ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Algoritmi di Ordinamento

Problema di Ordinamento

Consideriamo il seguente problema

Dato un vettore di interi, e.g.:

4	2	7	9	6	3	0

...Ne si vuole ordinare in senso crescente il contenuto:

0	2	3	4	6	7	9

Il problema può essere generalizzato:

- La struttura dati potrebbe non essere un array (e.g. una lista)
- Anziché interi, potremmo avere altri oggetti (e.g. float, date)
  - Basta esista una relazione d'ordine

Problema di Ordinamento

Il problema di ordinamento è ben studiato

Esistono diversi algoritmi per risolverlo. I principali sono:

- Naive sort
- Bubble sort
- Insertion sort
- Merge sort
- Quick sort

Tali algoritmi:

- Hanno efficienza diversa e sono indicati per casi diversi
- ...Sono un buon esempio per una analisi di complessità

Problema di Ordinamento

Vogliamo definire una libreria per ordinare vettori di interi

Nel file "sort.h", esporremo le dichiarazioni:

```
int naive_sort(int v[], int n);
int bubble_sort(int v[], int n);
int insertion_sort(int v[], int n);
int merge_sort(int v[], int n);
int quick_sort(int v[], int n);
```

Ogni funzione:

- Riceve in ingresso un vettore e la sua dimensione logica
- Ordina il vettore
- Restituisce il numero di confronti effettuati
- ...Che per questo problema è un buon indicatore di efficienza

Problema di Ordinamento

Vogliamo definire una libreria per ordinare vettori di interi

Nel file "sort.h", esporremo le dichiarazioni:

```
int naive_sort(int v[], int n);
int bubble_sort(int v[], int n);
int insertion_sort(int v[], int n);
int merge_sort(int v[], int n);
int quick_sort(int v[], int n);
```

- Discuteremo solo alcuni degli algoritmi
- ...Ma il codice completo della libreria sarà comunque fornito
- Gli interessati agli algoritmi rimanenti potranno:
  - Cercare materiale online
  - Analizzare il codice fornito

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Naïve Sort

Naïve Sort

Naïve sort è uno degli algoritmi di ordinamento più semplici

Dato il vettore di ingresso, e.g.:

4	2	7	9	6	3	0

- Consideriamo le posizioni una alla volta:

4	2	7	9	6	3	0

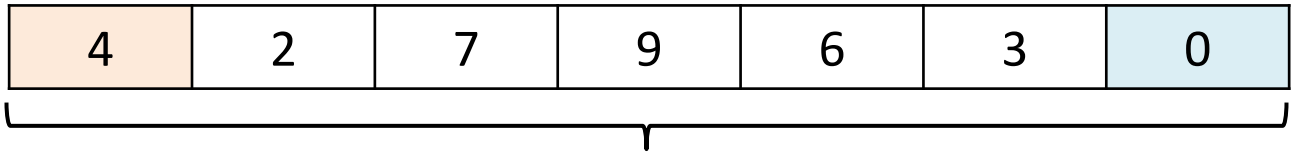
- Consideriamo tutti gli elementi a dx (incluso quello corrente)

4	2	7	9	6	3	0

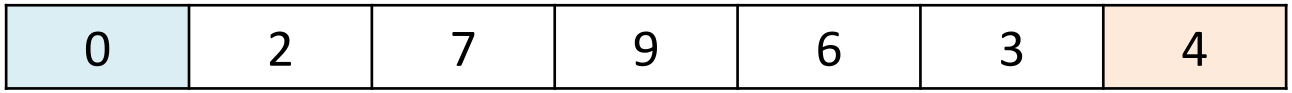
Naïve Sort

Quindi:

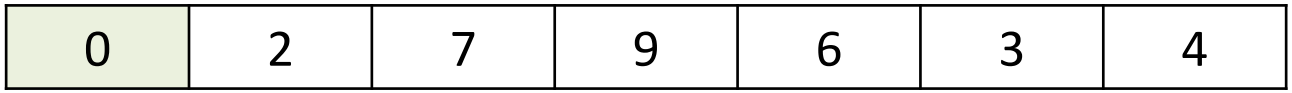
- Individuiamo l'elemento più piccolo:



