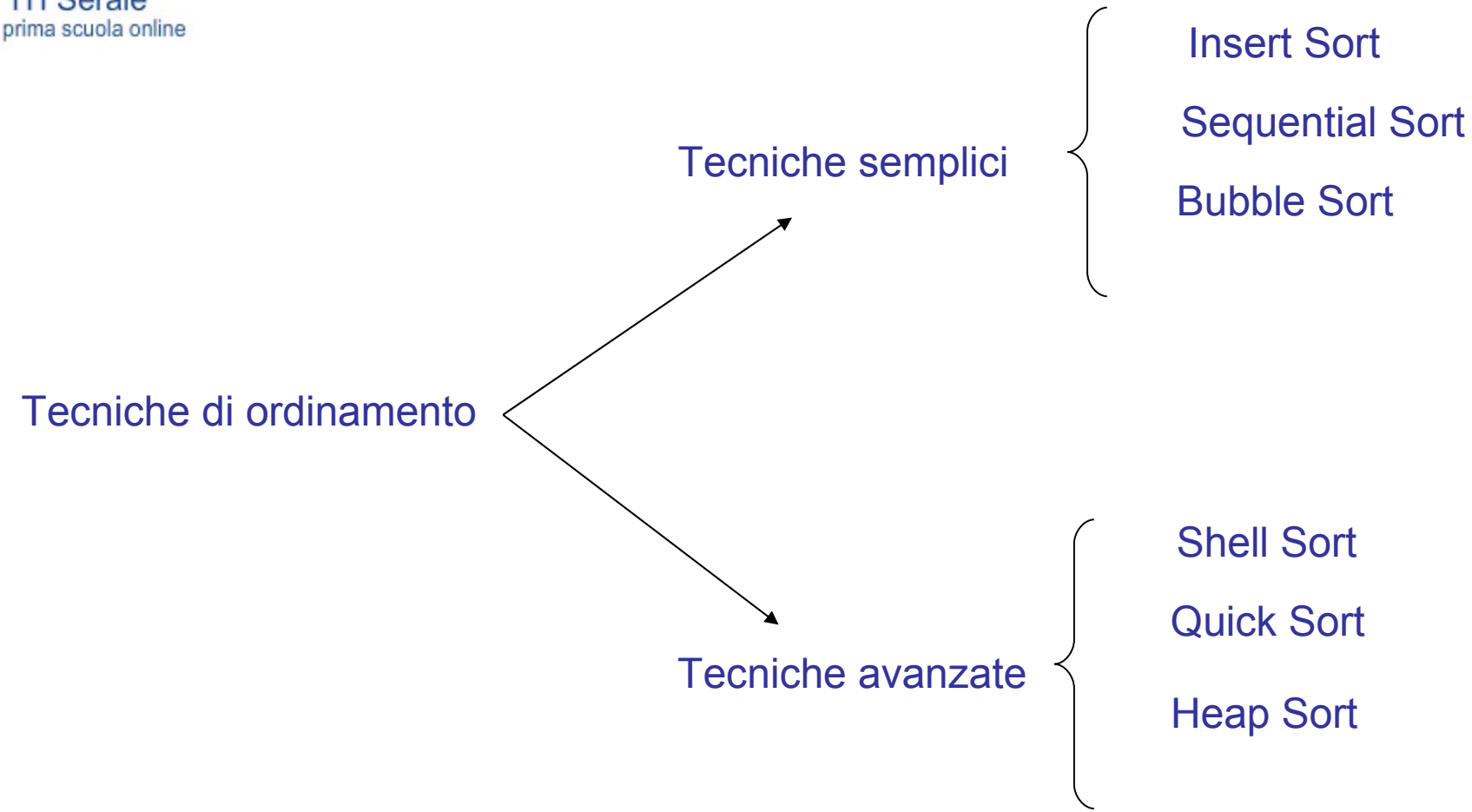
ITI Serale
La prima scuola online

Tecniche di ordinamento di un array

A cura del docente Giuliano Pellegrini Parisi - © 2009


Tecniche di ordinamento
Tecniche semplici
Insert Sort
Sequential Sort
Bubble Sort
Tecniche avanzate
Shell Sort
Quick Sort
Heap Sort

