Algoritmi e Strutture Dati

Lezione 6

10 ottobre 2022

ALGORITMO A $\quad f: \mathbb{N} \to \mathbb{N}$

A lavora in tempo $O(f(n))$ se

$\forall$ input I con $|I| = n$, A risolve I in $O(f(n))$

$O(f(n))$ è sufficiente per risolvere A

A richiede tempo $\Omega(f(n))$ se

$\forall n$ $\exists$ input I con $|I| = n$ t.c.

A per risolvere I impiega tempo almeno $\Omega(f(n))$

$\Omega(f(n))$ è necessario nel caso peggiore

Problema P

Un algoritmo, risolve P in tempo $O(f(n))$

$\Rightarrow$ P è risolubile in $O(f(n))$

caso peggiore

	1	2	3	4	5	...	10	..	20
n^2	1	4	9	16	25		100		400
2^n	2	4	8	16	32		1024		$> 1.000.000$

Tempo polinomiale rispetto a tempo esponenziale

- Algoritmi polinomiali
  Gli algoritmi che lavorano in tempo limitato da un polinomio (rispetto alla lunghezza dell'input)
  sono considerati "ragionevoli" o "praticabili"

- Algoritmi esponenziali
  Gli algoritmi che utilizzando tempo esponenziale
  sono considerati "impraticabili"

Tempo polinomiale rispetto a tempo esponenziale

Tempo utilizzato da un computer che esegue 1 miliardo di operazioni al secondo (pertanto ogni operazione impiega 1nsec)

	$n = 10$	$n = 20$	$n = 50$	$n = 60$	$n = 100$
n	10 10 nsec	20 20 nsec	50 50 nsec	60 60 nsec	100 100 nsec
n^2	100 100 nsec	400 400 nsec	2 500 2.5 μsec	3 600 3.6 μsec	10 000 10 μsec
n^3	1 000 1 μsec	8 000 8 μsec	125 000 125 μsec	216 000 216 μsec	1 000 000 1 msec
2^n	$\approx$ 1 000 1 μsec	$\approx$ 1 000 000 1 msec	$\approx 10^{15}$ 10^6 sec $\approx$ 11.5 giorni	$\approx 10^{18}$ 32 anni	$\approx 10^{30}$ $3 \cdot 10^{10}$ millenni

Righe = numero di operazioni di un algoritmo in funzione della lunghezza dell'input
Colonne = lunghezza dell'input
Celle = operazioni e tempo

nanosecondo 1 nsec $= 10^{-9}$ secondi
microsecondo 1 μsec $= 10^{-6}$ secondi
millisecondo 1 msec $= 10^{-3}$ secondi

Investire in hardware o in algoritmi?

- Vengono prodotti computer sempre più veloci
- A parità di tempo t, come cresce la dimensione delle istanze che possono essere risolte usando un computer più veloce?

Tempo utilizzato da un dato algoritmo	Massima lunghezza di un'istanza risolubile in un tempo fissato t		
	da un computer attuale	da un computer 100 volte più veloce	da un computer 1000 volte più veloce
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
2^n	N_4	$N_4 + 6.64$	$N_4 + 9.97$

Gli algoritmi che utilizzano tempo esponenziale non traggono alcun beneficio significativo dall'hardware più veloce!

Strutture indicizzate (array)

Array

Collezione di elementi dello stesso tipo, ciascuno dei quali è accessibile in base alla posizione

Caratteristiche tipiche:

- Memorizzato in una *porzione contigua di memoria*
- Accesso mediante *indice* (posizione)
- Tempo di accesso *indipendente* dalla posizione del dato

Limitazione:

- Struttura statica: non è possibile aggiungere nuove posizioni

Nota:

