# Correttezza e terminazione

Algoritmi e strutture dati

Ugo de'Liguoro, Andras Horvath

1

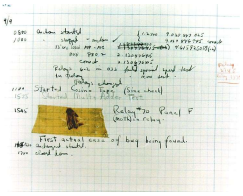
## Sommario

- Obiettivi
  - introdurre l'analisi qualitativa degli algoritmi, basata su tecniche di verifica
- Argomenti
  - pre e post condizioni
  - induzione per verificare algoritmi ricorsivi
  - invarianti di ciclo per verificare algoritmi iterativi
  - terminazione

2

## Bugs ed altri disastri

It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise — this thing gives out and [it is] then that "***Bugs***" — as such little faults and difficulties are called — show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached. (T. Edison, da Wikipedia)



3

## Bugs ed altri disastri

- 1962: la sonda Mariner 1 si schianta su Venere per un errore nel software di controllo del volo
- 1981: una TV nel Quebec decreta la vittoria alle elezioni di un partito sconosciuto; usavano un software difettoso per la rilevazione dei risultati
- 1983: un sistema computerizzato sovietico rileva un attacco nucleare con cinque missili balistici inesistenti
- 1985: alcuni pazienti vengono irradiati con dosi eccessive di raggi X dal sistema Therac-25 a controllo software

4

## Bugs ed altri disastri

- 1993: il processore Pentium, che incorpora il coprocessore aritmetico, sbaglia le divisioni in virgola mobile
- 1996: l'Ariane 5 esce dalla sua rotta e viene distrutto in volo a causa di un errore di conversione dei dati da 16 a 32 bit
- 1999: usando un nuovo sistema, le poste inglesi recapitano mezzo milione di nuovi passaporti ad indirizzi sbagliati o inesistenti
- 1999: Y2K è il celeberrimo millenium-bug
- (*continua…*)

5

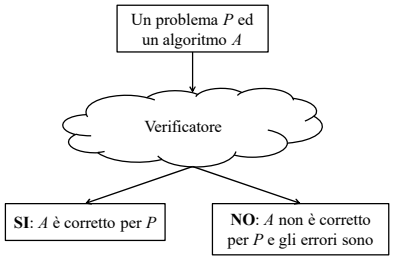
## TimSort

**Timsort** is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data.
…
It was implemented by Tim Peters in 2002 for use in the Python programming language.
…
Timsort has been Python's standard sorting algorithm since version 2.3. It is also used to sort arrays of non-primitive type in Java SE 7, on the Android platform, and in GNU Octave. (Wikipedia)

Il programma contiene un errore difficile da scoprire, perché si verifica con array di lunghezza > $2^{49}$

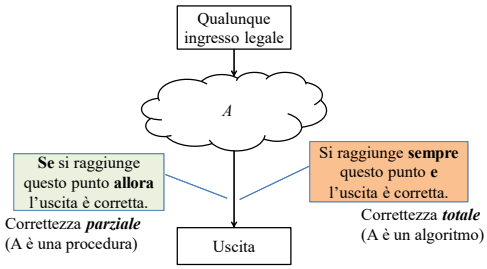
6

## Un verificatore immaginario




7

## Correttezza parziale e totale




8

## Specifica di un algoritmo

- *pre-condizioni*: ipotesi sull'ingresso (come deve essere fatto l'input)
- *post-condizioni*: proprietà dell'uscita (un criterio stabilisce come deve essere fatto l'output)

*Esempio, divisione intera:*

**Div**($a$, $b$)

**Pre**: $a \geq 0$, $b > 0$ numeri interi

**Post**: $q$ e $r$ tali che $a = b\,q + r$ e $0 \leq r < b$

9

## Che cos'è la ricorsione?

Un algoritmo è ricorsivo se nella sua definizione utilizza direttamente o indirettamente sé stesso.

- Un problema si presta ad essere risolto con un algoritmo ricorsivo quando la soluzione con un certo input ha una relazione più o meno semplice con la soluzione del problema con uno o più input "diminuiti" in qualche maniera.
- fattoriale: $n! = f(n) = n \cdot f(n-1)$
- massimo di un vettore:

  *mv(A[1..n])=max(mv(A[1..n/2]), mv(A[n/2+1..n]))*

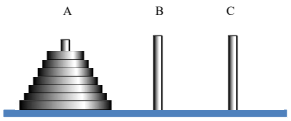
  se *n>1*

  (dove *n/2* denota *n* diviso per due arrotondato per difetto)

10

## Le torri di Hanoi (Lucas 1883)

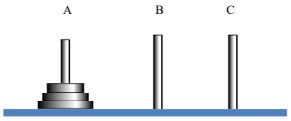
Dati tre pioli (A, B e C) e una torre di *n* dischi di diametro decrescente sul piolo A, spostare la torre dal piolo A (sorgente) al piolo B (destinazione), sfruttando il piolo C (appoggio), muovendo un disco alla volta, senza mai sovrapporre un disco più grande ad uno più piccolo.