Le differenze tra le tecniche semplici e quelle avanzate sono:

- le tecniche semplici usano algoritmi più leggibili e di più facile comprensione

- le tecniche semplici hanno tutte una complessità polinomiale P che nel caso peggiore è $O(n^2)$

- Le tecniche avanzate hanno tutte una complessità polinomiale P inferiore a $O(n^2)$ che va dalla complessità $O(n^{1´2})$ del Heap Sort al O(n*log *n)* del Quick Sort nel caso peggiore

- Le tecniche avanzate utilizzano algoritmi più complessi da capire e da leggere e molte volte si appoggiano a procedure ricorsive.

Quando si scrive un programma, o parte di esso, si dovrebbe cercare di fare in modo che i dati vengano elaborati nel minore tempo possibile, questo per salvaguardare le risorse del calcolatore.
Se l'utente deve elaborare tramite un algoritmo un insieme di dati *n*, la complessità dell'algoritmo stesso è l'indice di paragone che ci permette di capire il numero di operazioni che il personal computer deve fare sull'insieme di dati *n* per risolvere il problema voluto.

Si utilizza la funzione O per indicare questo indice di paragone chiamato complessità.

I casi di complessità su un insieme di dati n che potrete avere in generale sono:

- $O(\log n)$ = logaritmica
- $O(n)$ = lineare
- $O(n \log n)$ = subquadratica
- $O(n^2)$ = quadratica
- $O(n^3)$ = cubica
- $O(n^a)$ = polinomiale
- $O(a^n)$ = esponenziale
- $O(n!)$ = fattoriale

Di solito quando si analizza la complessità di un algoritmo per l'elaborazione di un insieme di *n* dati, si dovrebbe distinguere l'analisi nei vari casi:

- Caso peggiore
- Caso medio
- Caso migliore

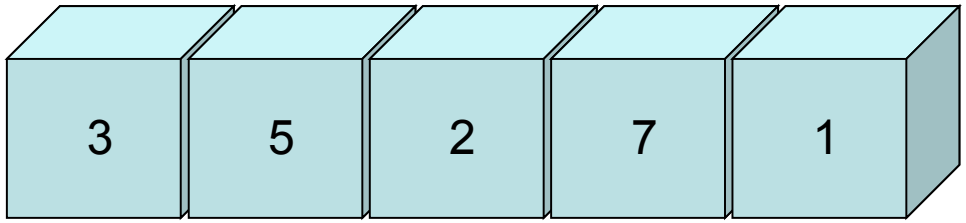
Col termine caso si intende in che modo i dati *n* sono disposti prima dell'elaborazione per la risoluzione del problema.

Le tecniche di ordinamento che andremo a studiare nel nostro corso saranno:

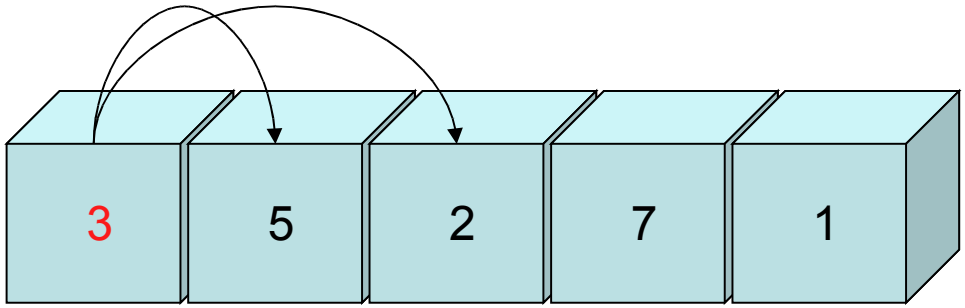
- Tecnica semplice di Sequential Sort
- Tecnica avanzata di Quick Sort

