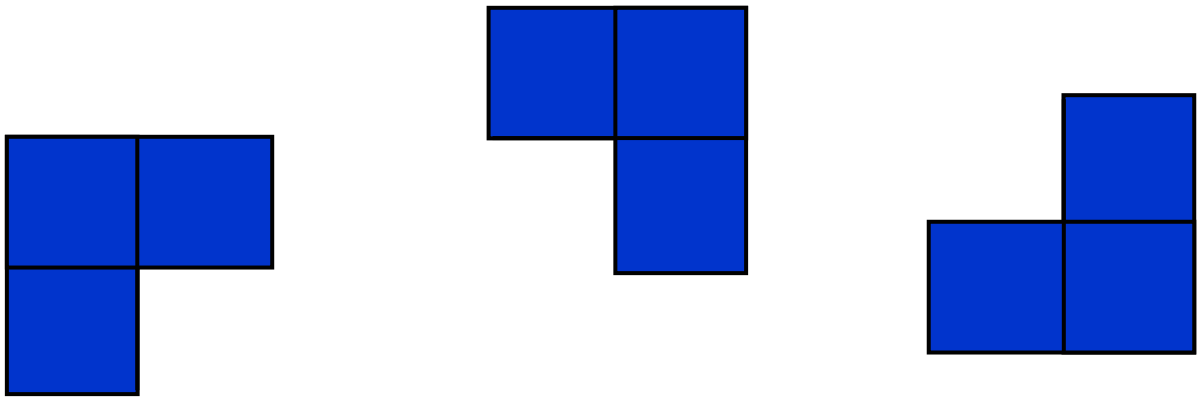
thus there are $2*2^k = 2^{k+1}$ elements in T.

Thus P(k+1) = **T** and $\forall n\ P(n) = \mathbf{T}$

Induction Proofs on Area/Size

Theorem:

Any 2^n x 2^n chessboard ($n \in \mathbb{Z}^+$), with 1 square removed, can be tiled with L-shaped tiles (each tile covers three squares).

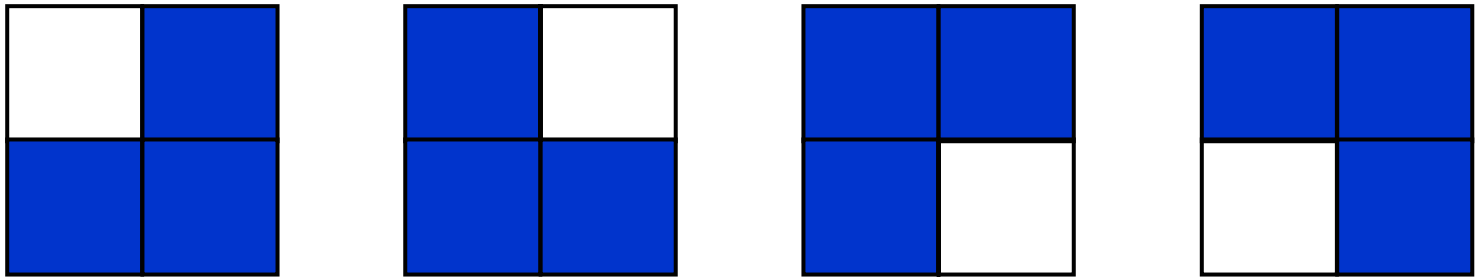


Induction Proofs on Area/Size

Proof: By induction on n:

P(n) = "Any 2^n x 2^n board with one square removed can be tiled with L-tiles".

1) Basis step: P(1) = **T** because any 2 x 2 board with 1 tile removed can be tiled with L-tiles:



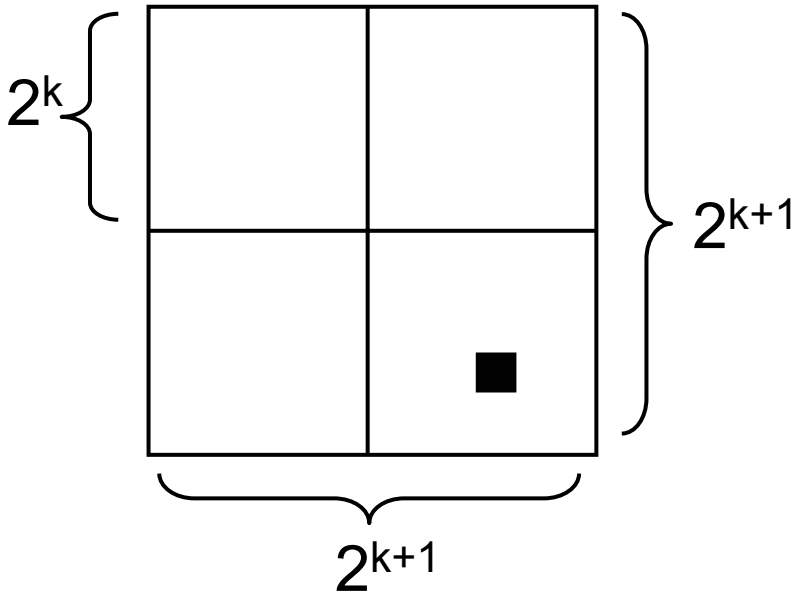
Induction Proofs on Area/Size

Proof cont:

2) Inductive Step: Assume P(k) is True.

