Algoritmi e Strutture Dati

Lezione 18

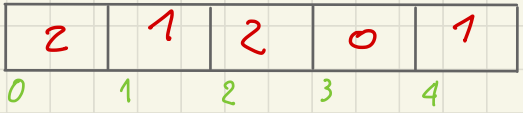
9 novembre 2022

Ordinare senza confrontare

Problema: ordinare n interi in $[0..b-1]$

A

Y

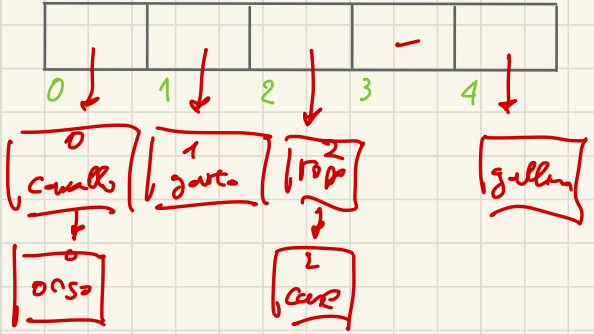


integerSort

Problema: ordinare n record con chiavi intere in $[0..b-1]$

A

X

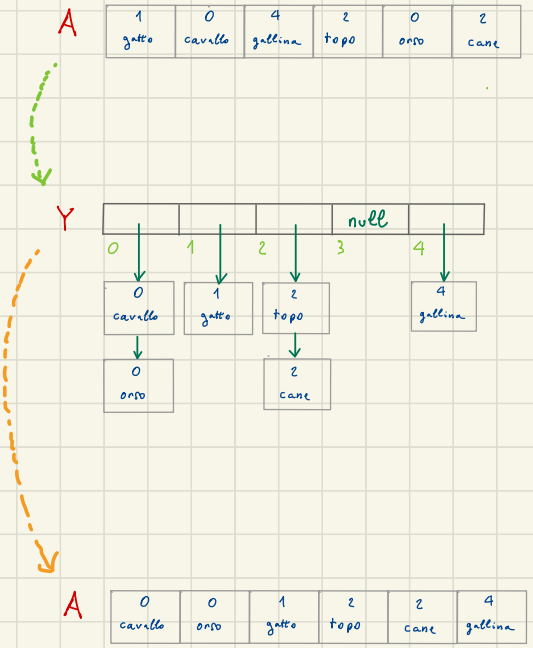


bucketSort

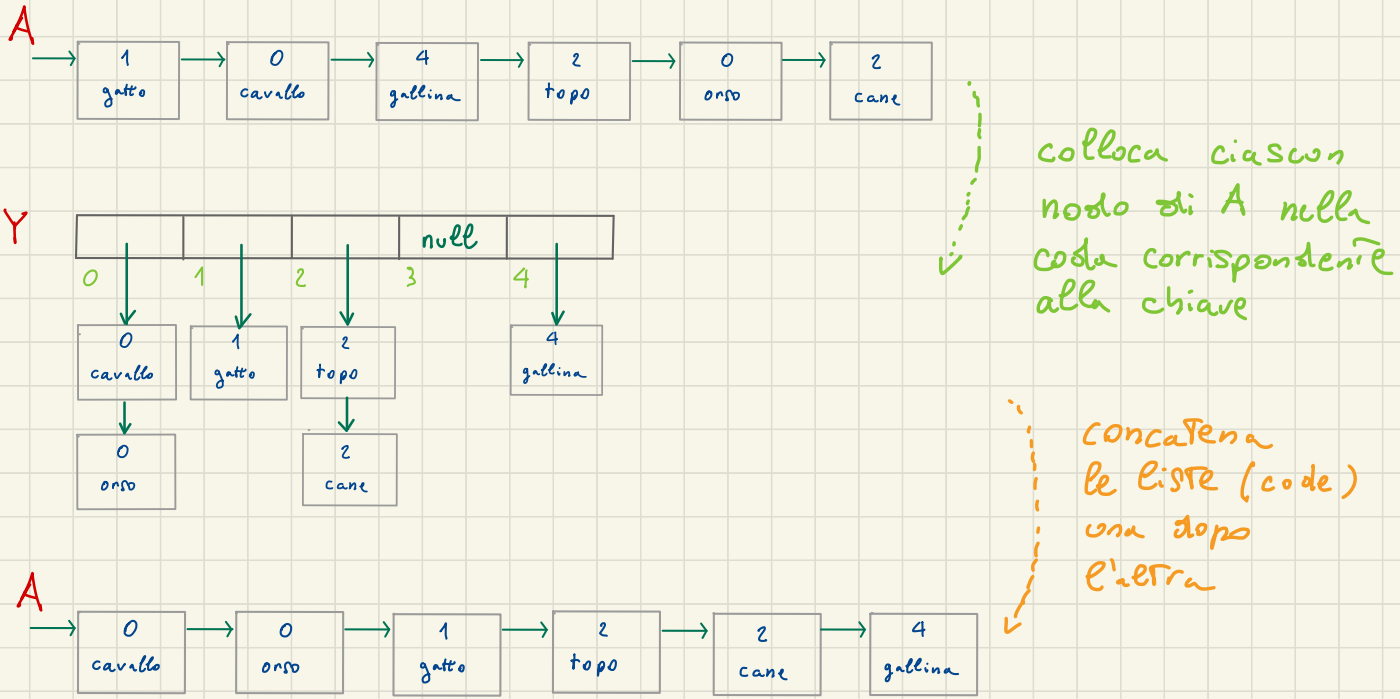
Problema: ordinare n record con chiavi intere in $[0..b-1]$

```
ALGORITMO bucketSort(Array A[0..n-1], intero b)
    Sia Y[0..b-1] un array                          predisposizione bucket
    FOR i ← 0 TO b-1 DO                             Θ(b) passi
        Y[i] ← coda vuota
    FOR i ← 0 TO n-1 DO                             riempimento bucket
        x ← A[i].chiave                             Θ(n) passi
        Y[x].enqueue(A[i])
    j ← 0
    FOR i ← 0 TO b-1 DO                             b iterazioni
        WHILE NOT Y[i].isEmpty() DO                 riempimento array ordinato
            A[j] ← Y[i].dequeue()                   Θ(b+n) passi
            j ← j+1
```