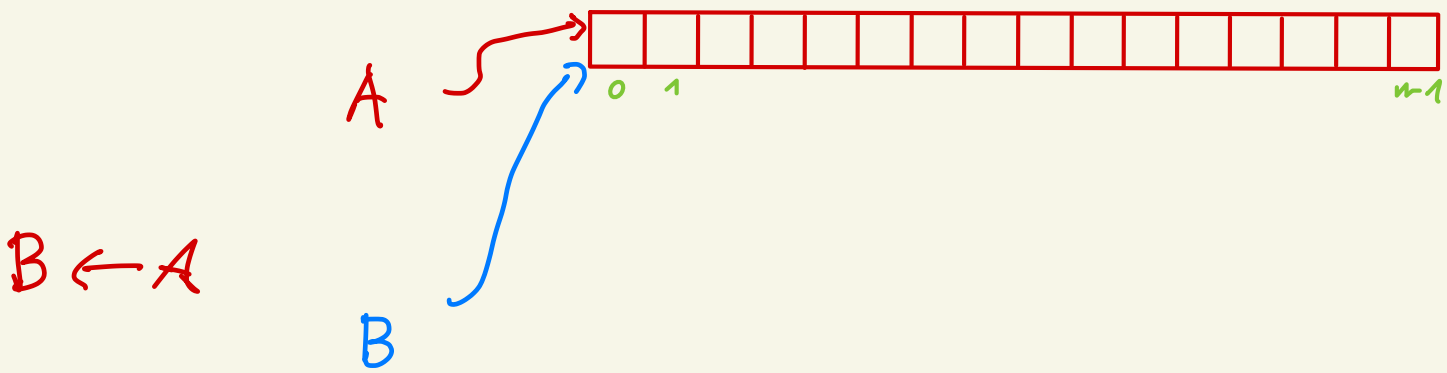
nei singoli linguaggi di programmazione (es. Go) alcune caratteristiche degli array e delle relative variabili possono essere differenti

Array

Collezione di elementi dello stesso tipo, ciascuno dei quali è accessibile in base alla posizione

Variabili array:

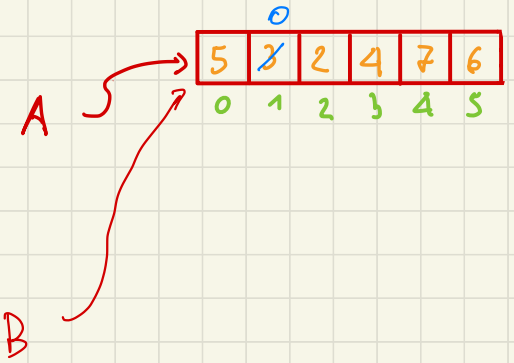
- Sono *riferimenti (puntatori)* all'array



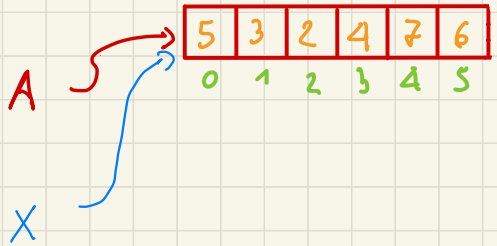
$B \leftarrow A$

$B[1] \leftarrow 0$

stampa $(A[1])$



```
PROCEDURA azzera (Array X[0..n-1])
    FOR i ← 0 TO n-1 DO
        X[i] ← 0
```



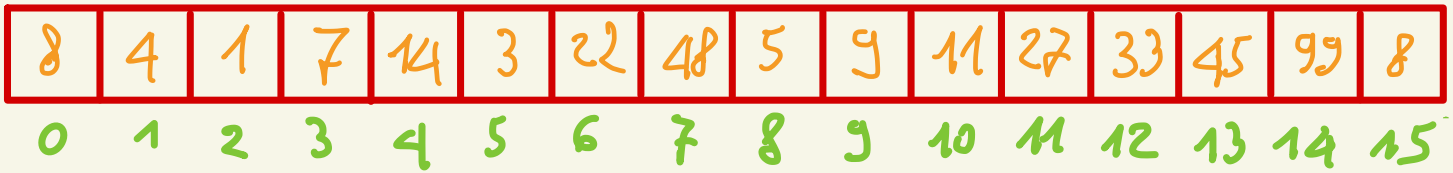

```
ALGORITMO ...
    ⋮
    azzera (A)
    ⋮
```