- Li scambiamo:

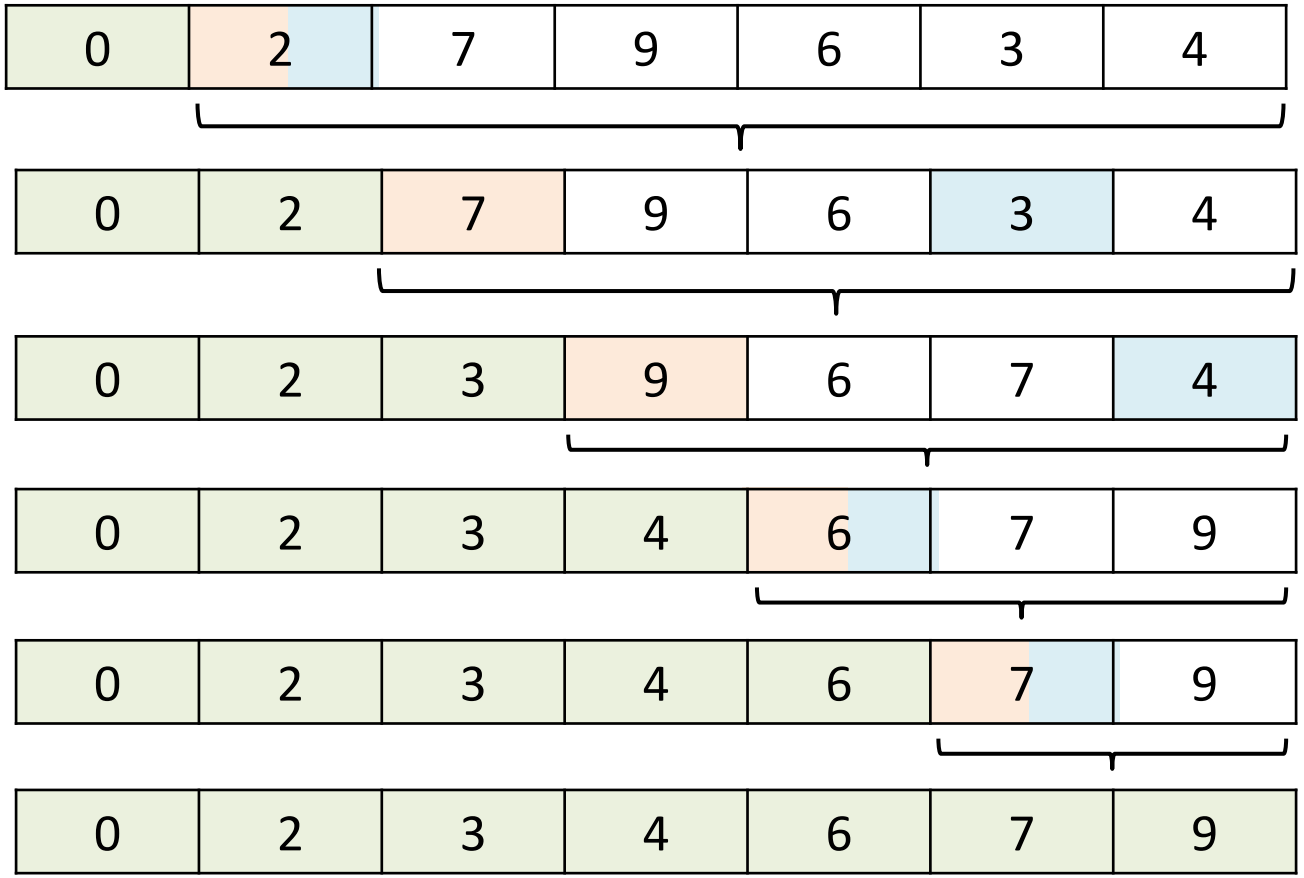


- Il primo elemento adesso è ordinato:



Naïve Sort

Ripetiamo il processo fino alla fine del vettore



Naïve Sort

Pseudo codice

- Per ogni indice "i" dell'array da 0 a "n-2"
  - Individuare l'indice "minidx" dell'elemento più piccolo a dx
  - Scambiare gli elementi in posizione "i" e "minidx"