Consider a 2^{k+1} x 2^{k+1} chessboard with 1 tile removed.

Split the board into four 2^k x 2^k boards:

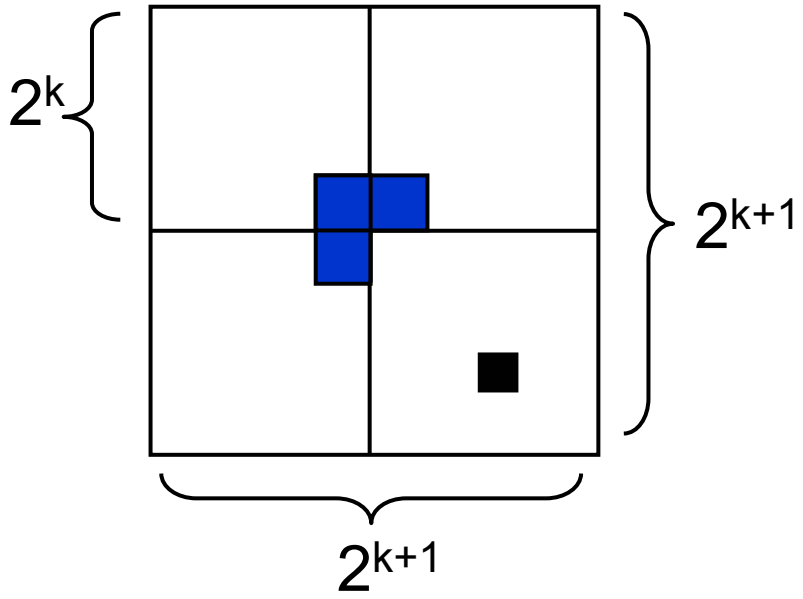


One quarter will have a missing piece . By the Induction Hypothesis, this quarter can be tiled with L-tiles.

Induction Proofs on Area/Size

Proof cont:

2) Inductive Step (cont): From each of the other 3 quarters, remove one corner tile:



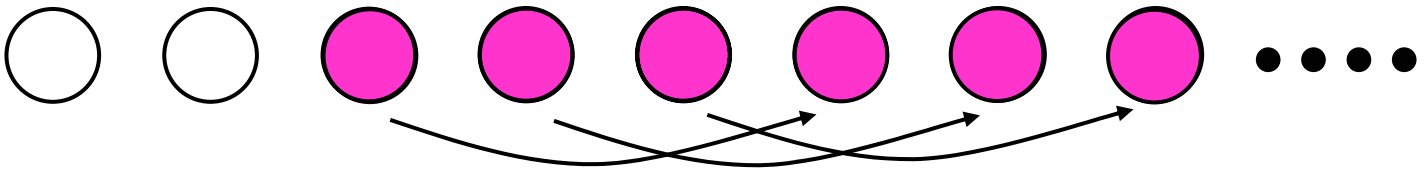
They can now be L-tiled (by the Induction Hypothesis).

The 3 missing squares can be tiled by 1 L-tile.

Thus all 4 quarters can be L-tiled and P(k+1) is True.

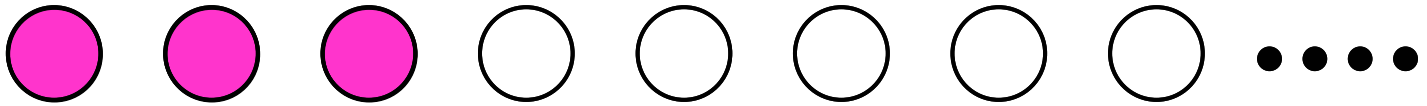
Induction Proofs with k-wise Basis Step

If the Induction Step implies truth of m elements forward,



$$\forall n\,(P(n) \rightarrow P(n+3))$$

then the Basis step must include m element:



$$[\,P(1) \wedge P(2) \wedge P(3) \wedge \forall n\,(P(n) \rightarrow P(n+3))\,] \rightarrow \forall n\,P(n)$$

Induction Proofs with k-wise Basis Step

Theorem: Any integer $n > 7$ is a sum of 3's and 5's.

Proof: By induction:

$P(n)$ = "n is a sum of 3's and 5's".

1) Basis step: $P(8) = T$: $8 = 3 + 5$

$P(9) = T$: $9 = 3 + 3 + 3$

$P(10) = T$: $10 = 5 + 5$

2) Inductive Step: prove that $P(k-2) \rightarrow P(k+1)$

Assume $P(k-2)$ for $k \geq 10$.

$k+1 = (k-2) + 3$

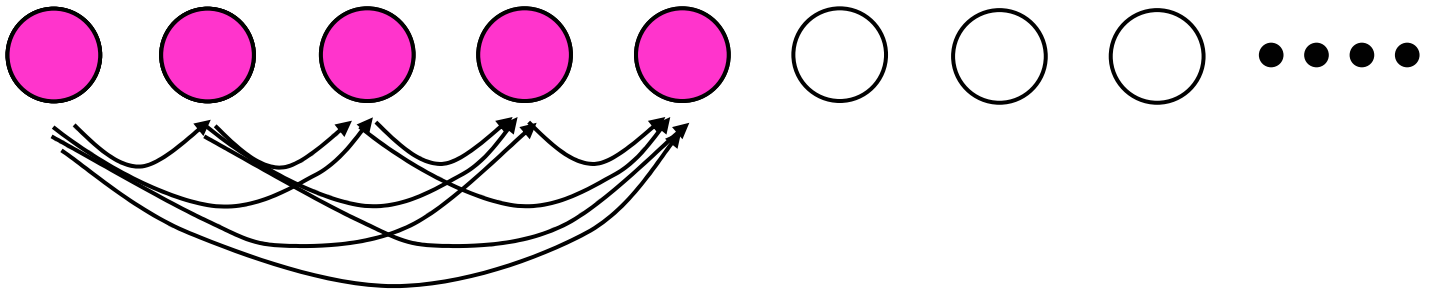
$(k-2)$ is a sum of 3's and 5's (Induction Hypothesis).