11

## Le torri di Hanoi (Lucas 1883)

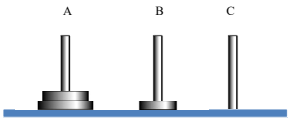
*n* = 3




12

## Le torri di Hanoi (Lucas 1883)

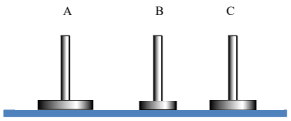
$n = 3$




13

## Le torri di Hanoi (Lucas 1883)

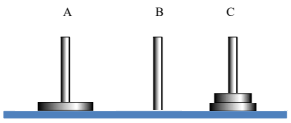
$n = 3$




14

## Le torri di Hanoi (Lucas 1883)

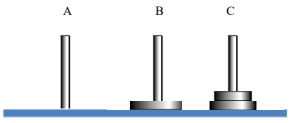
$n = 3$




15

## Le torri di Hanoi (Lucas 1883)

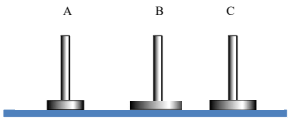
$n = 3$




16

## Le torri di Hanoi (Lucas 1883)

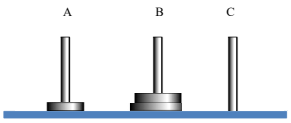
$n = 3$




17

## Le torri di Hanoi (Lucas 1883)

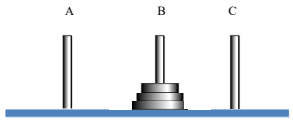
$n = 3$




18

## Le torri di Hanoi (Lucas 1883)

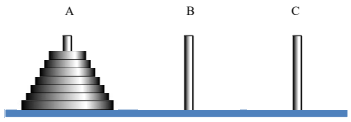
$n = 3$




19

## Le torri di Hanoi (Lucas 1883)

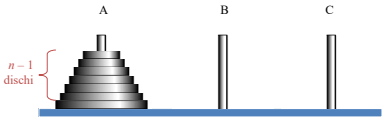
Come si procede in generale con $n$ dischi?




20

## Le torri di Hanoi (Lucas 1883)

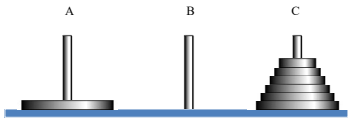
1. Sposta $n - 1$ dischi da A a C usando B




21

## Le torri di Hanoi (Lucas 1883)

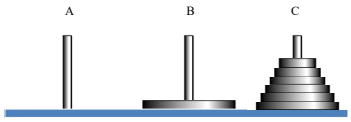
1. Sposta $n - 1$ dischi da A a C usando B
2. Sposta il disco in A su B




22

## Le torri di Hanoi (Lucas 1883)

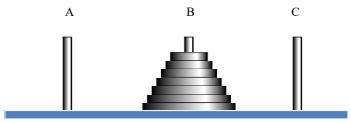
1. Sposta $n - 1$ dischi da A a C usando B
2. Sposta il disco in A su B
3. Sposta $n - 1$ dischi da C su B usando A




23

## Le torri di Hanoi (Lucas 1883)

1. Sposta $n - 1$ dischi da A a C usando B
2. Sposta il disco in A su B
3. Sposta $n - 1$ dischi da C su B usando A




24

## Le torri di Hanoi (Lucas 1883)

**Hanoi** (*n, sorgente*, *destinazione*, *appoggio*)

**Pre**: *n>0* & la base degli *n* dischi più in alto di *sorgente* ha un diametro più piccolo del disco più in alto sia di *destinazione* che di *appoggio* & la situazione attuale rispetta le regole

**Post**: la torre di *n* dischi più in alto su *sorgente* è spostata su *destinazione*

**if** *n=1* **then**

sposta un disco da *sorgente* a *destinazione*

**else**

Hanoi (*n-1*, *sorgente*, *appoggio*, *destinazione*)

sposta un disco da *sorgente* a *destinazione*

Hanoi (*n-1, appoggio*, *destinazione*, *sorgente*)