Naïve Sort

Pseudo codice

- Per ogni indice "i" dell'array da 0 a "n-2"
  - Individuare l'indice "minidx" dell'elemento più piccolo a dx
    - Si assuma che v[i] sia l'elemento più piccolo
    - Per ogni indice "j" da "i" fino ad "n-1"
    - Se v[j] < v[i], allora v[j] è l'elemento più piccolo
  - Scambiare gli elementi in posizione "i" e "minidx"

Per il codice effettivo si vedano le soluzioni

Naïve Sort

Come valutare l'efficienza dell'algoritmo?

Primo metodo: misurare il tempo di esecuzione

- Funziona per qualunque algoritmo ☺
- Molte sorgenti di "rumore" (e.g. cache, altri processi) ☹

Secondo metodo: complessità asintotica

1. Individuare il tipo di operazione che occorre più di frequente
2. Contarne le esecuzioni in funzione della dimensione dell'input

Obiettivo: trova un asintoto, i.e. un O-grande

Naïve Sort

Complessità asintotica di naïve sort:

Individuare il tipo di operazione che occorre più di frequente:

- Per tutti gli algoritmi di ordinamento, si tratta del confronto

Contarne le esecuzioni in funzione della dimensione dell'input:

- Ricordate: ci interessa un asintoto, non il numero esatto
- "n" confronti alla prima passata, "n-1" alla seconda, etc.
- Totale: n * (n-1) / 2
- Complessità asintotica: $O(n^2)$

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Valutazione Empirica

Valutazione Empirica

La complessità asintotica:

- Si basa su una analisi teorica
- Si focalizza sul caso peggiore

Ma come si fa a valutare un algoritmo su un caso pratico?

Risposta: si può usare una valutazione empirica

Si tratta di un approccio sperimentale:

- Si implementano diversi algoritmi
- Si sceglie un insieme di valori di input (i.e. un benchmark)
- Si eseguono gli algoritmi e si misurano le loro prestazioni

Valutazione Empirica

Qualche osservazione sul benchmark:

- È bene che non contenga troppo pochi ingressi
  - E.g. per poter calcolare delle medie affidabili
- È bene che sia lo stesso per tutti gli algoritmi
  - Il confronto deve essere equo!

Nel nostro caso possiamo:

- Realizzare un main che generi diversi vettori da ordinare
  - E.g. possiamo generare vettori con numeri casuali...
  - ...Ma usare un "seed" per generare sempre gli stessi
- Calcolare il numero medio di confronti per ogni algoritmo

Per un esempio di codice, si vedano le soluzioni

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Bubble Sort

Bubble Sort

Abbiamo accennato bubble sort alla prima lezione del corso

Dato il vettore di ingresso, e.g.:

4	2	7	9	6	3	0

- Consideriamo le posizioni una alla volta:

4	2	7	9	6	3	0

- Consideriamo l'elemento adiacente alla posizione corrente

4	2	7	9	6	3	0

Bubble Sort

Quindi:

