

M. Caramihai, ©2020

**STRUCTURI DE DATE
& ALGORITMI**

CURS 4

**Analiza (spatiala a)
algoritmilor.**

Iteratii si recursivitati

Aspecte generale (1)

- Analiza algoritmilor are in vedere atat aspectele **temporale** (in cati pasi se rezolva o problema) cat si cele **spatiale** (care sunt resursele de memorie necesare penntru rulara unui algoritm)
- Teoria complexitatii (**complexity theory**) are in vedere resursele utilizate in timp ce teoria computabilitatii (**computability theory**) are in vedere posibilitatea rezolvarii unei probleme prin intermediul algoritmilor, indiferent d resursele fizice utilizate.

Aspecte generale (2)

- **Iteratia** si **recursivitatea** reprezinta doua concepte fundamentale fara de care nu ar fi posibil calculul algoritmic
- Exemple de utilizare
 - Sortare nume
 - Calculul tranzactiilor cu carti de credit
 - Operatiune de tiparire
- **Observatie:** recursivitatea face algoritmii sa fie mai clari, fara a influenta in mod decisiv performanta acestora. Totusi, in anumite situatii, alg iterativi sunt mai rapizi decat cei recursivi

Definitii

- ***Iteratia*** reprezinta o operatie de repetare a etapelor (pasilor) de procesare. Numarul de repetitii este dat de diferiti factori.
- ***Rekursivitatea*** reprezinta o alta tehnica de rezolvare a problemelor, prin auto-apelarea unei functii/proceduri. Recursivitatea poate reprezenta (in unele cazuri) o metoda mai fireasca de rezolvare a problemelor decat iteratia.
- Un algoritm recursiv implica o metoda / functie capabila sa se auto-apeleze. Acest lucru este posibil prin "spargerea" problemei in componente mai mici – si mai usor de rezolvat.
- Recursivitate:
 - Directa (apelare repetata a propriei definitii)
 - Indirecta ($x \rightarrow y$ si $y \rightarrow x$)

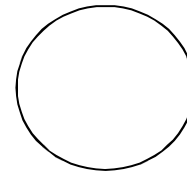
Observatii

- Algoritmii pot fi impartiti, in mod natural, in doua categorii: algoritmi iterativi sau recursivi.
- In general, algoritmii recursivi sunt mai rar aplicati decat cei iterativi.
- Iteratia poate conduce la o crestere a performantei programului. Recursivitatea poate conduce la o crestere a vitezei de programare (i.e. cresterea performantei programatorului)
- Exemplu
 - Iterativ: ospatarul taie un tort in n bucati pentru n persoane
 - Recursiv: fiecare persoana isi taie o bucata din tort si da mai departe ceea ce ramane

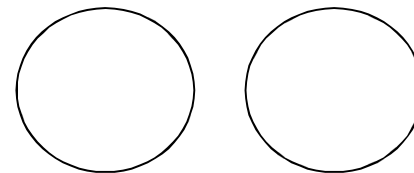
Reprezentare

Intelegerea relatiei dintre diferite moduri de reprezentare
(aceeasi informatie / idee)

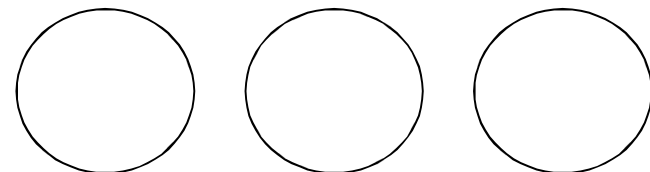
1



2



3



Reprezentarea algoritmilor

- Cod
- Exemple functionale
- Masina *Turing*
- Abstractizare de nivel inalt

Reprezentare prin cod (1)

```
class InsertionSortAlgorithm extends SortAlgorithm {  
    void sort(int a[]) throws Exception {  
        for (int i = 1; i < a.length; i++) {  
            int j = i;  
            int B = a[i];  
            while ((j > 0) && (a[j-1] > B)) {  
                a[j] = a[j-1];  
                j--; }  
            a[j] = B;  
        }  
    }  
}
```

Pro si contra?

Reprezentare prin cod (2)

Pro:

- Ruleaza pe un calculator
- Precis si succint

Contra

- Omul nu este un computer
- Este necesar un inalt nivel de intuitie
- Posibile erori
- Dependent de limbaj

Reprezentare prin exemple functionale (1)

Testarea unei probleme (rezolvari) printr'un exemplu functional.



Reprezentare prin exemple functionale (2)

88 52 98
14
31 98 88
25 30
62 23
14
79

14,23,25,30,31,52,62,79,88,98

Pro si contra?

Reprezentare prin exemple functionale (3)

Pro:

- Concret
- Dinamic
- Vizual

Contra:

- Este sarcina programatorului de a gasi un *pattern*.
- Nu trebuie explicat de ce lucreaza
- Se demonstreaza doar pentru un caz

Exemple:

testul Turing (1950) – masina inteligenta

John Searle (1980) – Camera Chineza: raspunsul masinii utilizand simboluri chinezești fara sa le inteleaga

Reprezentare prin abstractizare

Pro:

- Intuitiv (pentru operatorul uman)
- Util pentru
 - abordare
 - proiectare
 - descriere
- Corectitudine usor de identificat.

Contra:

- Matematizare excesiva
- Prea abstract

Alegere

Specificarea unei probleme de calcul

```
Max( A )


```
“preCond:
 Input is array A[1,n]
 of n values.”
i = 1
m = A[1]
loop
 exit when (i=n)
 i = i + 1
 m = max(m,A[i])
endloop
return(m)
“postCond:
 m is max in A[1,n]”
```


```

- Preconditii:
Orice ipoteza trebuie sa fie adevarata in raport cu intrarile.

- Postconditii:
Ce trebuie sa fie adevarat la terminarea algoritmului / programului

Corectitudine

$\langle \text{PreCond} \rangle \ \& \ \langle \text{cod} \rangle \Rightarrow \langle \text{PostCond} \rangle$

Daca intrarile indeplinesc preconditioniile si codul este corect, iesirile trebuie sa indeplineasca postconditiile.

Algoritmi iterativi

- Pentru un algoritm iterativ, analiza spatialitatii se reduce la identificarea declararii variabilelor, numarul de variabile, dimensiunea vectorilor, etc.

Algoritmi iterativi: bucle (1)

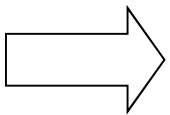
```
<preCond>  
codeA  
loop  
  <loop-invariant>  
  exit when <exit Cond>  
  codeB  
endloop  
codeC  
<postCond>
```

- Bucla:
Ipoteza a carei valoare de adevar
trebuie testata de fiecare data
cand bucla este parcursa.

Algoritmi iterativi: bucle (2)

```
<preCond>  
codeA  
loop  
  <loop-invariant>  
  exit when <exit Cond>  
  codeB  
endloop  
codeC  
<postCond>
```

Corectitudine

```
<preCond>  
<orice conditie>       <postCond>  
  cod
```

Cum poate fi dovedit ?

Numarul de iteratii nu este
cunoscut *a priori*.

Bucle: exemple (1)

Max(A)

“preCond: ~

Input is array A[1,n]
of n values.”

i = 1

m = A[1]

loop

“loop-invariant:

m is max in A[1,i]”

exit when (i=n)

i = i + 1

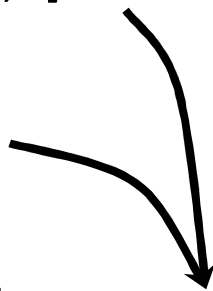
m = max(m,A[i])

endloop

return(m)

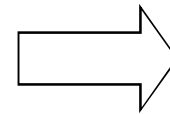
“postCond:

m is max in A[1,n]”



Pas 0

<preCond>
codA



<loop-inv>

Bucle: exemple (2)

Max(A)

“preCond: ◊

Input is array A[1,n]
of n values.”

i = 1

m = A[1]

loop

“loop-invariant:

~~m is max in A[1..i]~~

~~exit when (i=n)~~

~~i = i + 1~~

~~m = max(m, A[i])~~

endloop

return(m)

“postCond:

m is max in A[1,n]”

Pas 1

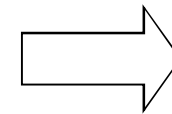
Pas 2

Pas i

<loop-invariant>

¬<exit Cond>

codB



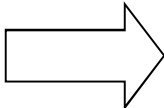
<loop-inv>

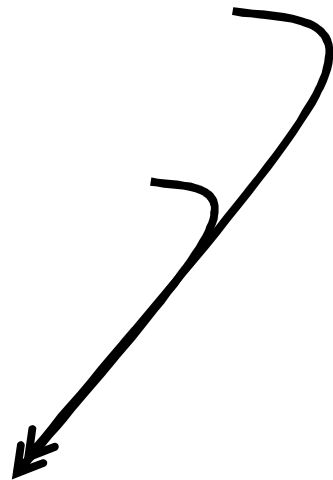
Bucle: exemple (3)

```
Max( A )

```

Ultimul pas

<loop-invariant>
<exit Cond>  <postCond>
codC



Finalizare algoritm

Trebuie definiți anumiți indicatori
de progres a.i. să se poată determina
posibilitatea de finalizare a algoritmului.

Algoritmi iterativi

- O buna modalitate de structurare a numeroase programe.
- Stocheaza informatii cheie (cunoscute de la inceput) in anumite structuri de date.
- Se pot modifica relativ usor.

