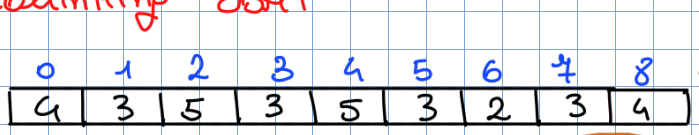
Ordinamento lineare

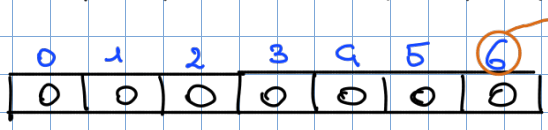
- HeapSort $O(n \log n)$
  $\Omega(n \log n)$
  $\Theta(n \log n)$

usando le comparazioni non esiste un algoritmo migliore

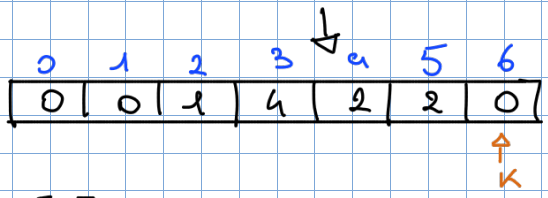
non tutti gli algoritmi necessitano di confronti

Counting Sort



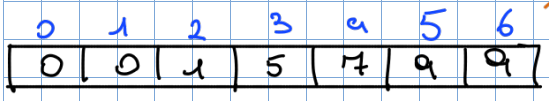


conta le occorrenze degli indici in A



$C[i]$ = numero di occorrenze di i in A $\quad 0 \le i \le K$

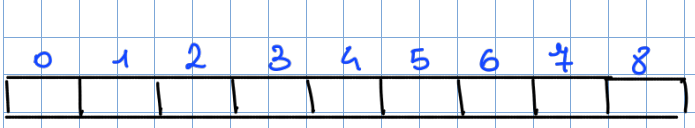
C'



abbiamo fatto

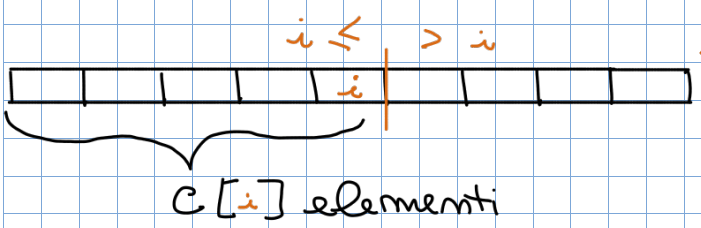
```
for i ← i to K do
    C[i] ← C[i] + C[i-1]
```

$C'[i]$ = numero di elementi in A minori o uguali ad i



$i = 3$

Prendiamo il 3 come esempio



l'elemento i va in posizione $C[i]-1$

Facendo queste cose per ogni elemento di C' avremo un array ordinato

CountingSort (A, n):

k ← max(A)

C ← new Array(k+1)

For i ← 0 to k Do C[i] ← 0

For i ← 1 to n-1 Do C[A[i]] ← C[A[i]] + 1 → C'

For i ← n-1 Do (inutile se scambiamo le due istruzioni)

B[C[A[i]] -1] ← A[i] → *

C[A[i]] ← C[A[i]] -1 →

facciamo queste cose per il caso di duplicati, se non ci fossero i duplicati sarebbero tutti nello stesso posizione

Radix Sort

$O(n+k)$ $k \in O(n) \Rightarrow O(n)$

↳ scorriamo tutti i valori del nostro array

Scriviamo un array in colonne

A
329
457
657
839
436
720
355

→ ordino rispetto alle prime cifre →

A
329
355
457
436
657
720
839

→ ordino rispetto alle seconde cifre →

A
329
355
436
457
657
720
839

↳ h cifre

ad occhio si fa me do gestire in un algoritmo è difficile

A
329
457
657
839
436
720
355

rispetto alle terze cifre

A
329
457
657
839
436
720
355

rispetto alle seconde

A
720
329
436
839
355
457
657

rispetto alle prime

A
329
355
436
457
657
720
839

l'algoritmo diventa stabile e anche più semplice da applicare

RadixSort(A, m, h)

for i ← 0 to h-1 DO ← cifre da controllare

countingSort(A, m, h)