Thus $(k-2)+3$ is a sum of 3's and 5's and

$P(k+1) = \mathbf{T}$.

Strong Inductions (אינדוקציה שלמה)

The Induction Step assumes truth of **all** previous elements.



Step 1: *Basis Step* (בסיס האינדוקציה)

prove $P(1) \Leftrightarrow \mathbf{T}$

Step 2: *Inductive Step* (צעד אינדוקטיבי)

prove $\forall k\ (P(1) \wedge P(2) \wedge P(3) \wedge \ldots \wedge P(k)) \rightarrow P(k+1)$

for $k \geq 1$

Strong Inductions (אינדוקציה שלמה)

Theorem: The Fundamental Theorem of Mathematics

(המשפט היסודי של המתימטיקה)

Any integer $n > 1$ is a product of primes.

Proof: By strong induction:

$P(n)$ = "n is a product of primes".

1) Basis step: $P(2) = T$: 2 is a product of 1 prime: 2.
2) Inductive Step: prove $P(k+1)$

   Assume $P(j)=\mathbf{T}$ for **all** $1 \le j \le k$.

   if $(k+1)$ is prime then $P(k+1) = T$.

Strong Inductions (אינדוקציה שלמה)

Proof (cont):

2) Inductive Step (cont):

Else (k+1) is not prime and there exist integers a,b

s.t. (k+1) = a*b and 1 < a,b < k+1

Since P(a)= P(b) = T (Induction Hypothesis).

a and b are products of primes.

Thus a*b=(k+1) is a product of primes.

Thus P(k+1) = **T**.

Recursion (רקורסיה)

Mathematical Induction is strongly related to Recursion.

Definition:

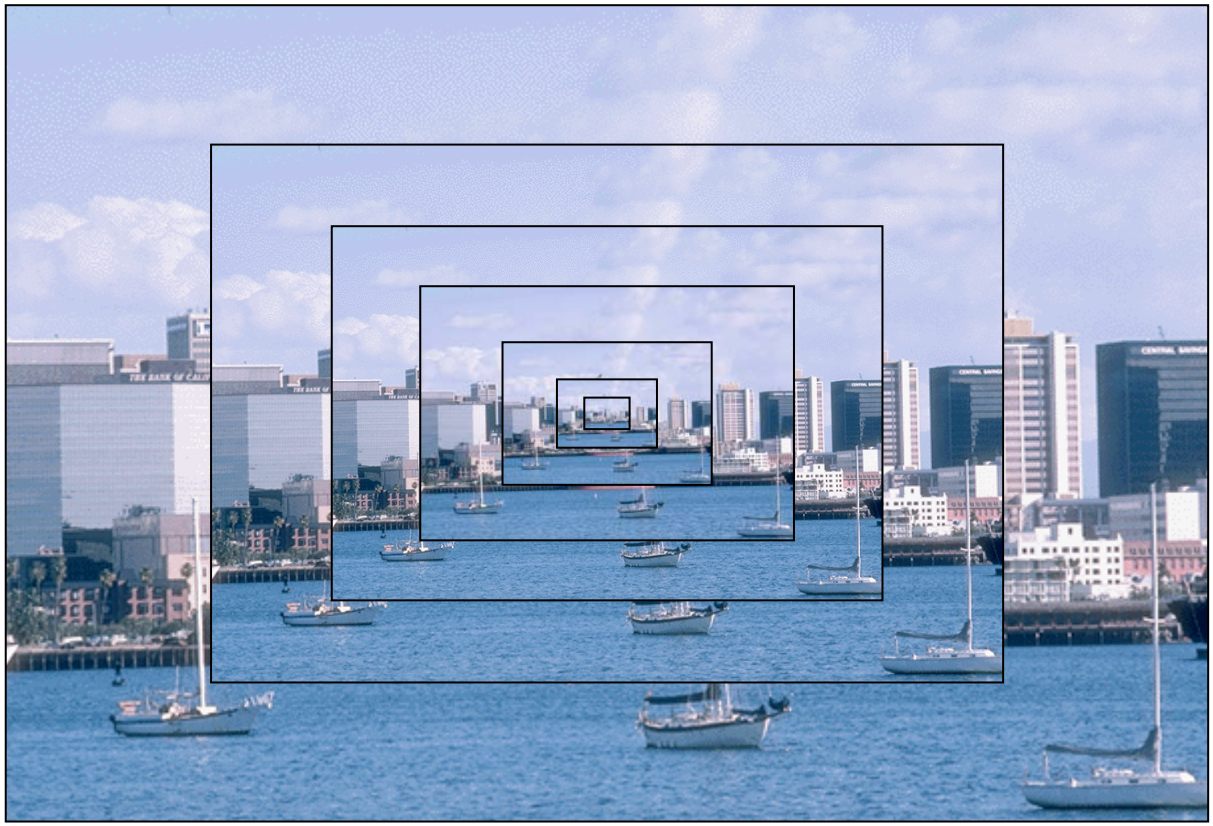
Defining an object in terms of itself is called *Recursion* (רקורסיה).