$\Theta(b+n)$



BucketSort può essere utilizzato anche per ordinare liste manipolando direttamente i puntatori



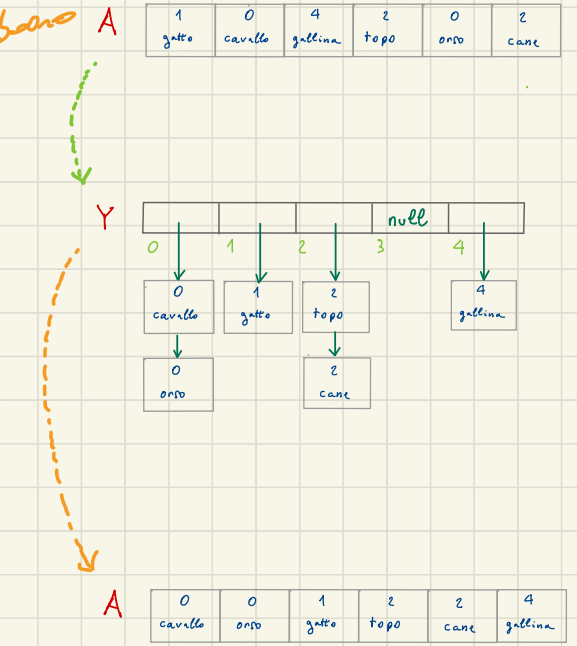
BucketSort

- Tempo $\Theta(n+b)$

  → se $b = O(n)$ Tempo $\Theta(n)$

  → se $b = \Theta(n^2)$ Tempo $\Theta(n^2)$

meglio usare heapSort



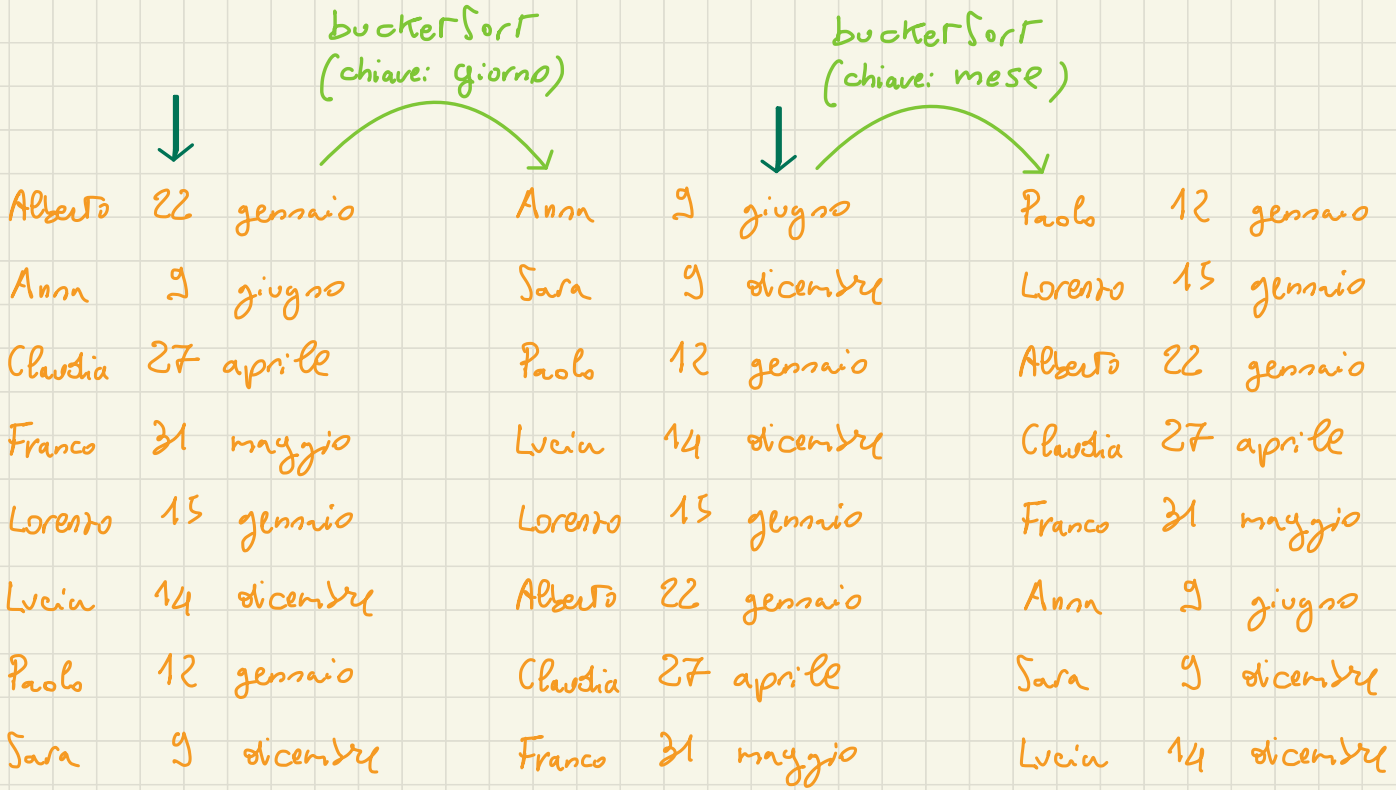
- NON IN LOCO
- Stabile
- Utilizzabile anche per liste ordinate (stessa complessità in tempo)

Problema: ordinare un insieme di persone rispetto alla data del compleanno

Alberto	22	gennaio
Anna	9	giugno
Claudia	27	aprile
Franco	31	maggio
Lorenzo	15	gennaio
Lucia	14	dicembre
Paolo	12	gennaio
Sara	9	dicembre