25

## Lo schema dell'induzione

1. Il caso di base: $P(1)$
2. Il passo induttivo: $P(m) \Rightarrow P(m+1)$. L'ipotesi $P(m)$ si chiama *ipotesi induttiva*.
3. 1. e 2. implicano che $\forall n \geq 1.P(n)$

$P(n) \Leftarrow \ldots \Leftarrow P(3) \Leftarrow P(2) \Leftarrow P(1)$

(se il caso base è P(k) allora la conclusione diventa $\forall n \geq k.P(n)$)

26

## Lo schema dell'induzione

La regola di inferenza (con caso base *P(0)*):

$$\frac{P(0) \qquad \forall m.P(m) \Rightarrow P(m+1)}{\forall n.P(n)}$$

(sopra la linea ci sono le **premesse** e sotto c'è la **conclusione)**

27

## Induzione per le torri

- $P(1)$: l'algoritmo è corretto se il numero di dischi è 1
  Dimostrazione: è triviale
- $P(m) \Longrightarrow P(m+1)$: se l'algoritmo è corretto con $m$ dischi allora è corretto con $m+1$ dischi
  Dimostrazione:
  - all'inizio i numeri di dischi sono: $a+m+1, b, c$
  - dopo **Hanoi (*m*, *sorgente*, *appoggio*, *destinazione*)** grazie all'ipotesi induttiva: $a+1, b, c+m$
  - dopo **sposta l'unico disco di cui è fatta *sorgente* su *destinazione***: $a, b+1, c+m$
  - dopo **Hanoi (*m*, *appoggio*, *destinazione*, *sorgente*)** grazie all'ipotesi induttiva: $a, b+m+1, c$
- quindi l'algoritmo è corretto per qualunque $m$

28

## Divisione intera

Problema della divisione intera:

**Div**($a$, $b$)

**Pre**: $a \geq 0$, $b > 0$ numeri interi

**Post**: $q$ e $r$ tali che $a = b\,q + r$ e $0 \leq r < b$

Sviluppiamo un algoritmo ricorsivo per trovare la soluzione e verifichiamo la sua correttezza.

29

## Divisione intera

Esempio:

con input *a=53, b=5*

l'output corretto è *q=10* e *r=3* con cui

$$53 = 5 \cdot 10 + 3$$

che può essere scritto anche come

$$53 - 5 \cdot 10 = 3$$

30

## Divisione intera ricorsiva

$$0 \le a \underbrace{- b - b - \ldots - b}_{q} = r < b$$

31

## Divisione intera ricorsiva

$$0 \le a \underbrace{- b - b - \ldots - b}_{q} = r < b$$

32

## Divisione intera ricorsiva

$$0 \le a \underbrace{- b - b - \ldots - b}_{q} = r < b$$

$$\Downarrow$$

$$0 \le (a - b) \underbrace{- b - \ldots - b}_{q'} = r < b$$

$a - b = b\,q' + r$
ossia $q' = q\text{-}1$, $r$ sono quoziente e resto di $a - b$ diviso $b$
con $a - b < a$ perché $b > 0$

33

## Divisione intera ricorsiva

Quale sarà il caso base?

Quando $a<b$ la soluzione è triviale ed è $q=0,r=a$.

Come sarà l'algoritmo?

Provate a scriverlo in pseudo-codice.

34

## Divisione intera ricorsiva

```
DIV-REC(a, b)
    ▷ Pre: a ≥ 0, b > 0
    ▷ Post: ritorna q, r tali che a = bq + r ∧ 0 ≤ r < b
if a < b then
    q, r ← 0, a
else
    q', r ← DIV-REC(a − b, b)
    q ← q' + 1
end if
return q, r
```

Come posso verificare la correttezza di Div-Rec?