Ricerca in un array

Ricerca in un array

input Array A, elemento x

output Indice i t.c. $A[i] = x$
-1, se A non contiene x



RICERCA SEQUENZIALE

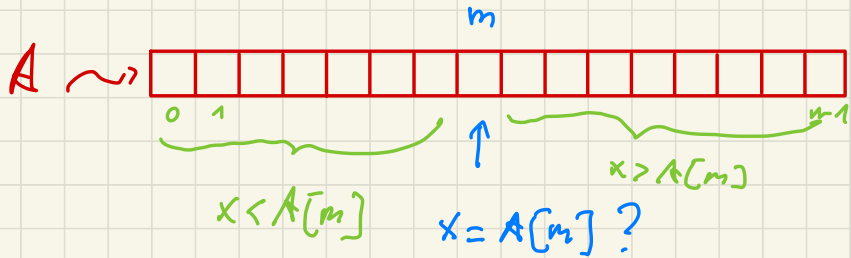
```
ALGORITMO ricercaSequenziale (Array A[0..n-1], elemento x) -> indice
    i ← 0
    WHILE  i<n  AND  A[i] ≠ x  DO
        i ← i+1
    IF  i = n  THEN  RETURN -1
               ELSE  RETURN i
```

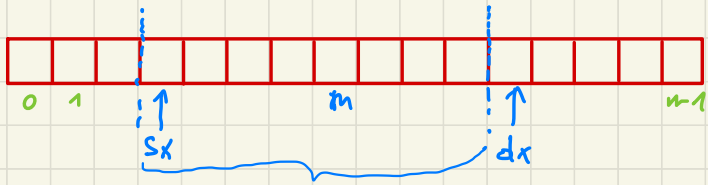
lazy evaluation

```
    i ← n-1
    WHILE  i≥0  AND  A[i] ≠ x  DO
        i ← i-1
    RETURN i
```

Tempo $\Theta(n)$ se il cfr $A[i] \neq x$ è in tempo costante (es. interi)

RICERCA IN ARRAY ORDINATO



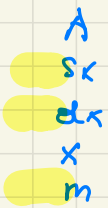



```
FUNZIONE ricercaRic(Array A, indice sx, indice dx, elemento x)
                                                  → indice
    IF dx ≤ sx THEN RETURN -1
    ELSE
        m ← (sx+dx)/2        // div intera
        IF x = A[m] THEN RETURN m
        ELSE IF x < A[m] THEN
                RETURN ricercaRic(A, sx, m, x)
        ELSE
                RETURN ricercaRic(A, m+1, dx, x)

ALGORITMO ricercaBinaria(Array A[0..n-1],
          elemento x) → indice
    RETURN ricercaRic(A, 0, n, x)
```

```
FUNZIONE ricercaRic (Array A, indice sx, indice dx, elemento x)
                                                    → indice
  IF dx ≤ sx THEN RETURN -1
  ELSE
     m ← (sx+dx)/2   // div intera
     IF x = A[m] THEN RETURN m
     ELSE IF x < A[m] THEN
              RETURN ricercaRic (A, sx, m, x)
          ELSE
              RETURN ricercaRic (A, m+1, dx, x)
```

Record di attivazione



A →	1	5	7	12	16	18	20	22
	0	1	2	3	4	5	6	7

x 12

ricercaRic (A, 0, 8, 12) → 3

sx	0
dx	8
m	4

ricercaRic (A, 0, 4, 12) → 3

sx	0
dx	4
m	2

ricercaRic (A, 3, 4, 12) → 3

sx	3
dx	4
m	3

x = 11 ?

$A[0..n-1]$ n elementi

1ª chiamata
$sx = 0$ $dx = n$ spazio di ricerca n

2ª chiamata $dx - sx \leq \frac{n}{2}$ $\leq \frac{n}{2}$

⋮

i-esima chiamata $dx - sx \leq \frac{n}{2^{i-1}}$ $\frac{n}{2^{i-1}}$

$\frac{n}{2^{i-1}} \stackrel{?}{=} 1 \rightarrow n = 2^{i-1}$

$i - 1 = \log_2 n$

$i = 1 + \log_2 n$

ulteriore chiamata
$\rightarrow$ 0 elementi

$\Rightarrow$ al più $2 + \log_2 n$ chiamate

ESERCIZIO: es precedente trovare x t.c. # chiamate sia $2 + \log_2 n$