Ordinamento semplice (o ingenuo) Sequential Sort

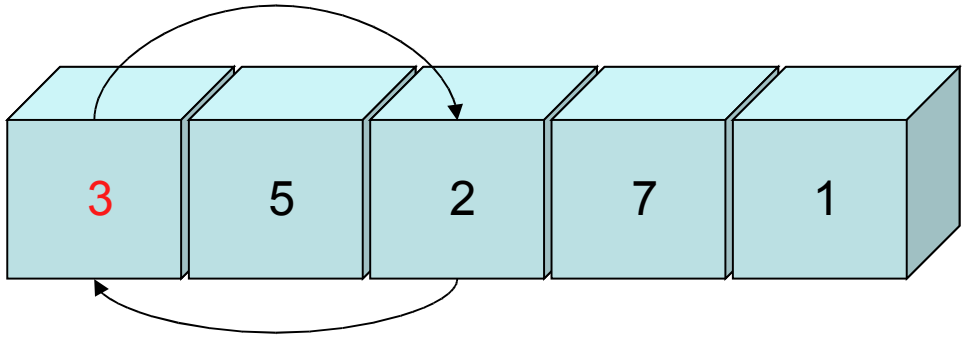
Prendiamo un vettore con 5 elementi inseriti



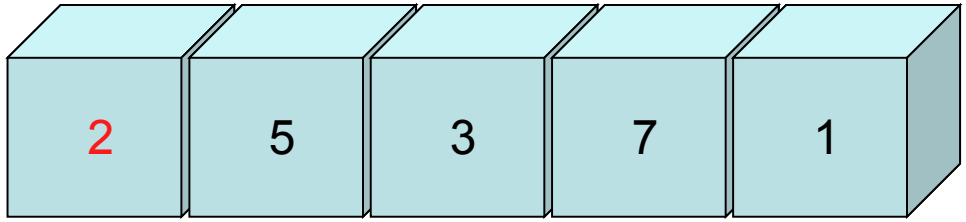
Fase 1: Partiamo dal primo elemento del vettore e confrontiamolo con tutti gli altri.
Se il primo elemento è più grande con quello confrontato, scambiamo i dati.



Ordinamento semplice (o ingenuo) Sequential Sort

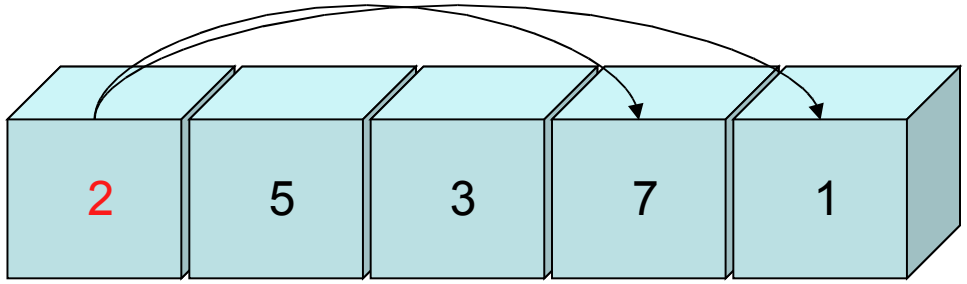


Notate che il numero 3 è più grande del 2, quindi dovete procedere allo scambio dei dati.

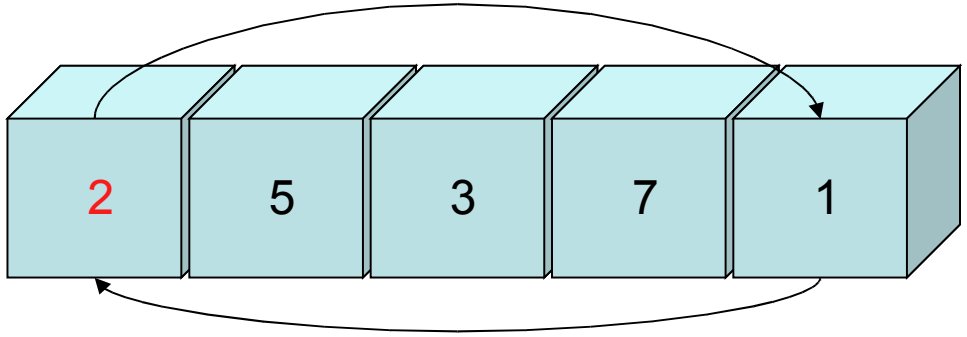


Ora il primo numero del vettore non è più il valore 3 ma il valore 2 che andrà confrontato con i restanti dati del vettore, ossia il numero 7 ed il numero 1.