"הגדרה ציקלית" "הגדרה מעגלית"

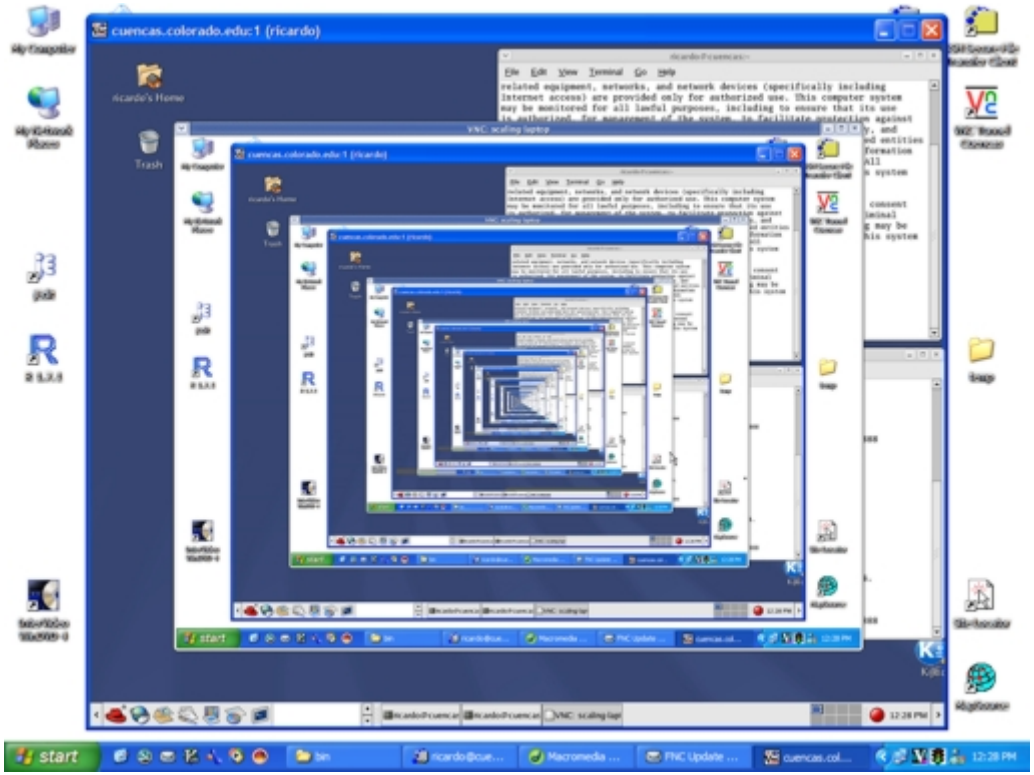
Sets, functions, procedures, algorithms...

Chapter 3.4

Recursive Image



Recursive Windows



Recursion (רקורסיה)

Recursive Definitions always have 2 parts:

1) a Basis Step
2) a Recursive Step

Example:

$f : \mathcal{N} \rightarrow \mathcal{N}$

$f(n) = 2^n$

Recursive Definition:

$f(0) = 1$ Basis Step

$f(n+1) = 2 * f(n)$ Recursive Step

Recursive Computer

What is the Basis step? the Recursive step?

Recursive Functions

A well defined function assigns a single value to every element of the domain.

The Recursive Definition of a function f : $\mathcal{N} \rightarrow \mathcal{N}$:

1) Basis Step - define f(0)
2) Recursive Step - define f(n+1) using f(n)

Recursive Functions

Example:

$f : \mathcal{N} \rightarrow \mathcal{N}$

f(n) = n!

Recursive Definition:

f(0) = 1 Basis Step

f(n+1) = (n+1)*f(n) Recursive Step

f(3) = 3*f(2) = 3*2*f(1) = 3*2*1*f(0) = 3*2*1*1 = 6

Recursive Functions

Example:

$f : \mathcal{N} \rightarrow \mathcal{N}$

f(0) = 3 Basis Step

f(n+1) = 2*f(n)+3 Recursive Step

f(1) = 2*f(0)+3 = 2*3+3 = 9

f(2) = 2*f(1)+3 = 2*9+3 = 21

f(3) = 2*f(2)+3 = 2*21+3 = 45

Recursive Functions

Example:

$$f : \mathcal{N} \rightarrow \mathcal{N}$$

$$f(n) = \sum_{i=0}^{n} a_i$$

Recursive Definition:

$$f(0) = \sum_{i=0}^{0} a_i = a_0$$ Basis Step

$$f(n+1) = \sum_{i=0}^{n+1} a_i = \sum_{i=0}^{n} a_i + a_{n+1} = f(n) + a_{n+1}$$ Recursive Step

Recursive Functions

Are recursively defined functions - well defined?

Prove by induction:

P(n) = "f is defined on n"

Basis Step of induction — Basis Step of Definition

P(0) = **T** since f(0) is defined.