35

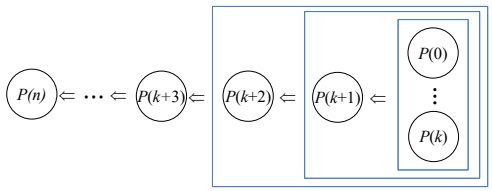
## Induzione completa vs induzione semplice

- nel caso dell'algoritmo Hanoi, si risolve il problema con *n* dischi richiamando l'algoritmo con *n-1* dischi (parametro diminuito di 1)
- verifica con *induzione semplice* con passo induttivo $P(m) \Rightarrow P(m+1)$
- per Div-Rec la soluzione con input *(a,b)* si trova richiamando l'algoritmo con input *(a-b,b)*
- parametro diminuito non di 1 ma di b: bisogna usare una forma diversa di induzione
- useremo *induzione completa*

36

## Induzione completa in generale

Caso base: $\exists k \geq 0.\, P(0) \wedge P(1) \wedge \cdots \wedge P(k)$
Passo induttivo: $\forall m \geq k.\, P(0) \wedge P(1) \wedge \cdots \wedge P(m) \Longrightarrow P(m+1)$
Conclusione: caso base e passo induttivo implicano $\forall n \geq 0.\, P(n)$




37

## Induzione completa in generale

La regola di inferenza (con caso base $P(0) \wedge \cdots \wedge P(k)$):

$$\frac{P(0) \wedge \cdots \wedge P(k) \qquad \forall m \geq k.\, P(0) \wedge \cdots \wedge P(m) \Longrightarrow P(m+1)}{\forall n.\, P(n)}$$

(sopra la linea ci sono le **premesse** e sotto c'è la **conclusione)**

38

## Induzione per divisione intera

- input: $a \geq 0, b > 0$
- output: $q$ e $r$, quoziente e resto di $a$ diviso per $b$
- caso base: se $a < b$ l'algoritmo restituisce $q = 0, r = a$ che è chiaramente corretto
- passo induttivo: dobbiamo dimostrare che
  - se $a \geq b$ e l'algoritmo è corretto per $a' = a - b < a, b$ (ipotesi induttiva)
  - allora l'algoritmo è corretto per $a, b$

39

## Induzione per divisione intera

- dimostrazione del passo induttivo del lucido precedente:
  - se per $a' = a - b, b$ l'algoritmo restituisce $q', r'$ allora per $a, b$ restituisce $q = q' + 1, r = r'$
  - secondo l'ipotesi induttiva la risposta $q', r'$ con input $a', b$ è corretta, quindi:

$$a - b = b\,q' + r \wedge 0 \leq r < b \quad \text{ipotesi ind.}$$

$$\begin{aligned} a &= b\,q' + b + r \\ &= b(q' + 1) + r \end{aligned}$$

  - e quindi $a = bq + r$ con $0 \leq r < b$ e quindi $q, r$ è la risposta giusta per $a, b$

40

## Divisione intera

```
DIV-REC(a, b)
    ▷ a ≥ 0, b > 0
if a < b then
    q, r ← 0, a
else    ▷ a ≥ b
    q', r ← DIV-REC(a − b, b)
        ▷ a − b = b q' + r ∧ 0 ≤ r < b
    q ← q' + 1
        ▷ q = q' + 1
end if
    ▷ a = b q + r ∧ 0 ≤ r < b
return q, r
```

*a<b ∧ q=0 ∧ r=a ⟹ a=bq+r (a = 0 + a)*

quindi $a - b \geq 0$

$a - b < a$

$q = q' + 1$

41

## Divisione intera

```
DIV-REC(a, b)
    ▷ a ≥ 0, b > 0
if a < b then
    q, r ← 0, a
else    ▷ a ≥ b
    q', r ← DIV-REC(a − b, b)
        ▷ a − b = b q' + r ∧ 0 ≤ r < b
    q ← q' + 1
        ▷ q = q' + 1
end if
    ▷ a = b q + r ∧ 0 ≤ r < b
return q, r
```

*a<b ∧ q=0 ∧ r=a ⟹ a=bq+r (a = 0 + a)*

quindi $a - b \geq 0$

$a - b < a$

$q = q' + 1$

42

## Commenti

- nel caso dell'induzione semplice il passo induttivo può essere
$$P(m) \Longrightarrow P(m+1)$$
ma anche
$$P(m-1) \Longrightarrow P(m)$$
- analogamente per l'induzione completa abbiamo due forme:
$$P(0) \wedge P(1) \ldots \wedge P(m) \Longrightarrow P(m+1)$$
$$P(0) \wedge P(1) \ldots \wedge P(m-1) \Longrightarrow P(m)$$

43

## Esercizio

- determinare cosa calcola il seguente algoritmo e verificare la sua correttezza

```
Rec-Calc(x)
    Pre: x ≥ 0
    Post: ...
    if x=0
        return 0
    else
        return Rec-Calc(x-1)+2x-1
```