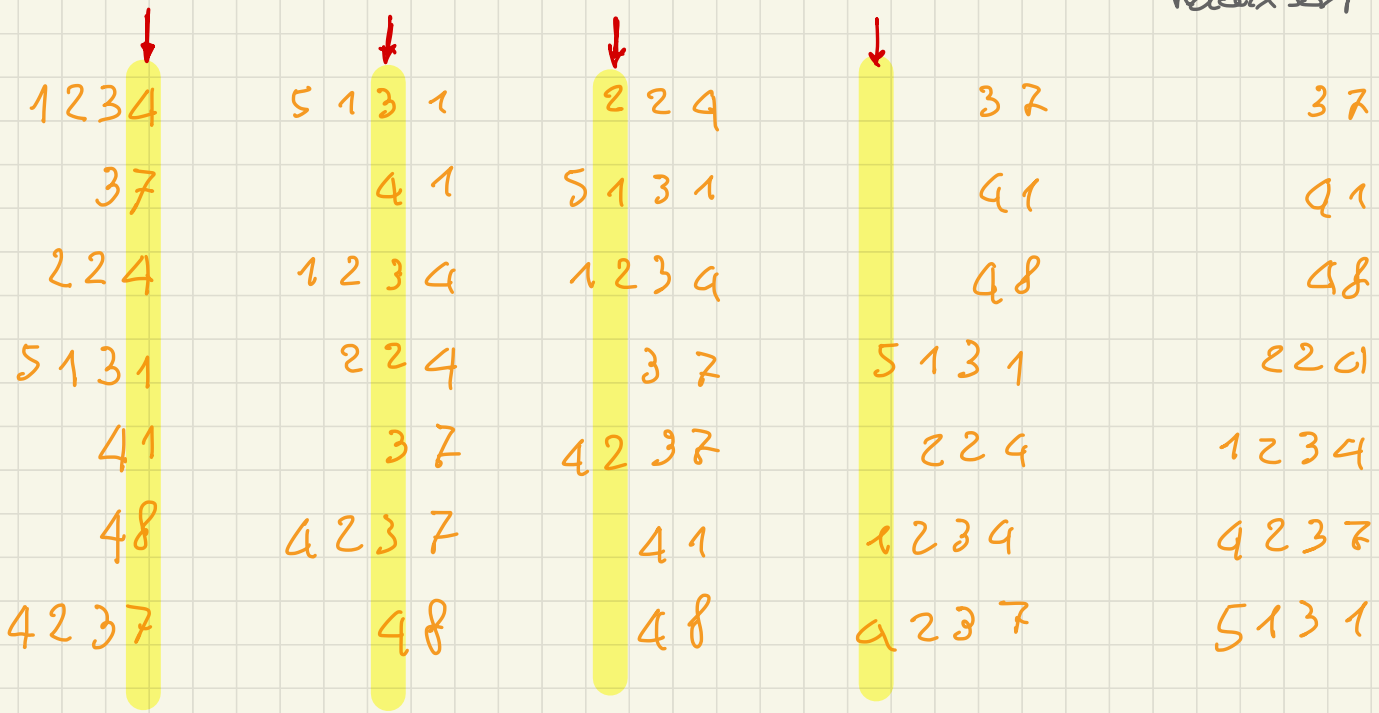
1. bucketSort su giorni
2. bucketSort su mesi

Problema: ordinare un insieme di persone rispetto alla data del compleanno



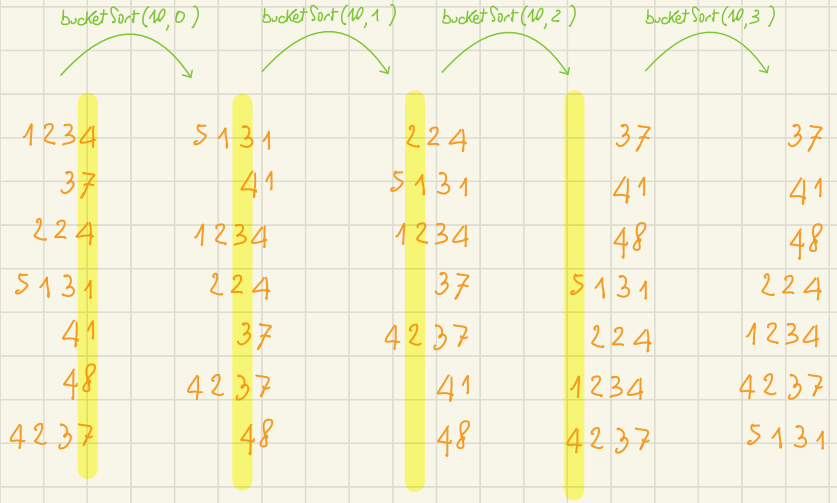
Problema: ordinare un insieme di dati con chiavi intere

RadixSort



```
ALGORITMO radixSort (Array A[0..n-1])
    t ← 0
    WHILE (∃chiave K in A t.c. ⌊K/b^t⌋ ≠ 0) DO
    |   bucketSort (A, b, t)
    |   t ← t+1
```

A contiene una chiave composta da almeno t+1 cifre




```
PROCEDURA bucketSort (Array A[0..n-1], intero b, intero t)
    Sia Y[0..b-1] un array                          | predisposizione
    FOR i ← 0 TO b-1 DO                             | bucket
    |   Y[i] ← coda vuota

    FOR i ← 0 TO n-1 DO                             | riempimento
    |   x ← cifra di posto t nella                  | bucket
    |       rappresentazione in base b
    |       di A[i].chiave
    |   Y[x].enqueue (A[i])

    j ← 0                                           | riempimento
    FOR i ← 0 TO b-1 DO                             | array
    |   WHILE NOT Y[i].isEmpty() DO                 | ordinato
    |   |   A[j] ← Y[i].dequeue()
    |   |   j ← j+1
```

RadixSort

- Se devo ordinare n chiavi tra 0 e 999.999.999:

  con $B = 10^3$ bastano 3 passate di bucketSort

Meglio usare B potenza di 2 (divisioni più veloci!)