Inductive Step — Recursive Step of Definition

Assume P(k) =**T** and prove P(k+1)

P(k) = **T** so f(k) is defined

f(k+1) = h(f(k)) (h some function) (Recursive definition)

and P(k+1) = **T**.

Recursive Functions

Generalize Recursive Definition of functions analogous to Strong Induction.

The Recursive Definition of a function f : $\mathcal{N} \rightarrow \mathcal{N}$:

1) Basis Step - define f(0)
2) Recursive Step - define f(n+1) using the values f(k) for k ≤ n

Fibonacci Numbers

Example: The Fibonacci Numbers (מספרי פיבונצ'י)

Leonardo de Pisa (1180 - 1228)

= Fibonacci

Wrote the book “Liber Abaci” in which Arabic numerals and algorithms were introduced.

Fibonacci Numbers

The Rabbit problem

2 rabbits are placed on a secluded island

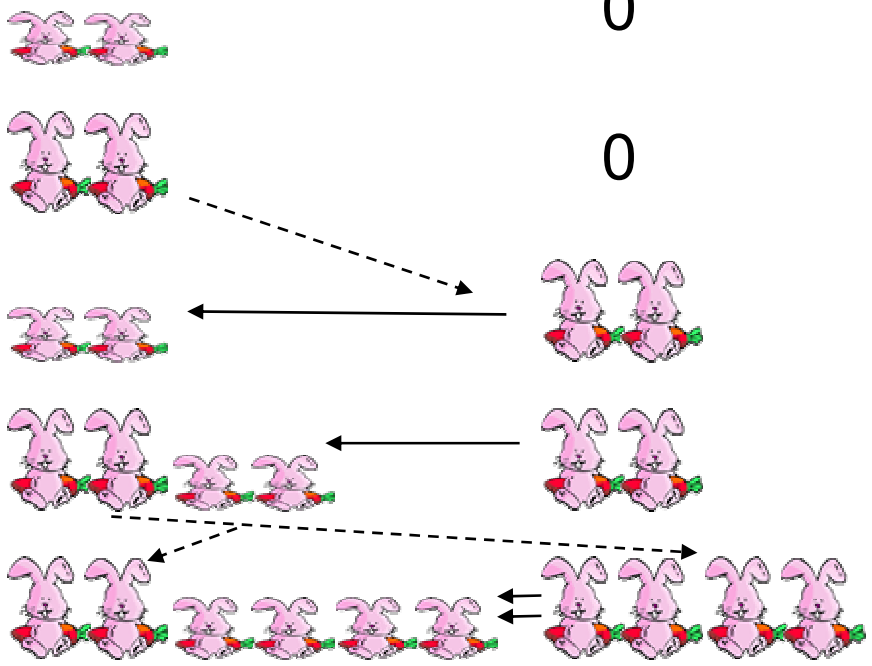
- A rabbit pair starts breeding only after 2 months.
- Thereafter, every month they breed 2 rabbits.
- Rabbits live forever.

How many pairs of rabbits are there after n months?

Fibonacci Numbers

The Rabbit problem

Month	Young	Adults	Total #pairs
1		0	1
2		0	1
3			2
4			3
5			5
•			•
•			•
•			•



Fibonacci Numbers

The Rabbit problem

f(n) = the # of rabbit pairs at end of month n.

Recursive Formula:

Basis Step: f(1) = 1

f(2) = 1

Recursive Step:

f(n) = f(n-1) + f(n-2)

f(n) = the number of pairs in previous month (rabbits don't die) + the number of new born pairs .

Fibonacci Numbers

Definition:

The Fibonacci Numbers (מספרי פיבונצ'י) $f_0, f_1, f_2, \ldots$ are defined recursively:

$$f_0 = 0$$

$$f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2} \qquad \text{for } n \geq 2$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Recursive Definitions of Sets

Recursive Definition of sets is typically analogous to Strong Induction.

The Recursive Definition of a function f : $\mathcal{N} \rightarrow \mathcal{N}$:

1) Basis Step - define specific elements in the set.
2) Recursive Step - define additional elements in the set using elements already in the set.

Recursive Sets

Example:

The set **S** is defined recursively:

$3 \in S$

$x + y \in S \quad \text{if } x \in S \text{ and } y \in S$

Basis Step

Recursive Step

Analogous to Strong Induction:

Basis Step : $P(1)$

Induction Step: $(P(1) \wedge \ldots \wedge P(k)) \rightarrow P(k+1)$

S = the set of all positive integers divisible by 3.

Proof ?

Recursive Sets

Example:

The set of Mathematical Formulae.

Basis Step

f is a formula if it is a number or variable.

Recursive Step

if f, g are formulae then:

(f + g) , (f - g), (f * g), (f / g)

are formulae.

Thus: 5 , x are formulae

(5 + x) , ((5*x) - (5/5)) are formulae

Recursive Sets

Example:

The set of Logical Compound Propositions.

Basis Step

T , **F** are Compound Propositions

Statement with truth value is a Compound Proposition

Recursive Step

if p, q are Compound Propositions then:

$\neg p$, $p \wedge q$, $p \vee q$, $p \oplus q$, $p \rightarrow q$

are Compound Propositions.

Recursive Sets

Example:

The set Σ^* of strings over the alphabet Σ

Basis Step

$\lambda \in \Sigma^*$ where λ is the empty string.