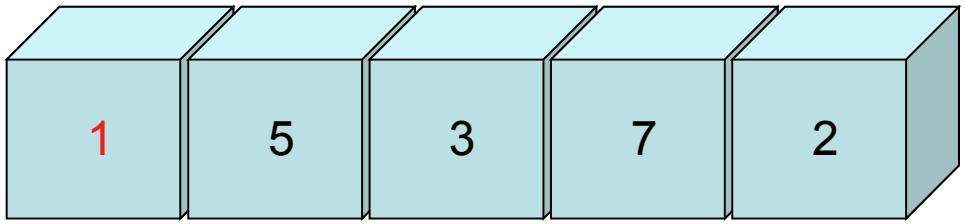
Ordinamento semplice (o ingenuo) Sequential Sort



Il numero 2 è più grande di 1 quindi si procede allo scambio.



Ordinamento semplice (o ingenuo) Sequential Sort



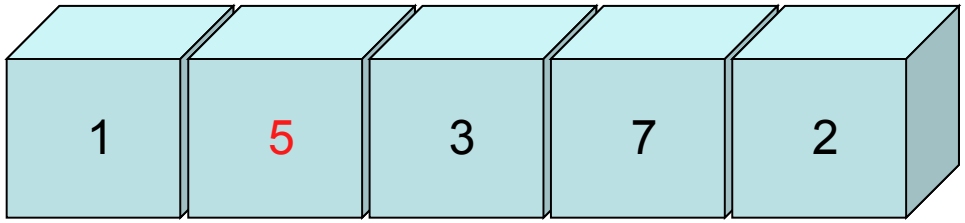
Quanti confronti abbiamo fatto in questa prima fase?

Elenco confronti:

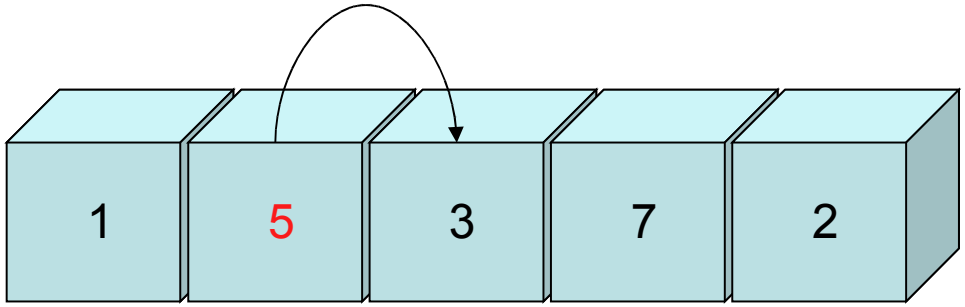
a. 3 con 5
b. 3 con 2
c. 2 con 7
d. 2 con 1

In totale 4 confronti, quindi essendo 5 gli elementi del vettore possiamo in generale dire che nella prima fase, se ho *n* elementi, i confronti sono *n*-1.