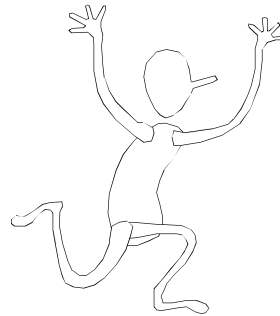
Abordare: exemplu (1)



Drumul la scoala

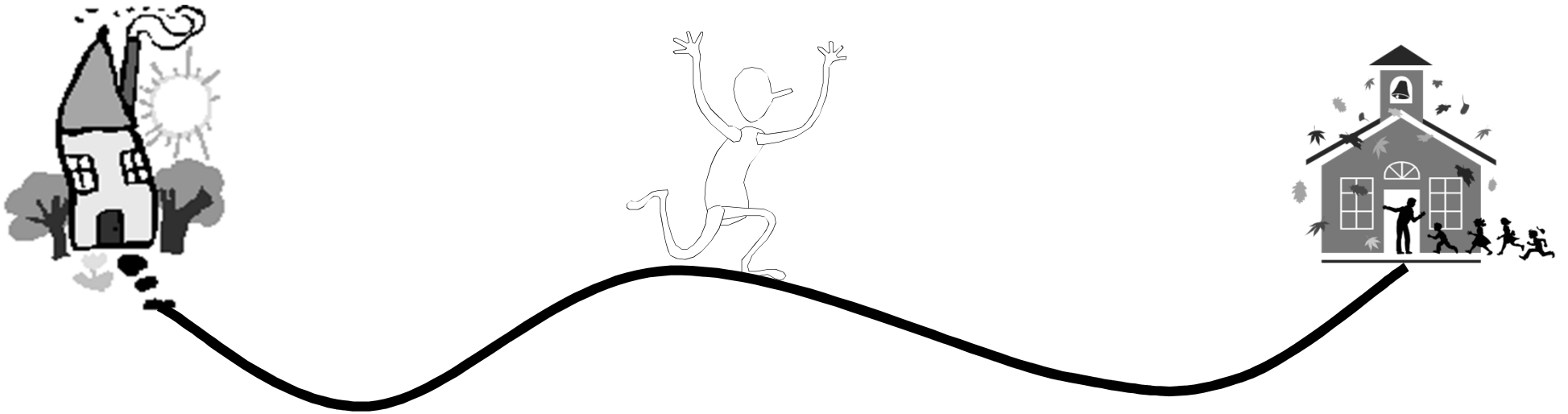
Abordare: exemplu (2)

- Preconditie: locatia casei si a scolii
- Postconditie: deplasare de acasa la scoala



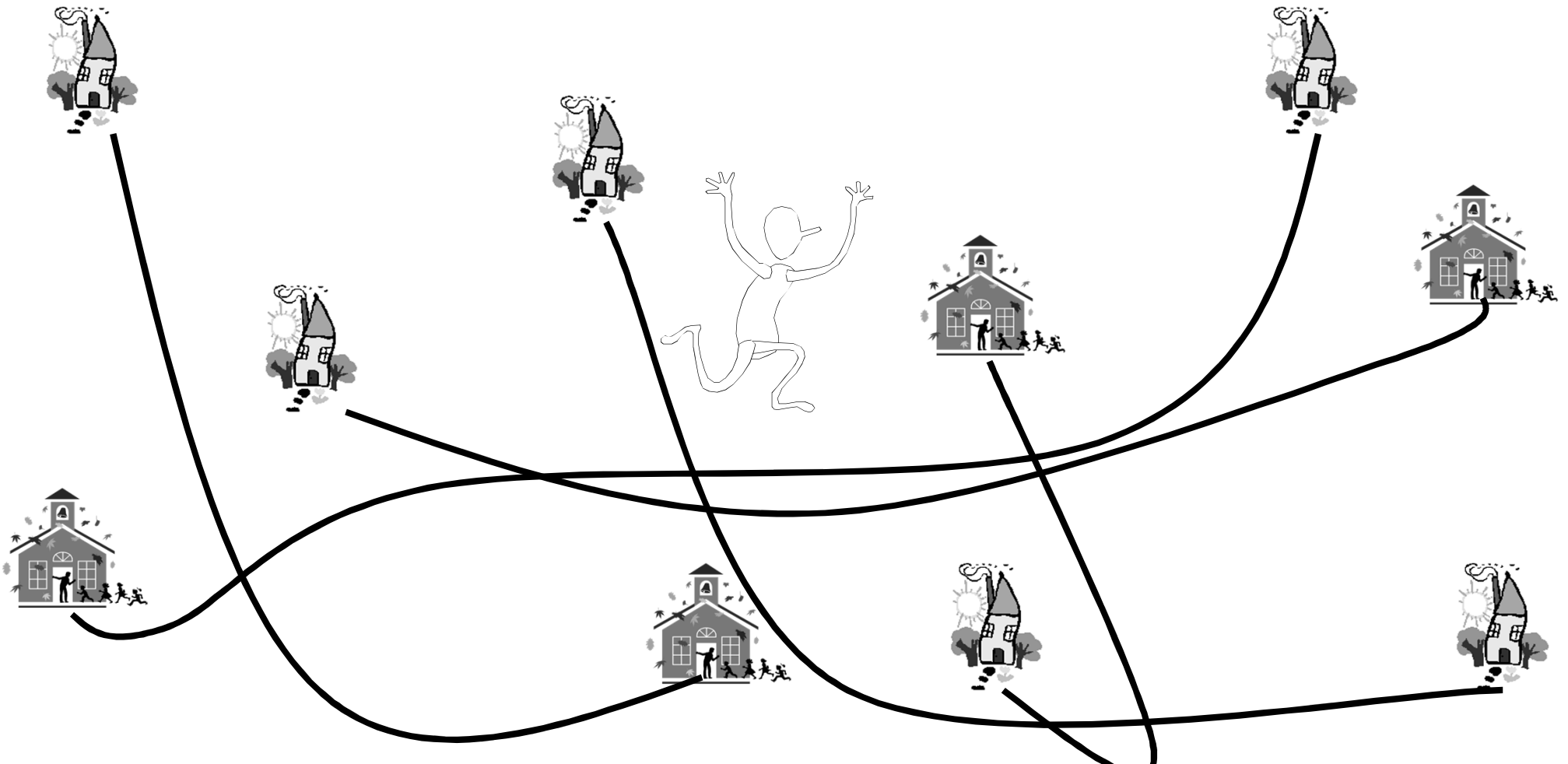
Abordare: exemplu (3)

- Algoritmul trebuie sa defineasca drumul de acasa la scoala.



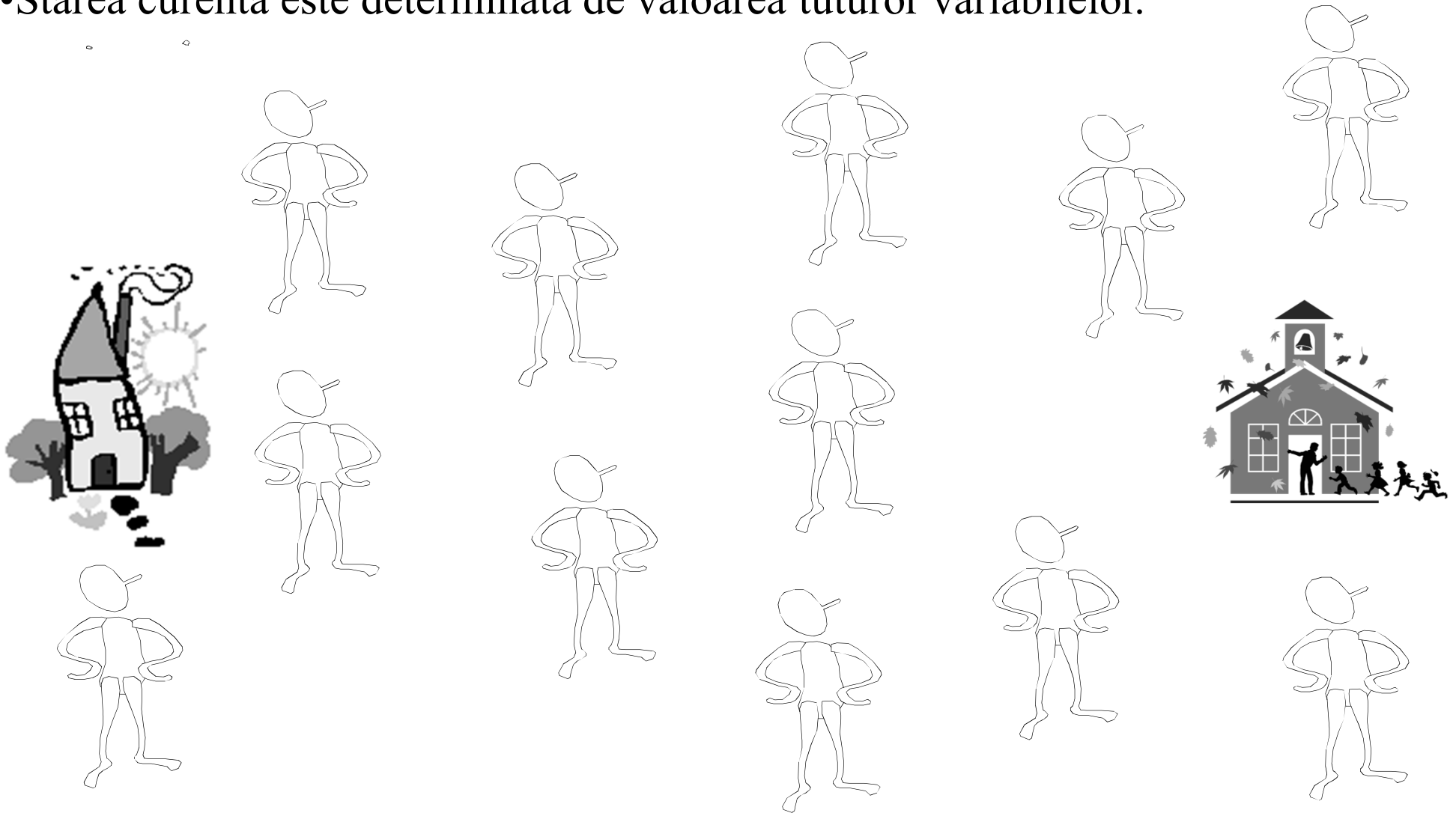
Abordare: exemplu (4)

- Exista o infinitate de intrari
- Algoritmul trebuie sa fie functional pentru fiecare dintre ele.



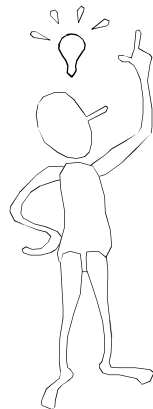
Abordare: exemplu (5)

- Starea curenta este determinata de valoarea tuturor variabilelor.