44

## Esercizio

- per capire cosa calcola l'algoritmo conviene simularlo:
  con $x$=0 restituisce 0
  con $x$=1 restituisce 1
  con $x$=2 restituisce 4
  con $x$=3 restituisce 9
- sembra calcolare il quadrato di $x$ ma dobbiamo dimostrarlo

45

## Esercizio

- procediamo con induzione semplice (parametro diminuisce di 1 in ogni giro)
- *P(n)*: l'algoritmo è corretto quando il parametro è uguale a *n*
- caso base: *P(0)*, cioè l'algoritmo restituisce correttamente il quadrato di 0
- passo induttivo: $P(n-1) \Longrightarrow P(n)$, cioè se l'algoritmo restituisce correttamente il quadrato di $n-1$ (ipotesi induttiva) allora restituisce correttamente il quadrato di $n$

46

## Esercizio

- caso base: triviale
- passo induttivo:
  abbiamo
  Rec-Calc(n) = Rec-Calc(n-1)$+2n-1$
  dove, per ipotesi induttiva,
  Rec-Calc(n-1)=$(n-1)^2 = n^2 - 2n + 1$
  quindi
  Rec-Calc(n) = $n^2 - 2n + 1 + 2n - 1 = n^2$
- abbiamo dimostrato caso base e passo induttivo, dunque l'algoritmo è corretto

47

## Divisione intera iterativa

Riprendiamo la relazione fra l'input e l'output per trovare un procedimento iterativo per risolvere il problema della divisione intera.

$$a = b\,q + r \quad \text{e} \quad 0 \le r < b$$

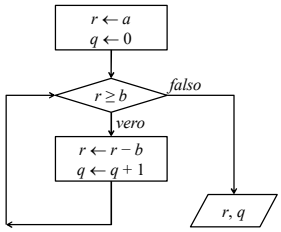
cioè

$$0 \le a \underbrace{- b - b - \ldots - b}_{q} = r < b$$

Diminuendo $a$ $q$ volte si ottiene $r$ che deve soddisfare $0 \le r < b$.

48

## Divisione iterativa




49

## Divisione iterativa

DIV-IT$(a, b)$
▷ Pre: $a \geq 0, b > 0$
▷ Post: ritorna $q, r$ tali che $a = bq + r \wedge 0 \leq r < b$
$r \leftarrow a$
$q \leftarrow 0$
**while** $r \geq b$ **do**
$r \leftarrow r - b$
$q \leftarrow q + 1$
**end while**
**return** $q, r$

Come si ragiona su di un ciclo?

50

## Invariante di ciclo

- Un ***invariante*** è una proposizione che esprime una proprietà delle variabili che resta sempre vero in vari punti dell'algoritmo:
  - ***inizializzazione***: la proposizione vale immediatamente prima di entrare nel ciclo
  - ***mantenimento***: se la proposizione vale prima di eseguire il corpo del ciclo, allora vale anche dopo aver eseguito il corpo del ciclo
- l'invariante ***deve essere utile***: insieme alle condizioni che si hanno all'uscita dal ciclo deve implicare qualcosa **per la correttezza** dell'algoritmo

51

## Divisione iterativa

```
DIV-IT(a, b)
      ▷ a ≥ 0, b > 0
r ← a
q ← 0
      ▷ a = bq + r ∧ 0 ≤ r
while r ≥ b do
          ▷ a = bq + r ∧ 0 ≤ r
    r ← r − b
    q ← q + 1
          ▷ a = bq + r ∧ 0 ≤ r
end while

return q, r
```


52

## Divisione iterativa

- l'invariante: $a = bq + r \wedge 0 \leq r$
- ***inizializzazione:***
  - prima di iniziare il ciclo $r=a$ e $q=0$
  - visto che $a \geq 0$ e $b > 0$:

    $a = bq + r = b \cdot 0 + a = a \wedge 0 \leq r = a$ è vero

  - inizializzazione ok!

53

## Divisione iterativa

- l'invariante: $a = bq + r \wedge 0 \leq r$
- ***mantenimento:***
  - denotiamo con $q'$ e $r'$ i valori aggiornati di $q$ e $r$ dopo l'esecuzione del corpo del ciclo
  - dobbiamo dimostrare che

    $a = bq + r \wedge 0 \leq r \Rightarrow a = bq' + r' \wedge 0 \leq r'$

    (cioè se vale prima allora vale anche dopo)
  - visto che il ciclo si esegue solo se $r \geq b$ e i nuovi valori sono $q' = q + 1$ e $r' = r - b$ abbiamo

    $a = bq + r = b(q + 1) + (r - b) = bq' + r' \wedge 0 \leq r' = r - b$

  - è dunque il mantenimento è garantito!