- Se l'ordine è invertito, li scambiamo

2	4	7	9	6	3	0

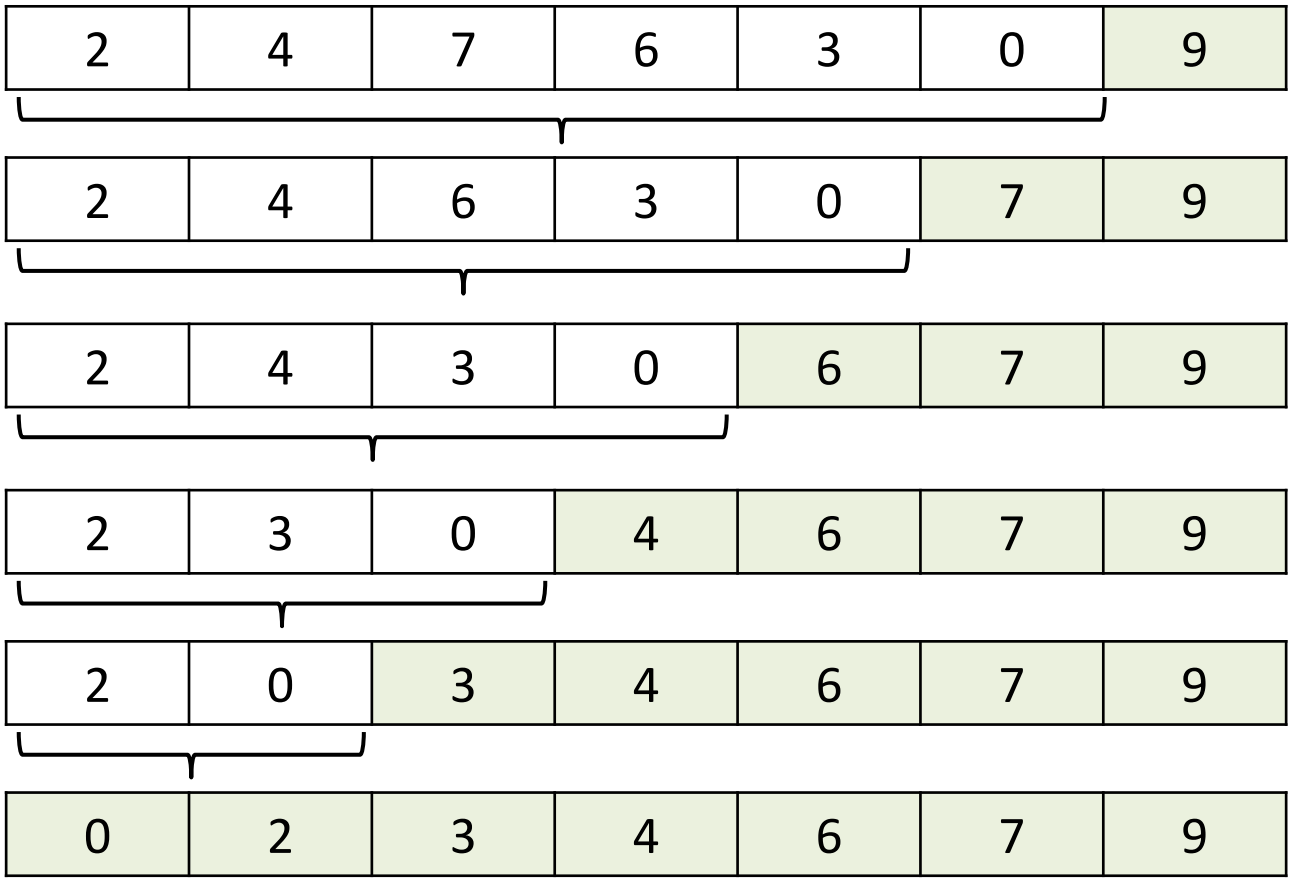
- Ripetiamo il processo: alla fine l'ultimo elemento è ordinato

2	4	7	9	6	3	0
2	4	7	9	6	3	0
2	4	7	9	6	3	0
2	4	7	6	9	3	0
2	4	7	6	3	9	0
2	4	7	6	3	0	9

Bubble Sort

Quindi:

- Ripetiamo, limitandoci ai primi n-1 elementi e così via



Bubble Sort

Pseudo codice

- Facciamo variare un limite "m" da "n-1" a 2
  - Effettuiamo una passata di scambi
  - Se alla fine di una passata non ci sono stati scambi, possiamo fermarci (l'array è ordinato)

La condizione indicata:

- Permette di interrompere il processo in anticipo
- Si tratta di una ottimizzazione potenzialmente significativa

Bubble Sort

Pseudo codice

- Facciamo variare un limite "m" da "n-1" a 2
  - Effettuiamo una passata di scambi
    - Consideriamo gli indici "i" da 0 a "m-2"
    - Se v[i] > v[i+1], li scambiamo
  - Se alla fine di una passata non ci sono stati scambi, possiamo fermarci (l'array è ordinato)

Per il codice effettivo si vedano le soluzioni

Bubble Sort

Complessità asintotica di bubble sort:

Contiamo il numero di confronti:

- Alla prima passata: n-1 confronti
- Alla seconda passata: n-2 confronti, e così via fino alla fine

Nel caso peggiore, ancora $O(n^2)$ confronti

- Se però l'array diventa ordinato prima della fine...
- ...Il processo viene interrotto e risparmiamo dei confronti!