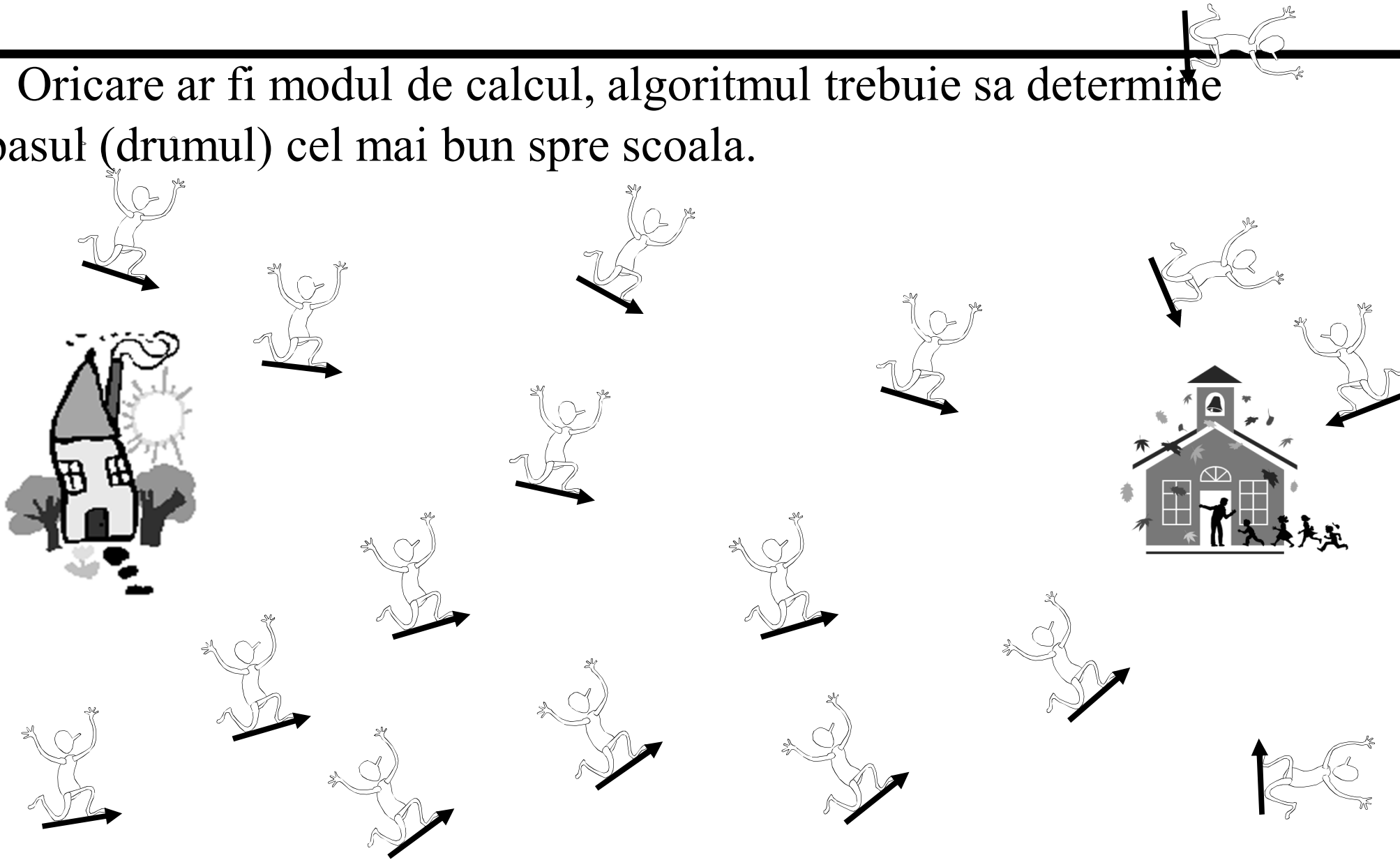
Principiu general

- NU trebuie analizat intregul algoritm de la bun inceput !
- Se parcurge cate un singur pas la fiecare interval de timp

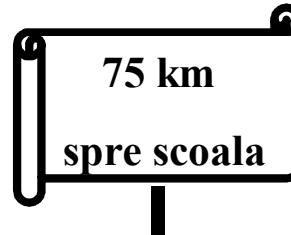
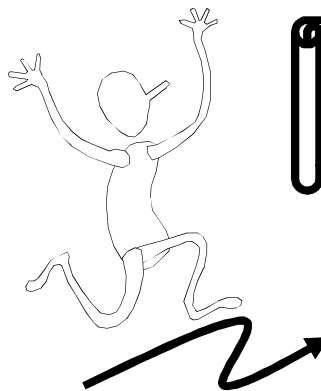
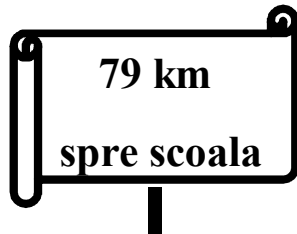


Definire algoritm

- Oricare ar fi modul de calcul, algoritmul trebuie sa determine pasul (drumul) cel mai bun spre scoala.



Cum se masoara progresul ?

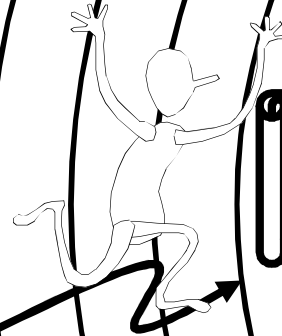


Algoritmi functionali (1)

- Calculul permite apropierea graduala de obiectiv.



79 km
spre scoala

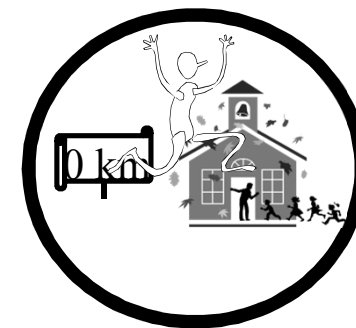
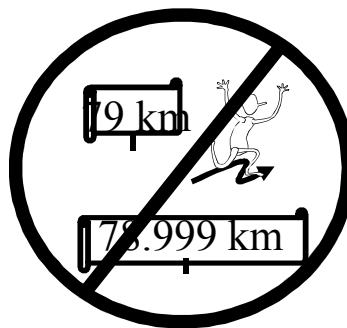
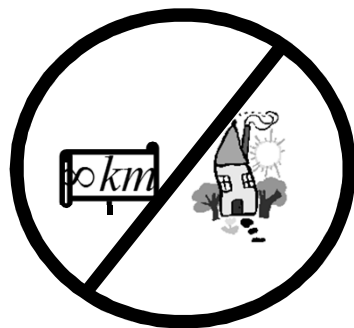
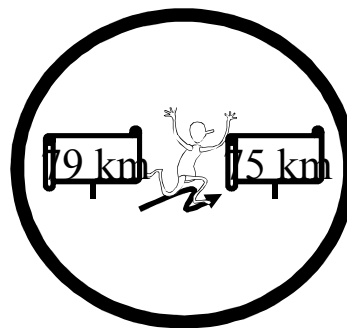
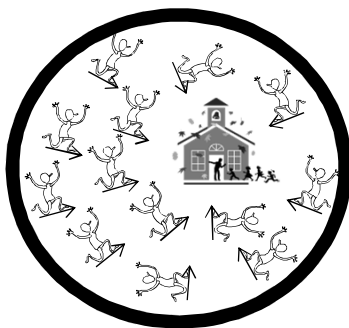


75 km
spre scoala



Algoritmi funzionali (2)

- Conditionari



Finalizarea unui algoritm

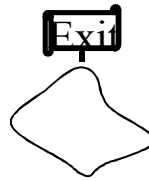
Definirea conditiilor de iesire

Terminare:

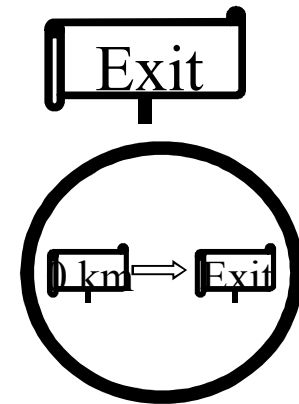
Parcursarea algoritmului trebuie sa conduca la indeplinirea conditiilor de iesire.

La finalizare, se cunosc

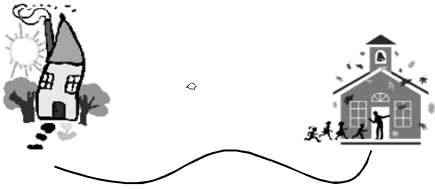
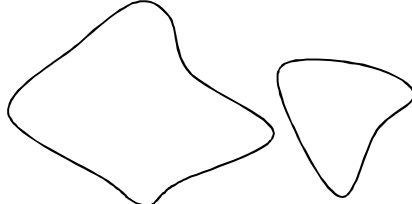
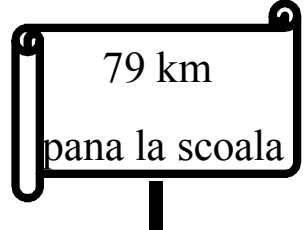
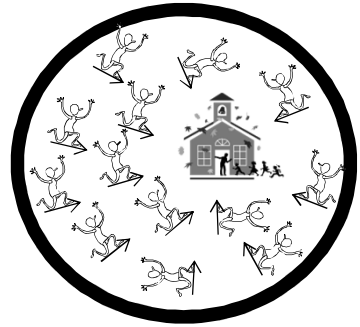
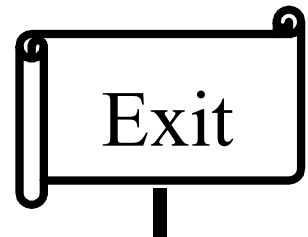

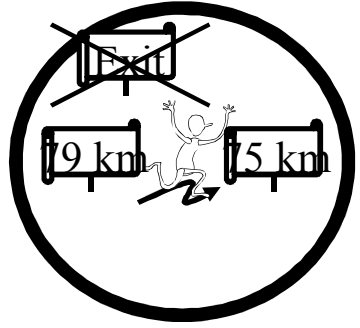


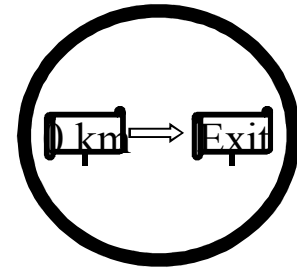
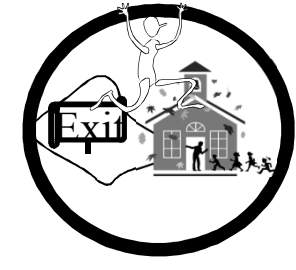
- Conditile de iesire (Adevarate)
- Invarianta buclei



In raport cu aceste elemente trebuiesc stabilite postconditiile



Sinteza...