Recursive Step

if $w \in \Sigma^*$ and $x \in \Sigma$ then: $wx \in \Sigma^*$

What if strings can not be empty?

Example: $\Sigma = \{0,1\}$ 0, 1, 00, 11011, 1110 ... $\in \Sigma^*$

Induction on Recursively Defined Sets

Theorem:

Math formulae have equal number of left & right parentheses.

Proof: By induction on the structure of the set of formulae

Basis Step : A number or variable has no parenthesis so number of left and right parenthesis are equal.

Inductive Step: Assume f, g are formulae with equal number of left and right parenthesis: $l_f = r_f$ & $l_g = r_g$ then:

(f + g) , (f - g), (f * g), (f / g)

each have $l_f + l_r + 1$ left parenthesis and $r_f + r_g + 1$ right parenthesis. Since $l_f = r_f$, $l_g = r_g$ we have: $l_f + l_r + 1 = r_f + r_g + 1$ - equal number of left and right parenthesis.

Recursive Procedures

Recursive functions translate directly to programming of procedures and functions.

```
function factorial (n : integer) : integer
        if n = 1 then
                factorial := 1                    Basis Step
        else
                factorial := n*factorial(n-1)     Recursive Step
        endif
end
```

Recursive Procedures

```
function power (a : pos real,  n : non-neg integer) : integer
        if n = 0 then
                power := 1                          Basis Step
        else
                power := a*power(a,n-1)     Recursive Step
        endif
end
```

Recursive Procedures

```
function g (a ,b : integer) : integer
        if a < b   then    g := g(b,a)
        else    /* a ≥ b */
                if a == b then  g := a          Basis Step
                else  /* a > b */
                        g := g(a-b,b)           Recursive Step
                endif
        endif
end
```

What does this function compute?

Recursive Procedures

function g (a ,b : integer) : integer

Example runs:

a , b
9 , 6
3 , 6
6 , 3
3 , 3

g(9,6) = 3

a , b
7 , 3
4 , 3
1 , 3
3 , 1
2 , 1
1 , 1

g(7,3) = 1

a , b
10 , 2
8 , 2
6 , 2
4 , 2
2 , 2

g(10,2) = 2

Recursive Procedures

Improving the GCD function:

```
function g (a ,b : integer) : integer
        if b = 0   then    g := a          Basis Step
        else    /* b ≠ 0 */
                   g := g(b,a mod b)       Recursive Step
        endif
end
```

Example runs:

a , b	a , b	a , b
9 , 6	7 , 3	10 , 2
9 , 3	3 , 1	2 , 0
3 , 0	1 , 0	
g(9,6) = 3	g(9,6) = 1	g(10,2) = 2

Recursive Algorithms

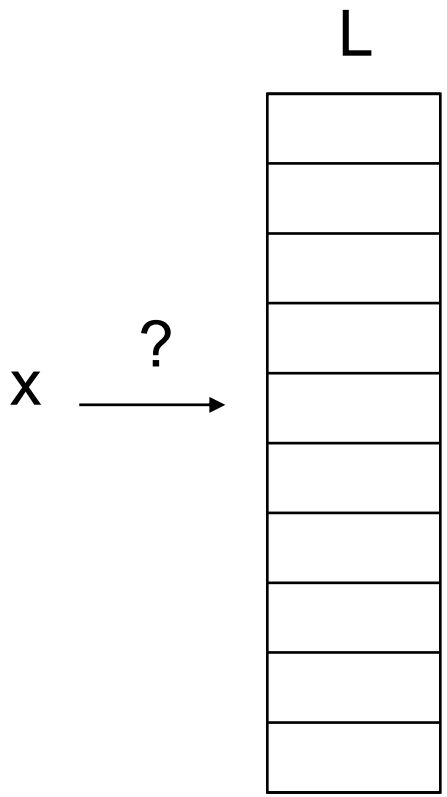
Definition:

An algorithms is called *Recursive* if it solves the given problem by reducing it to the same problem with a smaller/simpler input.

Recursive Algorithms

Example: Search Algorithms

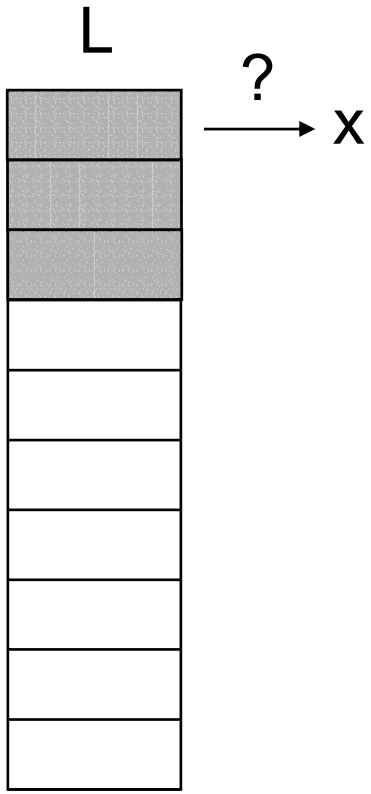
Given an ordered list L of numbers, find a given number, x, in this list.