Ordinamento semplice (o ingenuo) Sequential Sort

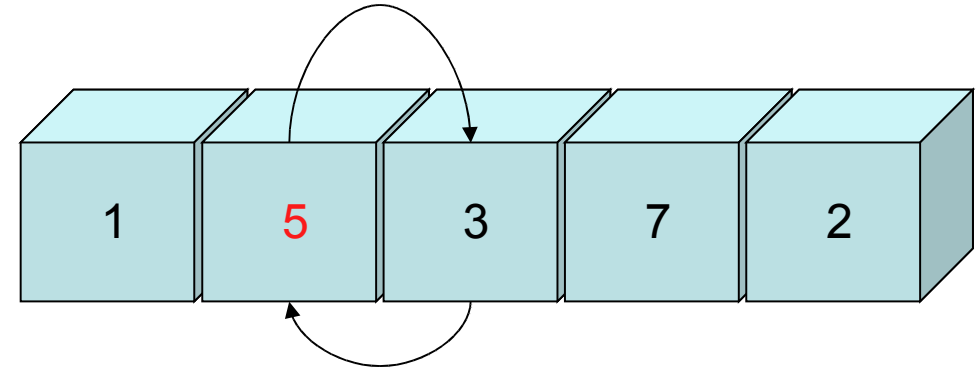


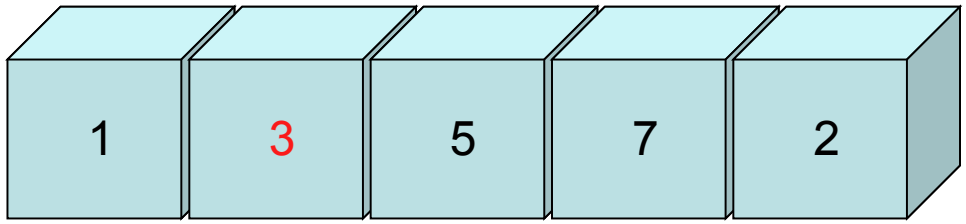
Fase 2: Ora il primo numero del vettore che verrà confrontato sarà proprio il secondo elemento.



Ordinamento semplice (o ingenuo) Sequential Sort

Procediamo con lo scambio tra il valore 5 ed il valore 3.

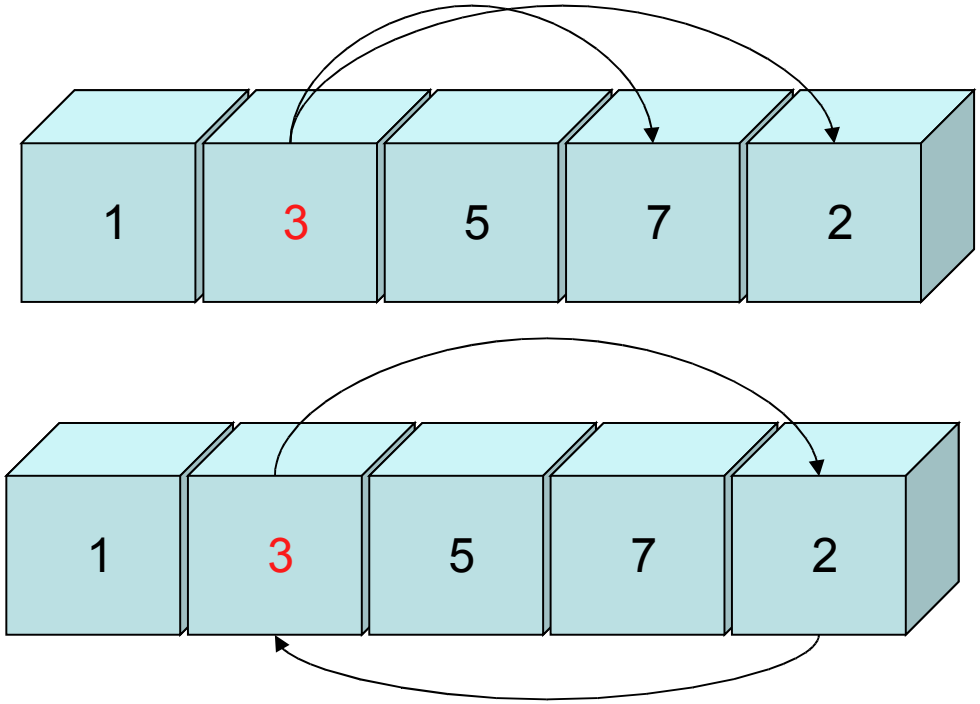




Il numero elemento da confrontare non è più il valore 5 ma il valore 3.

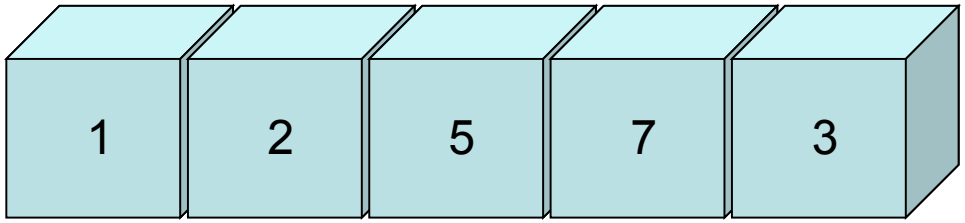
Ordinamento semplice (o ingenuo) Sequential Sort

Continuiamo il confronto sui restanti valori ossia 7 e 2.