Definire problema	Definire invarianta bucla	Definire progres
		
Definire pas	Definire conditii iesire	Mentinere invarianta bucla
		
Realizare progres	Conditii initiale	Finalizare
	 	 

Ce reprezinta invarianta buclei ?

◦ Invarianta buclei reprezinta

- Cod
- O preconditie
- O postconditie
- O propozitie cu valoare de adevar (intotdeauna) d.e.
"1+1=2"
- Pasi parcursi de algoritm



O problema

□ Cum se înmulțesc 2 numere complexe ?

$$\square (a+bi)(c+di) = [ac - bd] + [ad + bc] i$$

□ Input: a, b, c, d Output: $ac - bd, ad + bc$

□ Dacă o înmulțire costă 1 și o adunare costă 0.01. Întrebare: care este cea mai ieftină soluție de obținere a rezultatului ?

□ Poate fi mai bine de 4.02?

Metoda Gauss

Intrari: a,b,c,d lesiri: ac-bd, ad+bc

$$\square m_1 = ac$$

$$\square m_2 = bd$$

$$\square A_1 = m_1 - m_2 = ac - bd$$

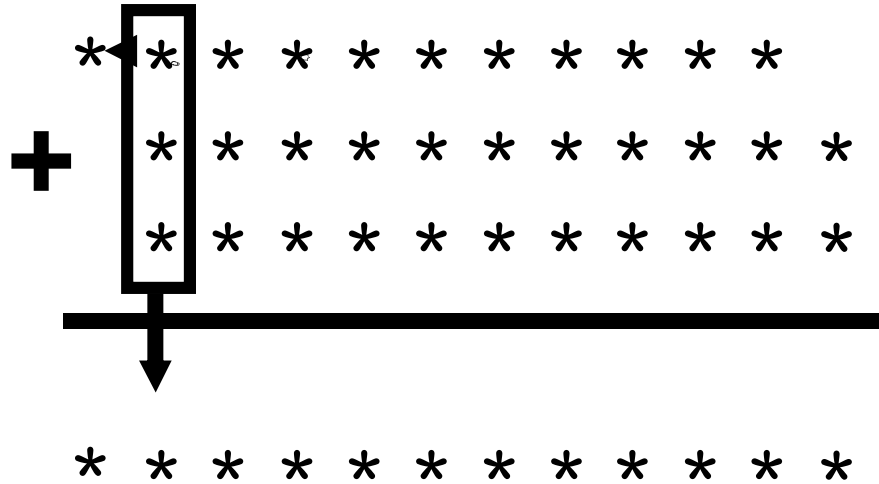
$$\square m_3 = (a+b)(c+d) = ac + ad + bc + bd$$

$$\square A_2 = m_3 - m_1 - m_2 = ad + bc$$

Interpretare

- Metoda Gauss permite "salvarea" unei operatii de inmultire (i.e. 25% mai putin de lucru).
- Poate fi considerata aceasta o metoda generala de efectuare intotdeauna a 3 inmultiri in loc de 4 ?
- Performanta: 3.05 vs. 4.02 !

Adunare a 2 numere binare

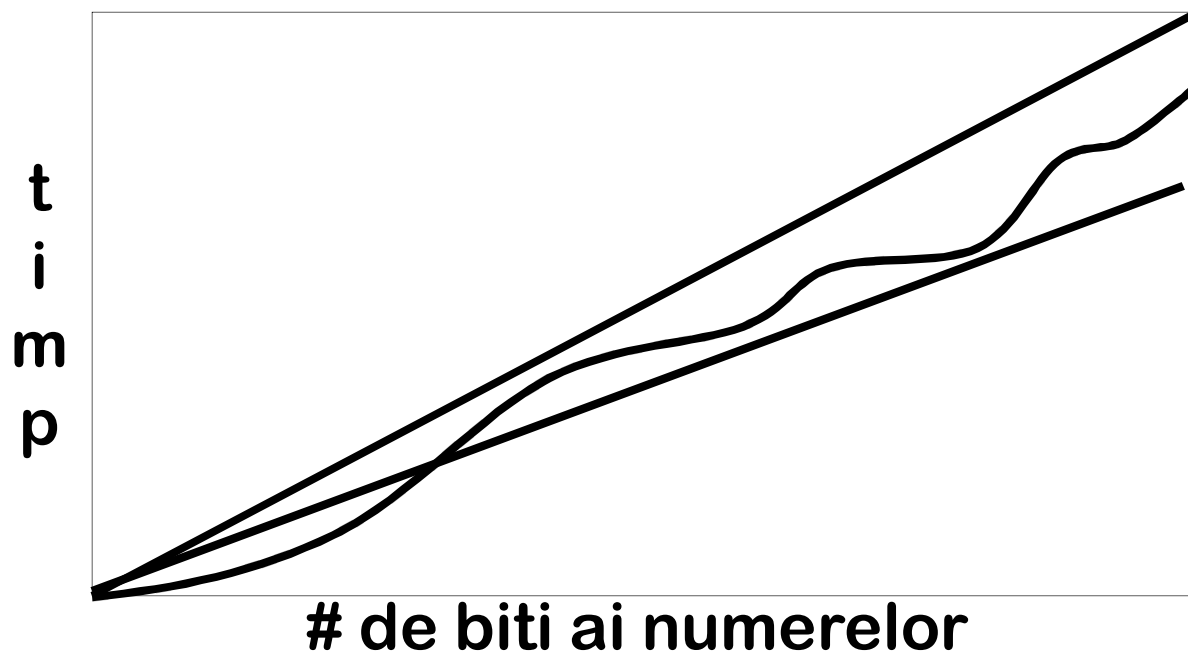


Pe un calculator obisnuit, adunarea a doua numere binare se poate face intr'o perioada fixa de timp.

$T(n)$ = Durata de timp pentru
adunarea a doua numere, fiecare
de catre n biti
 $= \theta(n)$ = functie lineara.

Adunare a 2 numere binare

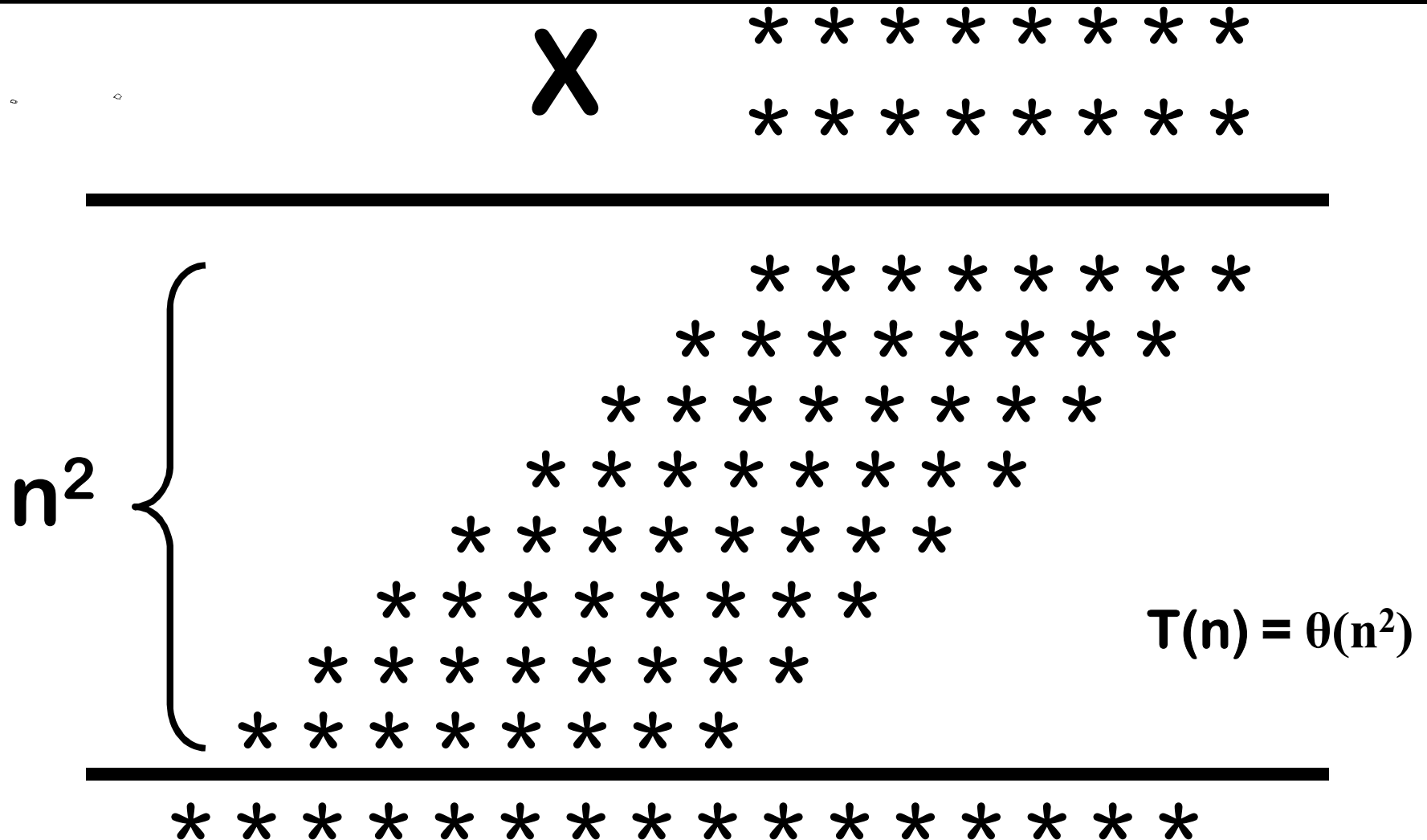
$f = \theta(n)$ semnifica faptul ca f este marginita de doua linii



Intrebare

- **INTREBARE:** Poate exista si un alt algoritm mai performant de adunare a doua numere de cate n biti?
- Performanta se refera la timpul alocat
- Raspuns: **Nu**

Inmultirea a doua numere de cate n biti



Algorithmul Kindergarten (1)

$$a \times b = \underbrace{a + a + a + \dots + a}_b$$

$$T(n) = \theta(b) = \text{time linear}$$

Este mai rapid ?

$$10000000000000000000 \times 10000000000000000000$$


Dar pentru inmultirea de mai sus ?