Recursive Algorithms

Algorithm I: Linear Search (iterative)

```
function L_search (x,start,end) : integer
    for ind = start  to  end
          if L(ind) == x   /* x found */
          L_search := ind;
          return;
    endfor
    L_search := -1   /* not found */
end
```



Recursive Algorithms

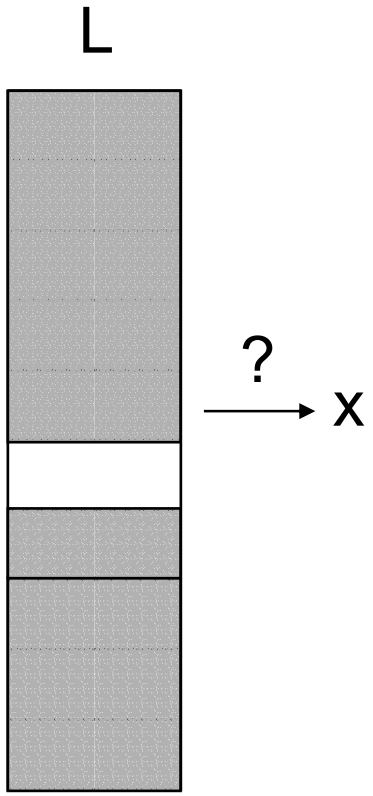
Algorithm I: Linear Search (recursive)

```
function L_search (x,start,end) : integer
    if list(start) == x   /* x found */
        L_search := start;
    elseif  start == end
        L_search := -1   /* not found */
    else    /* keep searching */
        L_search = L_search(x, start+1,end);
    endif
end
```

Recursive Algorithms

Algorithm II: Binary Search

- Test middle of list.
- If equal to x then - found.
- Else, reduce search to smaller search:
  - if x is smaller than middle element - search 1st half of list.
  - if x is greater than middle element - search 2nd half of list.



Recursive Algorithms

```
function  B_search (x,start,end) : integer
    middle = floor((start+end)/2)
    if list(middle) == x   /* x found */
          B_search := middle;
    elseif  (x<list(middle)) & (start < middle)   /* search 1st half of list */
          B_search :=  B_search(x,start,middle-1)
    elseif  (x>list(middle) & (end > middle)     /* search 2nd half of list */
          B_search :=  B_search(x,,middle+1,end)
    else
          B_search := -1   /* not found */
    endif
end
```

Recursive Algorithms

Run times:

Linear search worst case O(n)

Binary Search worst case $O(\log_2 n)$

When number is in first in list:

Best case for linear search & worst case for binary search.

What should be minimized in terms of search time?

worst case

average case ?

simplest

Recursive Algorithms

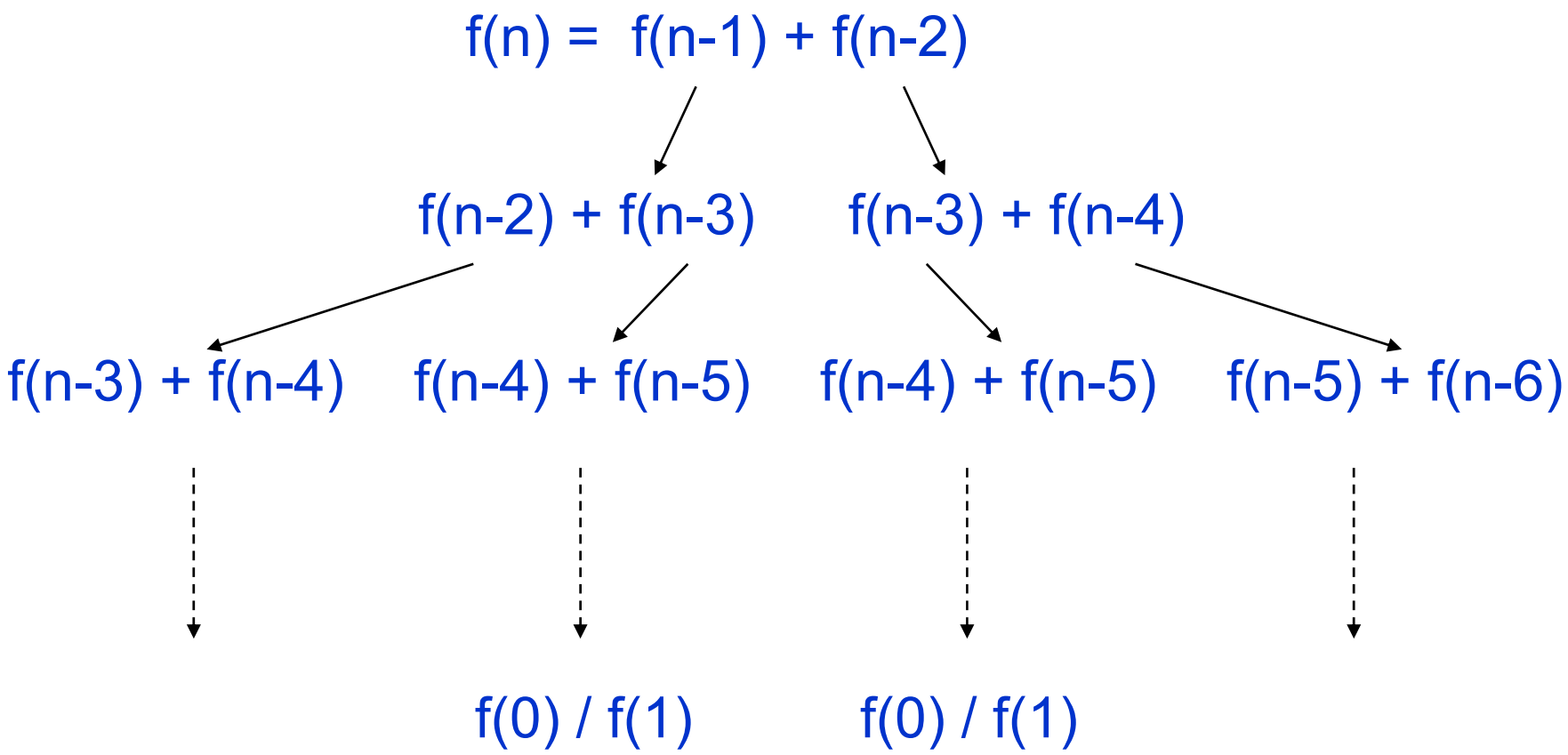
Recursive solution is not always faster!

```
function f (n: non-neg integer) : integer
   if n == 0
         f := 0;
   elseif n == 1
         f := 1
   else
         f := f(n-1) + f(n-2);
  endif
end
```