Nel caso migliore (array ordinato): n-1 confronti

- Bubble sort funziona bene se gli elementi sono abbastanza vicini alla loro posizione corretta

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Insertion Sort

Insertion Sort

Insertion sort opera attraverso inserimenti ordinati successivi

Dato il vettore di ingresso, e.g.:

4	2	7	9	6	3	0

- La prima metà (all'inizio il primo cella) si considera ordinata

4	2	7	9	6	3	0

- Consideriamo l'elemento immediatamente successivo

4	2	7	9	6	3	0

Insertion Sort

Quindi:

- Inseriamo l'elemento nella prima metà, in modo ordinato
  - In questo caso, basta uno scambio

2	4	7	9	6	3	0

- Dopo l'inserimento, una nuova cella è ordinata e ripetiamo
  - In questo caso, nessuno scambio è necessario

2	4	7	9	6	3	0

- Di nuovo, nessuno scambio è necessario

2	4	7	9	6	3	0

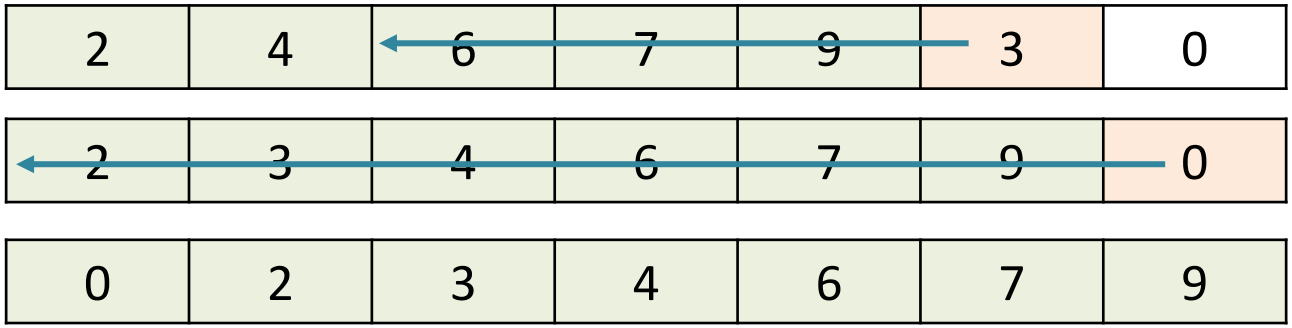
Insertion Sort

Quindi:

- Nel caso generale, effettuiamo una serie di scambi verso sx

2	4	7	9	6	3	0
2	4	7	6	9	3	0
2	4	6	7	9	3	0

- Si ripete il processo fino all'ultimo elemento:



Insertion Sort

Pseudo codice

- Consideriamo gli indici "i" da 1 fino ad "n-1"
  - Inseriamo l'elemento i-mo nella posizione corretta, nella metà sx dell'array (i.e. indici da 0 a "i-1")

Insertion Sort

Pseudo codice

- Consideriamo gli indici "i" da 1 fino ad "n-1"
  - Inseriamo l'elemento i-mo nella posizione corretta, nella metà sx dell'array (i.e. indici da 0 a "i-1")
    - Consideriamo gli indici "j" da "i" fino a 1
    - Se v[j] < v[j-1], li scambiamo

Per il codice effettivo si vedano le soluzioni

Insertion Sort

Complessità asintotica di insertion sort:

Contiamo il numero di confronti nel caso peggiore:

- Per il primo elemento: 1 confronti
- Per il secondo elemento: 2 confronti, e così via fino alla fine

Nel caso peggio, ancora $O(n^2)$ confronti

- Se però solo pochi elementi sono fuori posto...
- ...Bisogna re-inserire solo quelli!

Nel caso migliore (array ordinato): n-1 confronti

- Insertion sort funziona bene se ci sono pochi elementi fuori posto

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>