Algoritmul Kindergarten (2)

$$a \times b = \underbrace{a + a + a + \dots + a}_b$$

$$\begin{aligned} T(n) &= \text{Timp multiplicare} \\ &= \theta(b) = \theta(2^n). \end{aligned}$$

Numere de 2 n-biti

$$1000000000000000000000000 \times 1000000000000000000000000$$

$$\text{Valoare} = b = 1000000000000000000000000$$

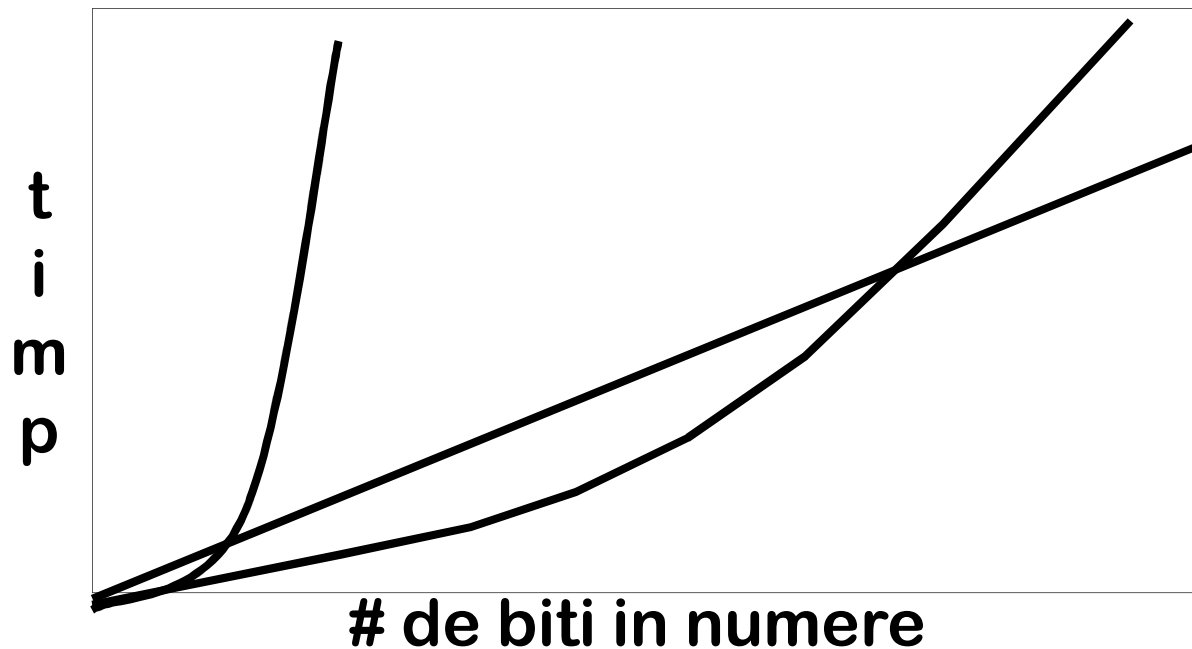
$$\# \text{ biti} = n = \log_2(b) = 60$$

Aspecte comparative

Adunarea: Timp Linear

Inmultirea: Timp Patratic

Inmultirea Kindergarten : Timp Exponential



Exemplu

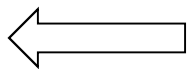
```
def countdown(i):  
    print i  
    countdown(i-1)
```

Observatie: algoritmul este recursiv si infinit (i.e. nu are conditie de oprire)

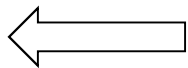
Orice algoritm recursiv contine doua parti:

- i. O functie de baza (nu se poate auto-apela – valoarea este cunoscuta fara a se utiliza recursivitatea)
- ii. O functie recursiva (se autoapeleaza)

```
def countdown(i):  
    print i  
    if i <= 0:  
        return  
    else:  
        countdown(i-1)
```



Base case

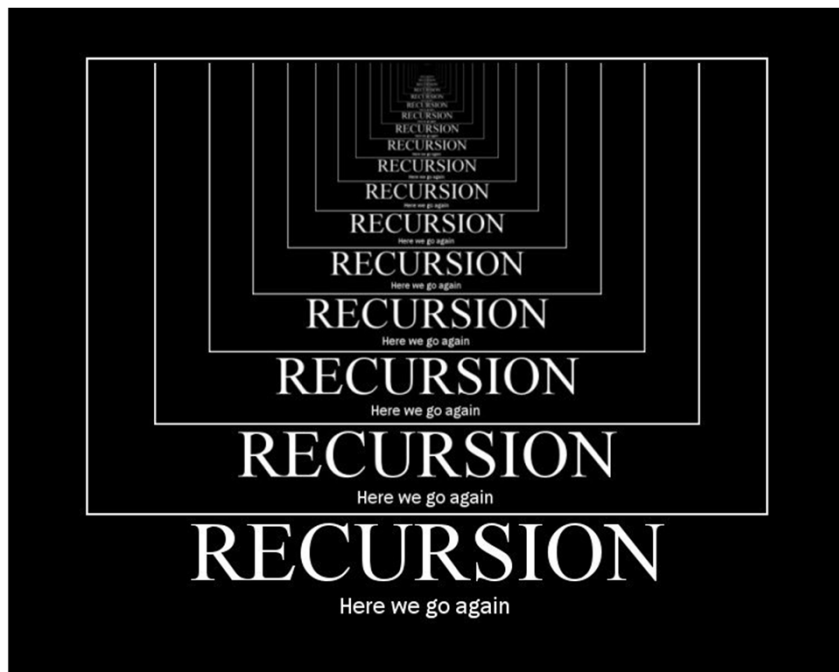


Recursive case

Regulile recursivitatii

Reguli:

1. Trebuie sa existe cel putin un caz care nu se rezolva prin recursivitate
2. Fiecare apel de recursivitate trebuie sa modifice cel putin un parametru
3. Se porneste de la ipoteza ca recursivitatea functioneaza
4. Regula non-duplicarii: aceeasi instanta NU trebuie rezolvata prin recursivitati paralele



<https://medium.freecodecamp.org/learn-recursion-in-10-minutes-e3262ac08a1>

Reprezentari diferite ale algoritmilor recursivi

Reprezentari

Pro

Cod

- Implementabili pe computer

Stack of Stack Frames

- Ruleaza pe computer

Tree of Stack Frames

- Vizibilitate a modului de calcul

Inductie

Observatie: fiecare apel la recursivitate trebuie sa “apropie” cu un pas de solutie

Reprezentare prin cod (1)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

Pro & contra ?

Reprezentare prin cod (2)

Pro:

- Ruleaza pe calculator
- Precis si succint

Contra:

- Omul nu rationeaza ca un calculator
- Este necesar un nivel ridicat de intuitie.
- Afectata de *bug*-uri
- Dependent de limbaj

Stack of Stack Frames (1)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

X = 2133

Y = 2312

ac =

bd =

$(a+b)(c+d) =$

XY =

Stack Frame: O executie particulara a unei rutine in raport cu o instanta particulara de intrare.

Stack of Stack Frames (2)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

X = 2133

Y = 2312

ac = ?

bd =

$(a+b)(c+d) =$

XY =

Stack of Stack Frames (3)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

X = 2133

Y = 2312

ac = ?

bd =

$(a+b)(c+d) =$

XY =

X = 21

Y = 23

ac =

bd =

$(a+b)(c+d) =$

XY =

Stack of Stack Frames (4)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

X = 2133

Y = 2312

ac = ?

bd =

$(a+b)(c+d) =$

XY =

X = 21

Y = 23

ac = ?

bd =

$(a+b)(c+d) =$

XY =

Stack of Stack Frames (5)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

X = 2133

Y = 2312

ac = ?

bd =

$(a+b)(c+d) =$

XY =

X = 21

Y = 23

ac = ?

bd =

$(a+b)(c+d) =$

XY =

X = 2

Y = 2

XY =

Stack of Stack Frames

Reprezentarea unui algoritm

Pro:

- Inregistreaza (*trace*) tot ce se intampla in computer
- Concret.

Contra:

- Aproape imposibil de descris in cuvinte
- Nu explica de ce lucreaza (de ce se obtin rezultate).
- Demonstreza doar pentru una (din mai multe intrari).

Tree of Stack Frames

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

X = 2133
Y = 2312
ac = 483
bd = 396
(a+b)(c+d) = 1890
XY = 4931496

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 54
Y = 35
ac = 15
bd = 20
(a+b)(c+d) = 72
XY = 1890

X = 2
Y = 2
XY = 4

X = 1
Y = 3
XY = 3

X = 3
Y = 5
XY = 15

X = 3
Y = 1
XY = 3

X = 3
Y = 2
XY = 6

X = 6
Y = 3
XY = 18

X = 5
Y = 3
XY = 15

X = 4
Y = 5
XY = 20

X = 9
Y = 8
XY = 72

Tree of Stack Frames

Reprezentarea unui algoritm

Pro:

- Vizibilitate pentru intregul proces de calcul.
- Aplicabil pentru aplicatiile in timp real.

Contra:

- Trebuie descris intregul arbore.
 - Pentru fiecare modul
 - intrare
 - calcul
 - solutie.
 - Cine pe cine apeleaza
- Structura arborelui poate fi complexa.