- Se devo ordinare n chiavi su 32 bit, cioè tra 0 e $2^{32} - 1$

  - con $B = 2^8$: (256 bucket) 4 passate di bucketSort
  - con $B = 2^{16}$: 2 passate di bucketSort

Tempo $\Theta(n)$

Come ottenere la cifra di posizione* t di x in base b?

es base 10, $t=2$

$x = 1234$

$x = 12345$ ⇝ $12345/10^2$

1 2 3

$(x/10^2) \text{ MOD } 10$ (divisione intera)

in generale: $(x/b^t) \text{ MOD } b$ (divisione intera)

* contando da 0 dalla meno significativa

Come ottenere la cifra di posizione t di x in base b?

$$(x / b^t) \text{ MOD } b$$

es base 16, $t = 2$

$x = 9999$

$$(9999 / 16^2) \text{ MOD } 16 = 7$$

in binario:

x 0010 0111 0000 1111 $\gg 8 =$ 0010 0111 $x / 16^2$

$16^2 = (2^4)^2 = 2^8$

shift a dx di 8 posizioni

0010 0111 & AND

0000 1111 $= 2^4 - 1$

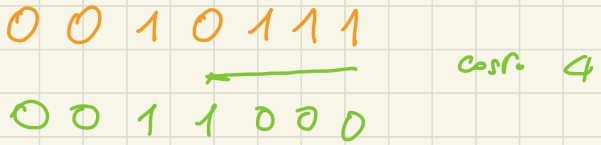
0000 0111

$(x / 16^2) \text{ MOD } 16$

Analisi Ammortizzata

Quanto costa incrementare un contatore a n bit?

costo ≡ n° di bit da ispezionare



```
ALGORITMO incrementaContatore (Array v[0..n-1] di bit)
    i ← 0
    WHILE i<n AND v[i]=1 DO
    |   v[i] ← 0
    |   i ← i+1
    IF i<n THEN v[i] ← 1
```

Caso peggiore costo $\Theta(n)$

k incrementi $O(kn)$

Esempio

0	000000	
1	000001	1
2	000010	2
3	000011	1
4	000100	3
5	000101	1
	⋮	

Il bit in posizione i cambia ogni 2^i incrementi

$$T(n,k) = k + \left\lfloor \frac{k}{2} \right\rfloor + \left\lfloor \frac{k}{2^2} \right\rfloor + \cdots + \left\lfloor \frac{k}{2^{n-1}} \right\rfloor$$

$$= \sum_{i=0}^{n-1} \left\lfloor \frac{k}{2^i} \right\rfloor \leq \sum_{i=0}^{n-1} \frac{k}{2^i} = k \sum_{i=0}^{n-1} \frac{1}{2^i} \leq k \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

$$= k \frac{1}{1 - \frac{1}{2}} = k \frac{1}{\frac{2-1}{2}} = 2k$$

in media

$$T_{a,n}(k) = \frac{T(n,k)}{k} = \frac{2k}{k} = 2$$

costo ammortizzato

ESPONENZIALE ITERATO

$$F(k) = \begin{cases} 1 & \text{se } k = 0 \\ 2^{F(k-1)} & \text{se } k \geq 1 \end{cases}$$

$F(0) = 1$

$F(1) = 2^{F(0)} = 2$

$F(2) = 2^{F(1)} = 2^2 = 4$

$F(3) = 2^{2^2}$

$F(4) = 2^{2^{2^2}}$

$\vdots$

$F(n) = 2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}$ } n volte

LOGARITMO ITERATO

$$\lg^{(1)} n = \lg_2 n$$

$$\lg^{(2)} n = \lg_2 (\lg^{(1)} n) = \lg_2 \lg_2 n$$

$$\vdots$$

$$\lg^{(k)} n = \lg_2 (\lg^{(k-1)} n) = \underbrace{\lg_2 \lg_2 \cdots \lg_2}_{k \text{ volte}} n$$

$$\lg^* n = \{ \min k \mid \lg^{(k)} n \leq 1 \}$$

$\lg^* 1 = 0$

$\lg^* 2 = 1$

$\lg^* 3 = 2$

$\lg^* 4 = 2$

$\lg^* x = 3 \quad 4 < x \leq 2^4 = 16$

$\lg^* x = 4 \quad 16 < x \leq 2^{16}$

$\lg^* x = 5 \quad 2^{16} < x \leq 2^{2^{16}}$

$\vdots$