54

## Divisione iterativa

```
DIV-IT(a, b)
    ▷ Pre: a ≥ 0, b > 0
    ▷ Post: ritorna q, r tali che a = bq + r ∧ 0 ≤ r < b
r ← a
q ← 0
while r ≥ b do        ▷ inv. a = bq + r ∧ 0 ≤ r
    r ← r − b
    q ← q + 1
end while
return q, r
```

La **conseguenza** dell'invariante all'uscita:
$a = bq + r \wedge 0 \leq r \wedge \neg(r \geq b)$
cioè
$a = bq + r \wedge 0 \leq r < b$
dunque
l'algoritmo è corretto

55

## Terminazione

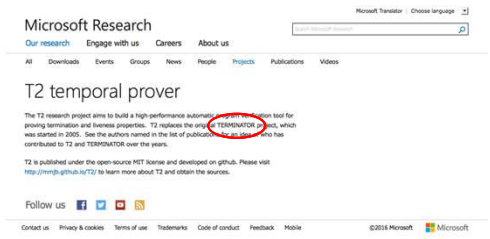
56

## Indecidibilità dell'Halt

Non esiste alcun algoritmo che, data una procedura ed un ingresso, sia in grado di decidere se la procedura termini!

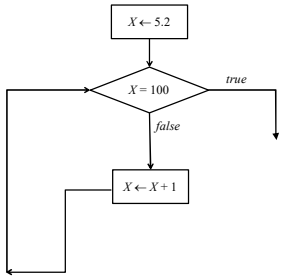
57

## Terminazione




58

## Errori logici




59

## Il problema 3*n* + 1

$$a_n = \begin{cases} \frac{1}{2} a_{n-1} & \text{se } a_{n-1} \text{ è pari} \\ 3a_{n-1} + 1 & \text{altrimenti} \end{cases}$$

Dato $a_0$ (intero positivo) trova $n$ tale che $a_n = 1$.

Esercizio: provare con $a_0 = 15$.

Risultato: 15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Congettura di Collatz: per ogni scelta di $a_0$ (intero positivo) esiste $n$ tale che $a_n = 1$.

60

## Argomenti decrescenti

Nel `fact` l'argomento della chiamata ricorsiva è sempre minore del valore del parametro:

```
int fact(int n)
{  if (n == 0) return 1;
   else return n * fact(n - 1);}
```

Nel problema 3n+1in un caso è maggiore:

```
int Collatz(int n)
{  if (n == 1) return 1;
   else if (n%2 == 0) return Collatz (n/2);
   else return Collatz (3*n + 1);}
```

61

## Argomenti decrescenti

- Per costruire una funzione ricorsiva terminante:
  - assicurarsi che qualcosa nel valore dei parametri e/o nella loro "dimensione" diminuisca
  - garantire che il valore dei parametri e/o la loro "dimensione" non può decrescere illimitatamente
  - contemplare casi minimali come *caso base*

62

## Ancora il fattoriale

Proposizione. fact(*n*) termina per ogni *n*

Caso base: fact(0) termina (e uguale a 1).

Passo induttivo: se fact(*n-1*) termina allora fact(*n*) termina.

L'algoritmo ricorsivo che calcola il fattoriale termina per tutte le entrate (non negative).

63

## Induzione sulla dimensione

Spesso, quando si definisce una funzione ricorsiva sugli array, si fa uso dell'induzione sull'ampiezza degli intervalli considerati, la loro dimensione:

$A[0..n-1]$ sia un array di n elementi.

$A[i..j]$ è l'array $(A[i], A[i+1], \ldots, A[j])$ di $j - i + 1$ elementi.

Se $i > j$ allora $A[i..j] = \varnothing$ (l'array con 0 elementi).

64

## Ricerca dicotomica

```
BinSearch-Ric(x, A, i, j)
    ▷ Pre: A[i..j] ordinato
    ▷ Post: true se x ∈ A[i..j]
if i > j then        ▷ A[i..j] = ∅
    return false
else
    m ← ⌊(i + j)/2⌋
    if x = A[m] then
        return true
    else
        if x < A[m] then
            return BinSearch-Ric(x, A, i, m − 1)
        else      ▷ A[m] < x
            return BinSearch-Ric(x, A, m + 1, j)
        end if
    end if
end if
```

L'intervallo è dimezzato

L'intervallo è dimezzato

65