Inductia (1)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

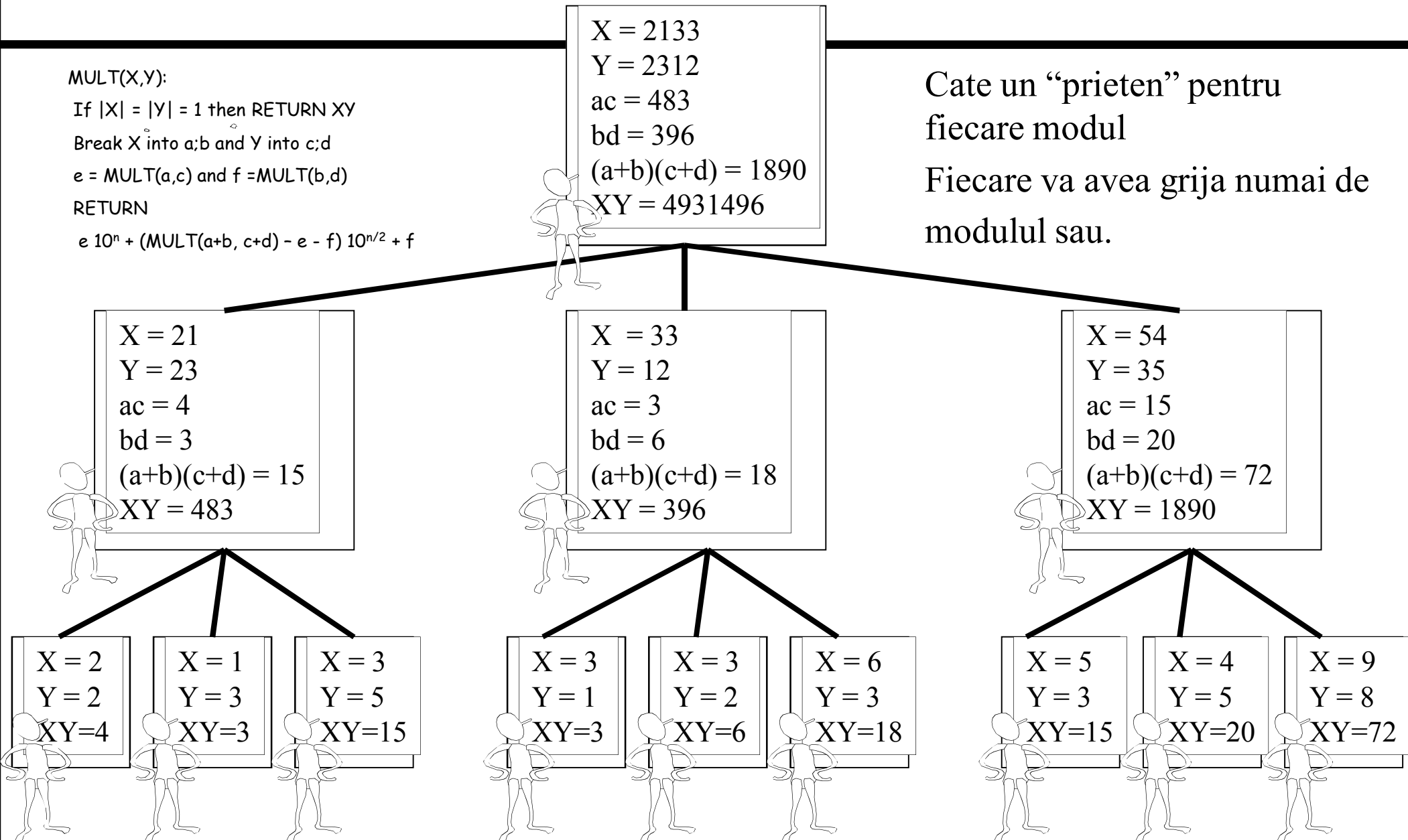
$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e \cdot 10^n + (\text{MULT}(a+b, c+d) - e - f) \cdot 10^{n/2} + f$

Cate un “prieten” pentru
fiecare modul

Fiecare va avea grija numai de
modulul sau.



Inductia (2)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

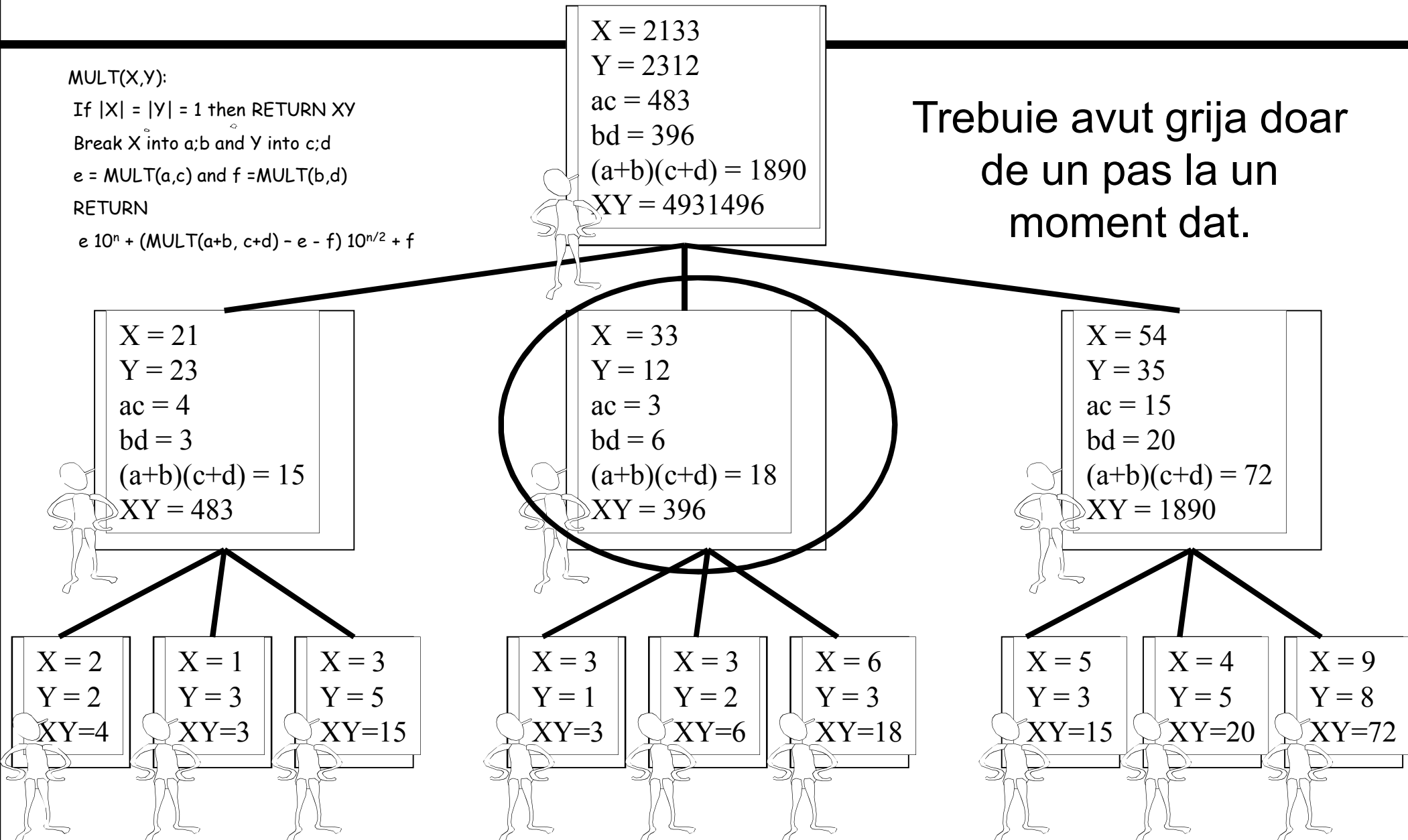
Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

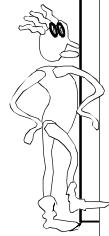
$e \cdot 10^n + (\text{MULT}(a+b, c+d) - e - f) \cdot 10^{n/2} + f$

Trebuie avut grija doar
de un pas la un
moment dat.



Demonstratie (1)

Fie intrarea



X = 33
Y = 12
ac =
bd =
(a+b)(c+d) =
XY =

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

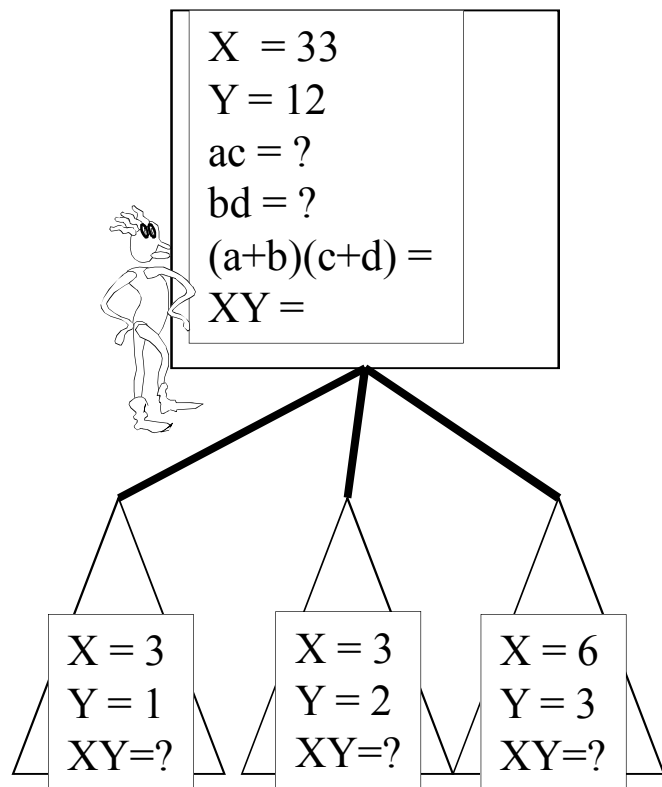
$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

Demonstratie (2)