A[i] % 10
(A[i]/10) % 10
} le cifre si estrae così

↓

(A[i] / 10^h) % 10

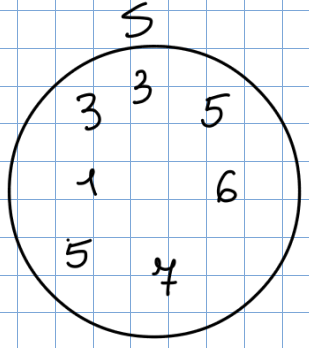
$O(h(m+k)) \in O(m)$

// Preavere e cose in confronto con altri

Dizionario (che strutture usiamo per implementarlo?)

	BST	WORST	AVG
INS	$\Theta(\log_m)$	$O(1)$	$O(1)$
DEL	$\Theta(\log_m)$	$O(1)$	$O(1)$
RIC	$\Theta(\log_m)$	$O(m)$	$O(1)$

Tabelle di indirizzamento diretto



T ↘

0	0
1	1
2	0
3	2
4	0
5	2
6	1
7	1
8	0

→ semplice e C del CountingSort

Insert(t, k)

t[k] ← t[k] + 1

Delete(t, k)

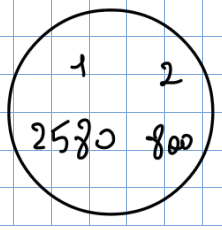
t[k] ← MAX(0, t[k] - 1)

Search(t, k)

If (t[x] ≥ 1) Return 1

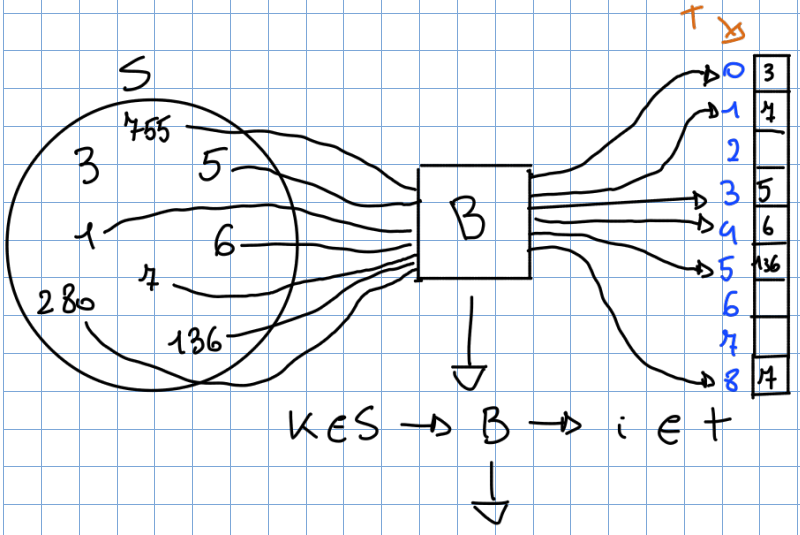
return

Non efficiente se la diff tra max e min è troppo alto infatti avremo tante celle vuote



0	0
1	1
2	1
⋮	
800	1
⋮	
2580	1

Vediamo le caratteristiche di una tabella di indirizzamento diretto ma con una struttura che consumi il giusto numero di celle



$k \in S \rightarrow B \rightarrow i \in t$

si occupa di indirizzare

Insert (t, k)
$t[B(k)] \leftarrow k$

Delete (t, k)
$t[B(k)] \leftarrow$ null

Search (t, k)
return $(t[B(k)] = k)$

B deve essere deterministica
$k_1, k_2 \in N \Rightarrow B[k_1] \neq B[k_2]$

stiamo mappando un insieme infinito in uno che ha m elementi? come è possibile?

infatti si hanno delle COLLISIONI

Vediamo ora questo problema le tabelle hash