Effettuiamo lo scambio tra il valore 2 ed il valore 3.

Ordinamento semplice (o ingenuo) Sequential Sort



Quanti confronti abbiamo fatto in questa seconda fase?

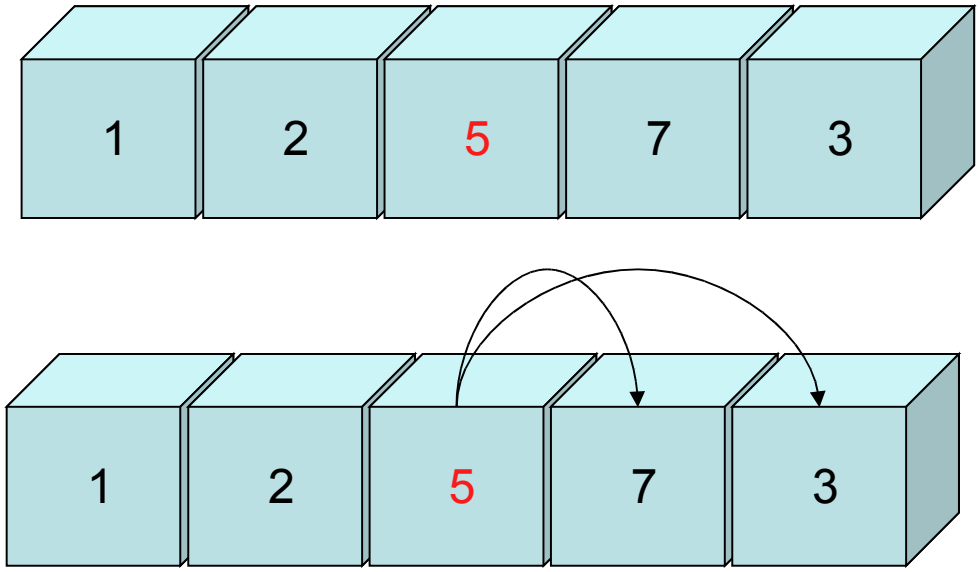
Elenco confronti:

a. 3 con 5
b. 3 con 7
c. 3 con 2

In totale 3 confronti, quindi essendo 5 gli elementi del vettore possiamo in generale dire che nella seconda fase, se ho *n* elementi, i confronti sono *n*-2.

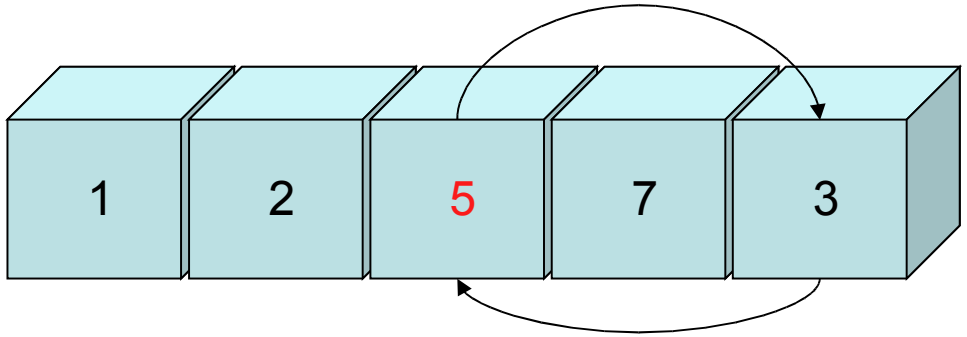
Ordinamento semplice (o ingenuo) Sequential Sort

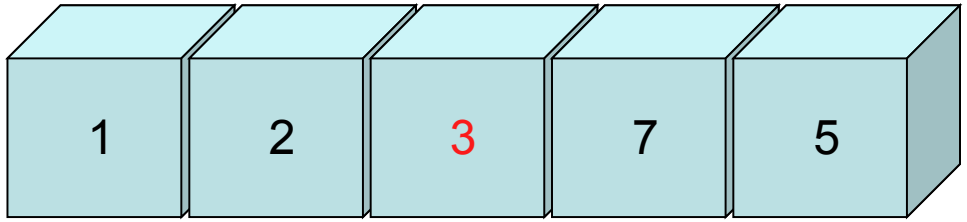
Ora il primo elemento da confrontare è il terzo elemento del vettore ossia il dato 5.