- Fie instantierea intrarii
- Se alocă taskurile
 - Trebuie construite una / mai multe instante



MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e \cdot 10^n + (\text{MULT}(a+b, c+d) - e - f) \cdot 10^{n/2} + f$

Demonstratie (3)

MULT(X,Y):

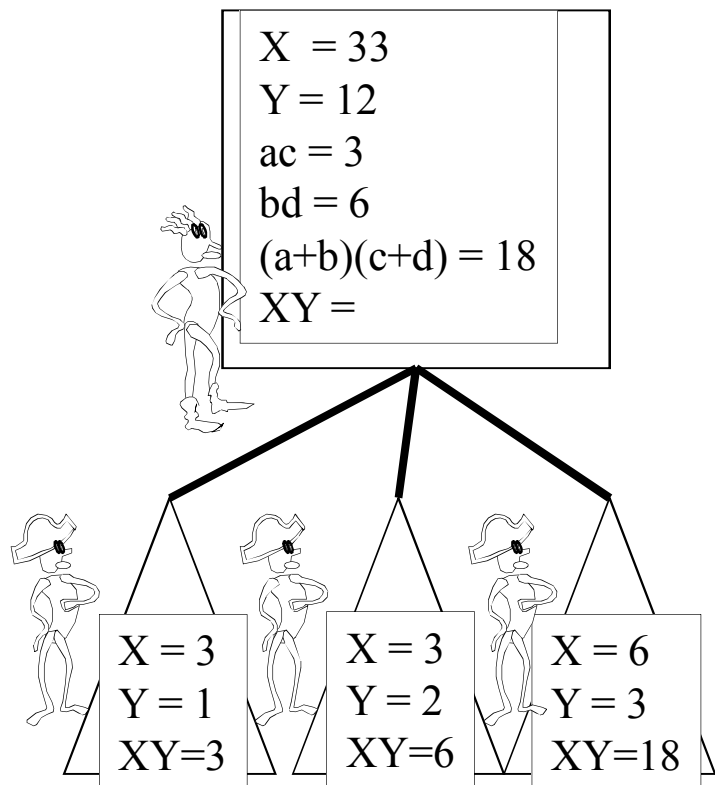
If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e \cdot 10^n + (\text{MULT}(a+b, c+d) - e - f) \cdot 10^{n/2} + f$



- Fie instantierea intrarii
- Se alocă taskurile
 - Trebuie construite una / mai multe instante
 - În fiecare modul se oferă câte un răspuns la taskul ce trebuie realizat.

Demonstratie (4)

MULT(X,Y):

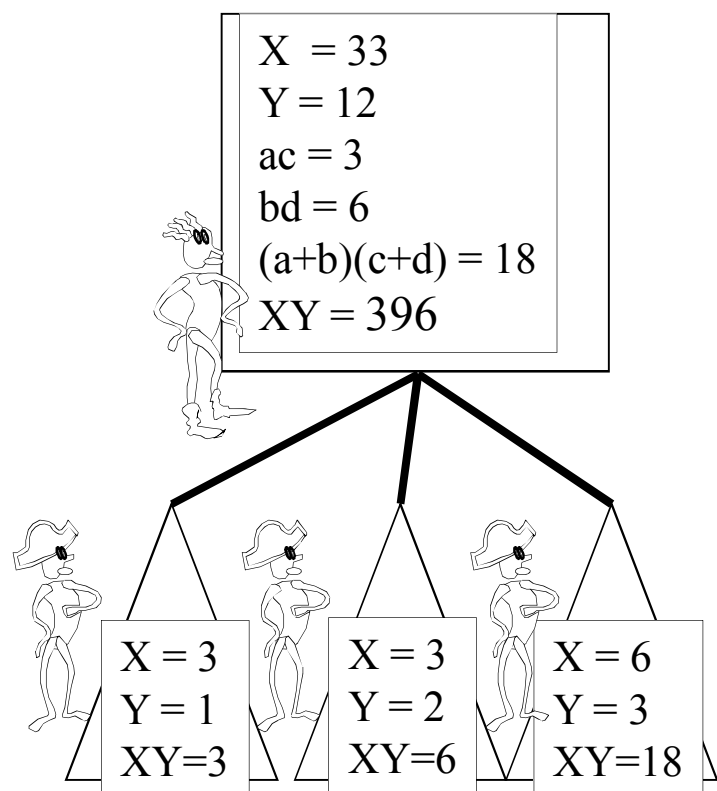
If $|X| = |Y| = 1$ then RETURN XY

Break X into $a;b$ and Y into $c;d$

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e \cdot 10^n + (\text{MULT}(a+b, c+d) - e - f) \cdot 10^{n/2} + f$



- Fie instantierea intrarii
- Se aloca taskurile
 - Trebuie construite una / mai multe instante
 - In fiecare modul se ofera cate un raspuns la taskul ce trebuie realizat.
- In acest fel poate fi realizata propria instanta.

Demonstratie (5)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into $a;b$ and Y into $c;d$

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

MULT(X,Y):

Break X into $a;b$ and Y into $c;d$

$e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$

RETURN

$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$



ac

bd

$(a+b)(c+d)$

Generic.

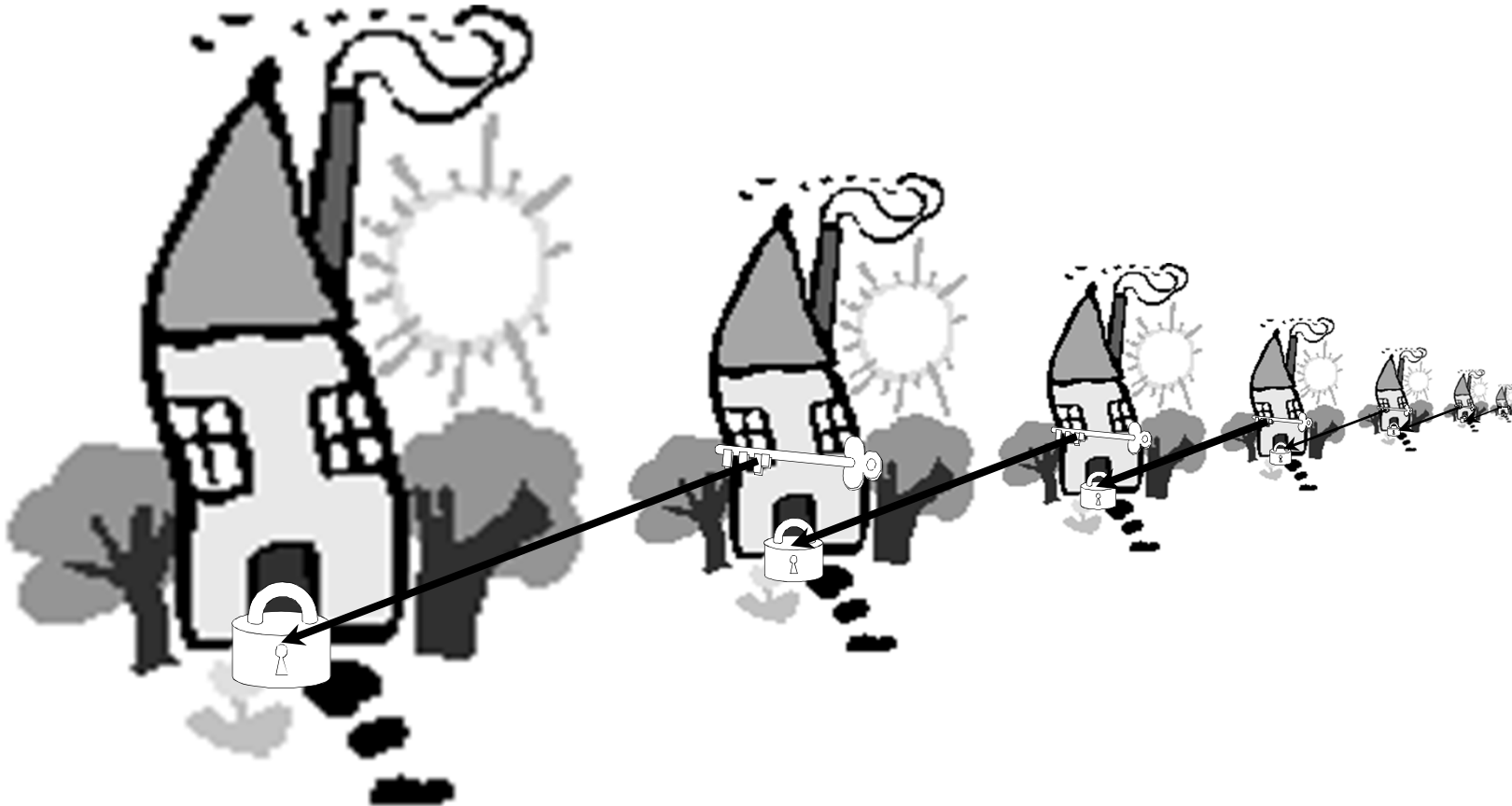
MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY



Algoritm recursiv

- Trebuie implementat un algoritm functional.
- Cu ajutorul lui se realizeaza un alt algoritm functional.



Inductia & inductia puternica (1)

$$\left. \begin{array}{l} S(0) \\ \forall i S(i) \Rightarrow S(i+1) \end{array} \right\} \Rightarrow \forall n S(n)$$

Inductie

$$\left. \begin{array}{l} S(0) \\ \forall i, [S(0), S(1), S(2), \dots, S(i-1) \Rightarrow S(i)] \end{array} \right\} \Rightarrow \forall n S(n)$$

Inductie puternica

Inductia & inductia puternica (2)

$S(n) =$

Algoritmul recursiv lucreaza pentru fiecare instanta / pas n .