What does this function calculate?

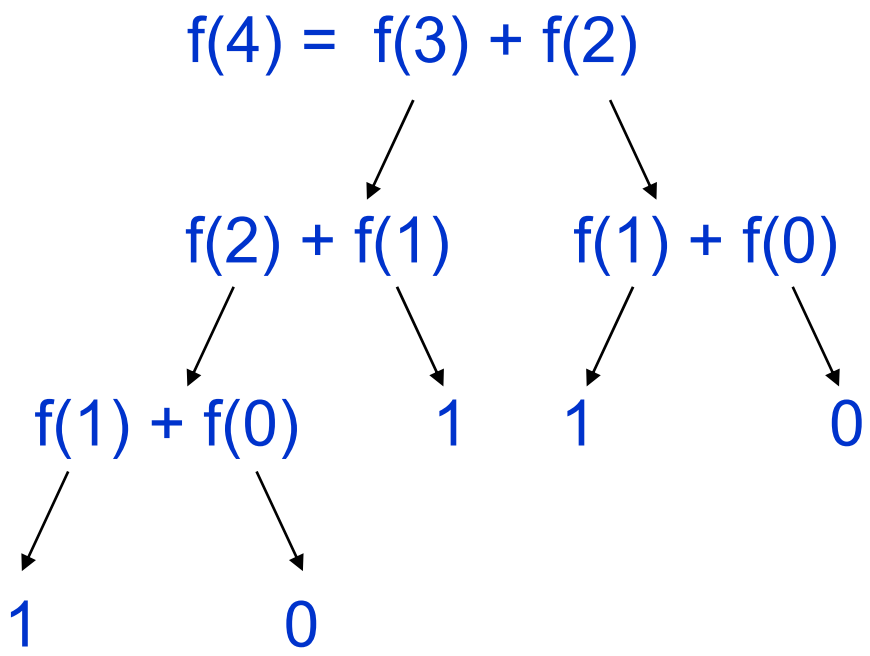
Recursive Algorithms

Recursive calculation of Fibonacci Numbers



Recursive Algorithms

Example:



f(4) = 1+0+1+1+0 = 3

Recursive Algorithms

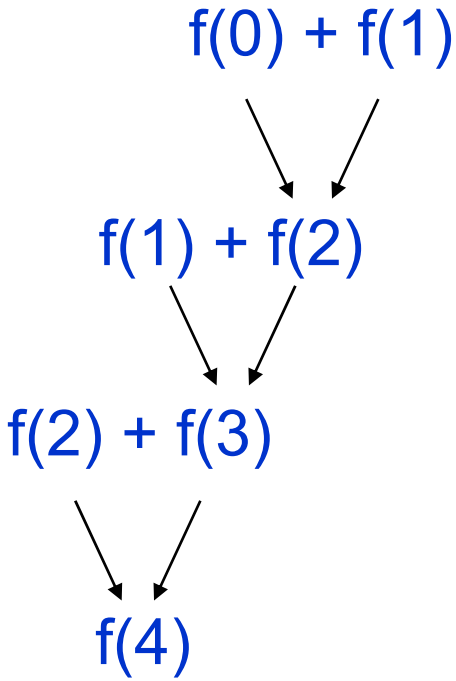
Iterative calculation of Fibonacci Numbers

```
function fib (n: non-neg integer) : integer
    if n == 0   then   fib := 0;
    else  /* n ≥ 1 */
        n_1 := 0;
        fib := 1
        for i = 1 to n-1
            n_2 := n_1;
            n_1 := fib;
            fib = n_1 + n_2
        end;
    endif;
```

Recursive Algorithms

Iterative calculation of Fibonacci Numbers

Example:



f(4) = 1+0+1+1+0 = 3

Recursive Algorithms

Comparison of number of Ops Iterative vs Recursive of Fibonacci Numbers

Recursive Alg: f_n is computed with $f_{n+1} - 1$ additions.

Iterative Alg: f_n is computed with $n - 1$ additions.

Recursive algorithms are often more time consuming BUT they are easier to understand and design.

Recursive Fun

http://www.mantasoft.co.uk/anim/

Jim Bryan (UBC)