Il valore 5 è più grande del valore 3 quindi procediamo allo scambio.

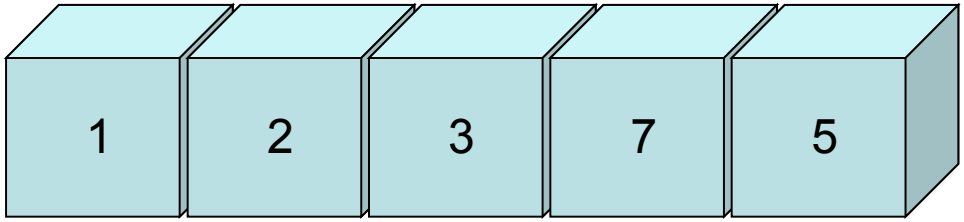
Ordinamento semplice (o ingenuo) Sequential Sort





Ovviamente non ci sono più elementi da confrontare in questa fase.

Ordinamento semplice (o ingenuo) Sequential Sort



Quanti confronti abbiamo fatto in questa terza fase?

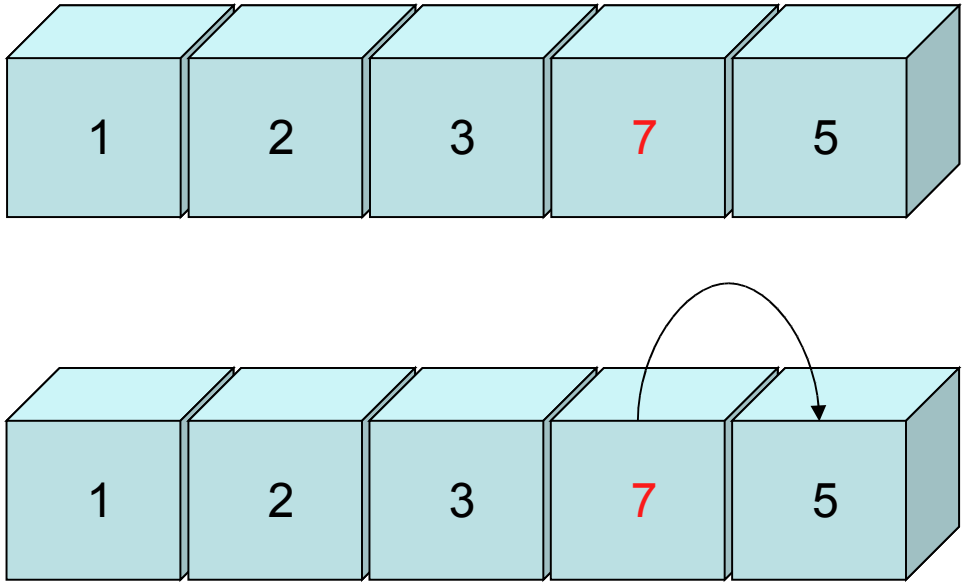
Elenco confronti:

a. 5 con 7
b. 5 con 3

In totale 2 confronti, quindi essendo 5 gli elementi del vettore possiamo in generale dire che nella terza fase, se ho *n* elementi, i confronti sono *n*-3.