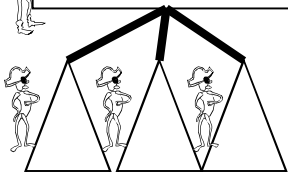
Initial

$\Rightarrow S(0)$

pas i

$\Rightarrow \forall i, [S(0), S(1), S(2), \dots, S(i-1) \Rightarrow S(i)]$

$\Rightarrow \forall n S(n) \Rightarrow$



Exemplu (1)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

Exemplu (2)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

n=3 A ? B ? C

Exemplu (3)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

n=3 A Y B X C

Exemplu (4)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

n=3 AYBXC

Exemplu (5)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

n=3 AYBXC

n=4 A ? B ? C

Exemplu (6)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

n=3 AYBXC

n=4 AAYBXC B Y C

Exemplu (7)

algorithm *Fun*(*n*)

⟨*pre-cond*⟩: *n* is an integer.

⟨*post-cond*⟩: Outputs a silly string.

begin

 if(*n* > 0) then

 if(*n* = 1) then

 put "X"

 else if(*n* = 2) then

 put "Y"

 else

 put "A"

Fun(*n* - 1)

 Put "B"

Fun(*n* - 2)

 Put "C"

 end if

 end if

end algorithm

n=1 X

n=2 Y

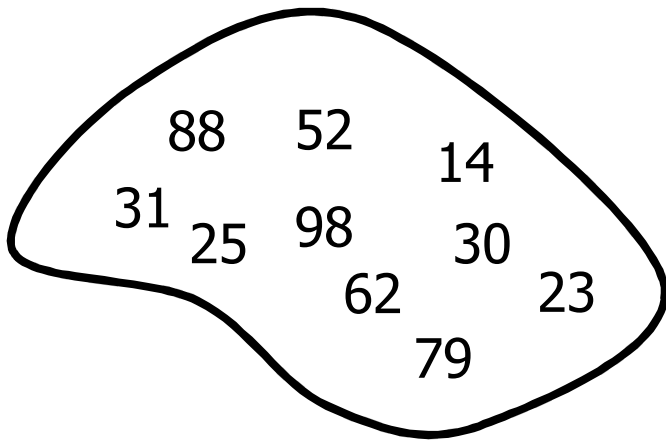
n=3 AYBXC

n=4 AAYBXCBYC

Etc...

Problema sortării

- Preconditii: Intrarea este reprezentata de o lista de n valori (cu posibile repetitii).
- Postconditii: Iesirea este reprezentata de aceeași lista ordonata crescator.



14,23,25,30,31,52,62,79,88,98

Sortarea recursiva

- Este data lista cu obiecte ce urmeaza a fi sortate
- Lista este "sparta" in doua subliste.
- In mod recursiv, fiecare sublista este sortata.
- Cele doua subliste sortate sunt combinate in lista intrega, sortata.

Tipuri de sortare recursiva

Marime subliste

$n/2, n/2$

$n-1, 1$

Efort minim de splitare
Efort mare de recombinaire

Merge Sort

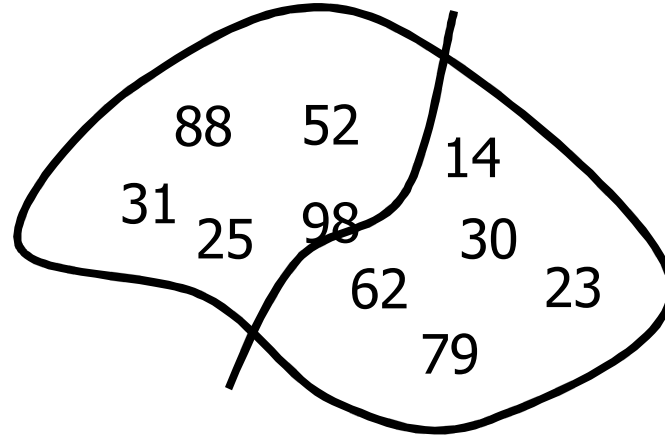
Insertion Sort

Efort mare de splitare
Efort mic de recombinaire

Quick Sort

Selection Sort

Merge sort (1)



Splitare in doua
jumatați

Algoritm pentru
prima jumătate

Algoritm pentru
a doua jumătate



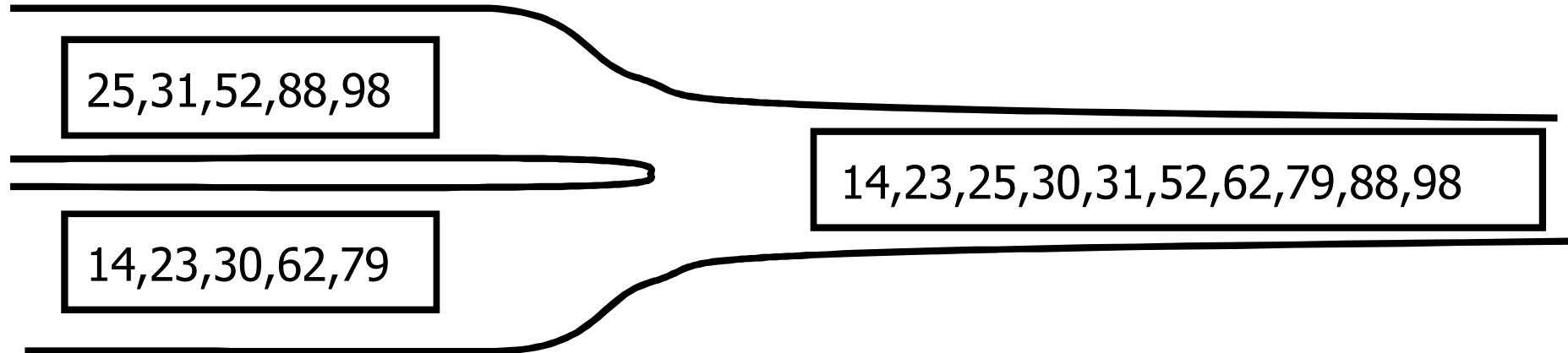
25,31,52,88,98



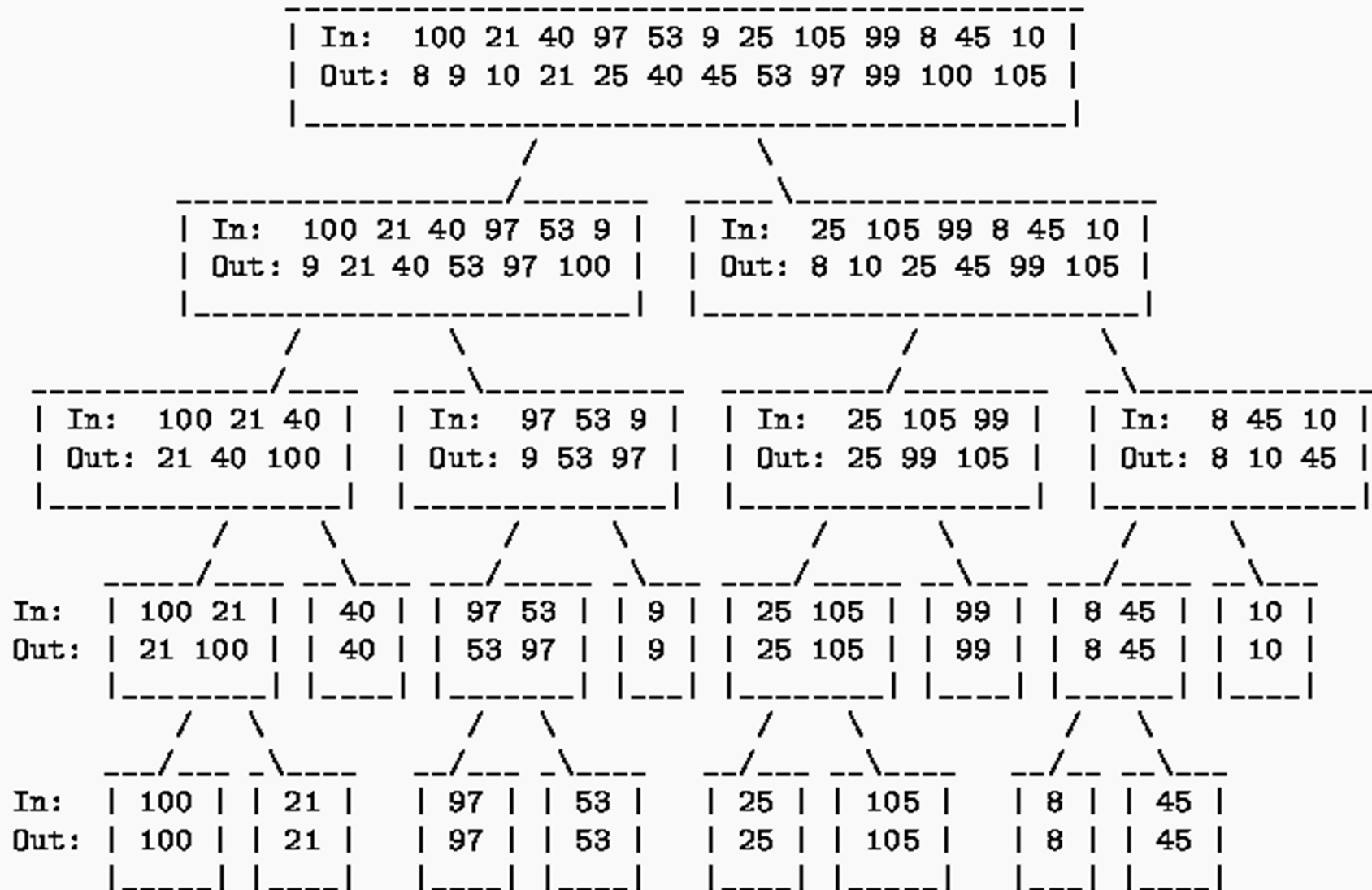
14,23,30,62,79

Merge sort (2)

Combinarea celor 2 liste sortate in una singura



Merge sort (3)



Merge sort (4)

Time: $T(n) = 2T(n/2) + \Theta(n)$
 $= \Theta(n \log(n))$

Concluzii: analiza comparativa

- Abordare:
 - Iterativ: fct se repeta pana cand o conditie nu mai este indeplinita
 - Recursiv: autoapelare fct pana la indeplinirea unei conditii
- Structura
 - Iterativ: bucle repetitive
 - Recursiv: bloc decizie
- Eficienta timp/spatiu
 - Recursiv: mai putin eficienta decat iterativ. (in anumite conditii)