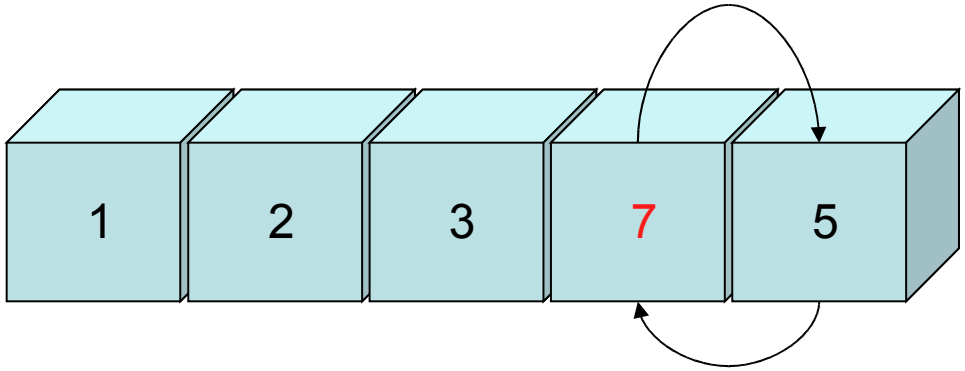
Ordinamento semplice (o ingenuo) Sequential Sort

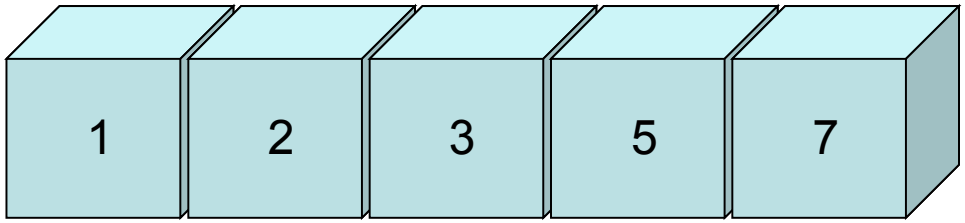
Il primo elemento da confrontare è il quarto elemento del vettore ossia il dato 7.



Il 7 è più grande del 5 quindi si deve procedere allo scambio.

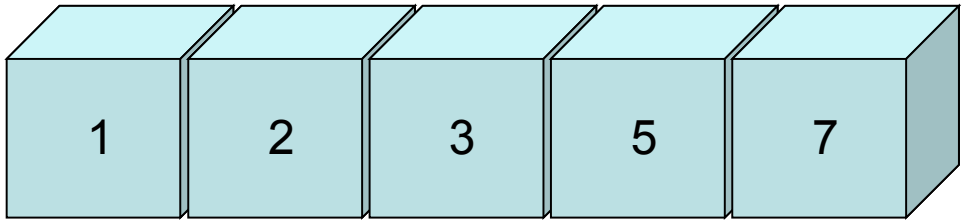
Ordinamento semplice (o ingenuo) Sequential Sort





Il vettore risulta dopo l'ultimo confronto ordinato.

Ordinamento semplice (o ingenuo) Sequential Sort



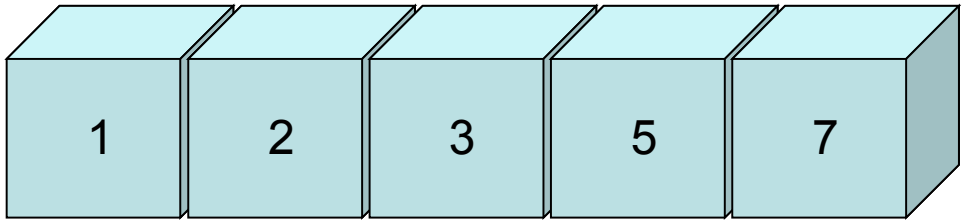
Quanti confronti abbiamo fatto in questa quarta fase?

Elenco confronti:

a. 7 con 5

Un solo confronto, quindi essendo 5 gli elementi del vettore possiamo in generale dire che nella quarta fase, se ho *n* elementi, i confronti sono *n*-4.

Ordinamento semplice (o ingenuo) Sequential Sort



La complessità finale O per ordinare un vettore di *n* elementi con l'algoritmo di Sequential Sort, sulla base dei confronti fatti, risulta quindi essere:

$$(n\text{-}1) + (n\text{-}2) + (n\text{-}3) + \ldots\ldots + 2 + 1 = \frac{n\,(n\text{-}1)}{2} = \frac{n^2\text{-}n}{2}$$

Se notate il numeratore, la potenza dominante è n^2, quindi la complessità finale dell'algoritmo risulta essere:

$$O(n^2)$$