Modulo: Algoritmi di ordinamento di tipo “Divide et Impera” [P2_12]

Unità didattica: Richiami sugli algoritmi “in place” a complessità quadratica [1-T]

Titolo: Classificazione e valutazione degli algoritmi di ordinamento

Argomenti trattati:

- ✓ Classificazione degli algoritmi di ordinamento
- ✓ Parametri di valutazione, caso medio e casi limite
- ✓ Alcuni algoritmi a complessità quadratica

Prerequisiti richiesti: array, complessità computazionale, ordinamento

I problemi di **ricerca** e di **ordinamento** di dati (rispetto ad un campo dell'informazione detto **chiave**) sono molto importanti nelle applicazioni.

Esistono vari metodi per risolvere lo stesso problema: questi si confrontano rispetto al ***parametro di efficienza*** (di spazio e di tempo) relativamente anche al particolare problema ed alla organizzazione dei dati.

Nella valutazione dell'algoritmo, si considerano anche i **casi "limite"** di dati già ordinati oppure totalmente disordinati.

Metodi di ordinamento <u>interni</u>: dati in memoria centrale.

Metodi di ordinamento <u>esterni</u>: dati su memoria di massa parzialmente in memoria centrale.

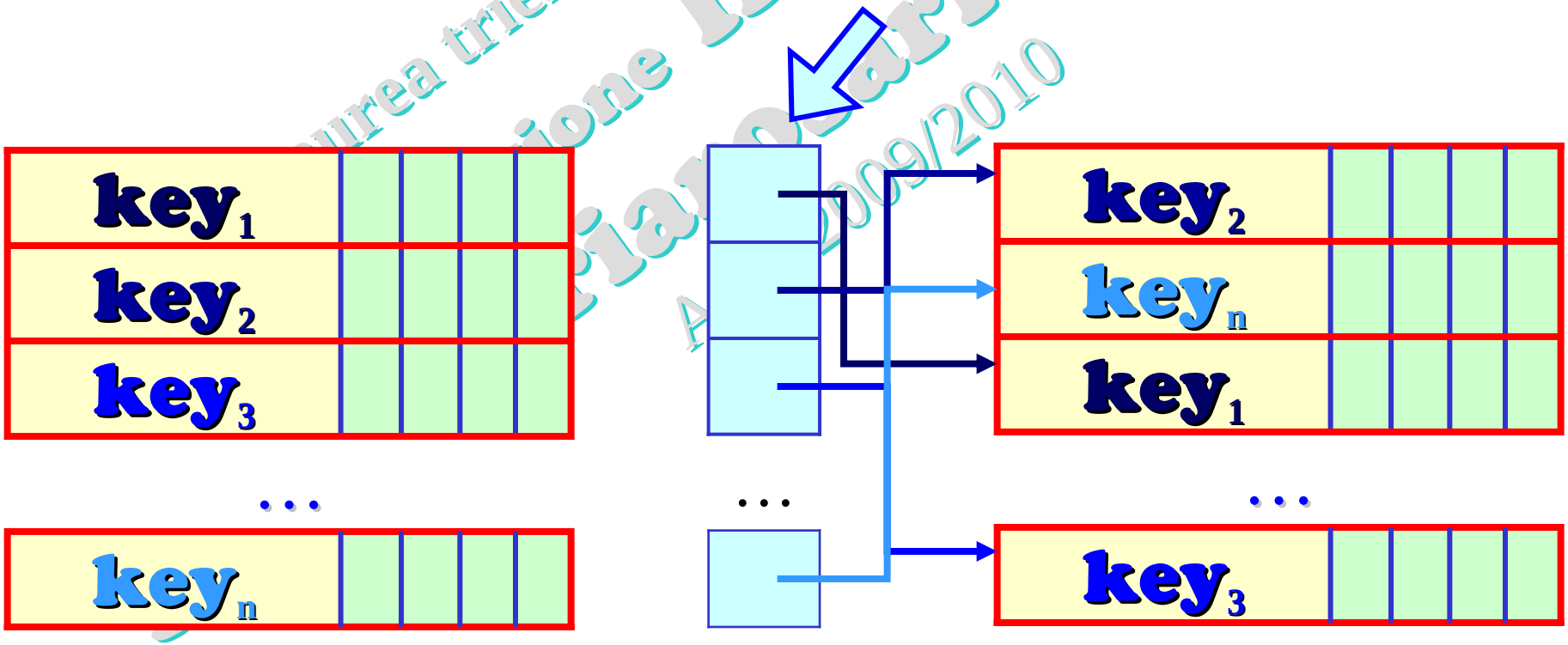
strutture di base usate in algoritmi di ordinamento

Array

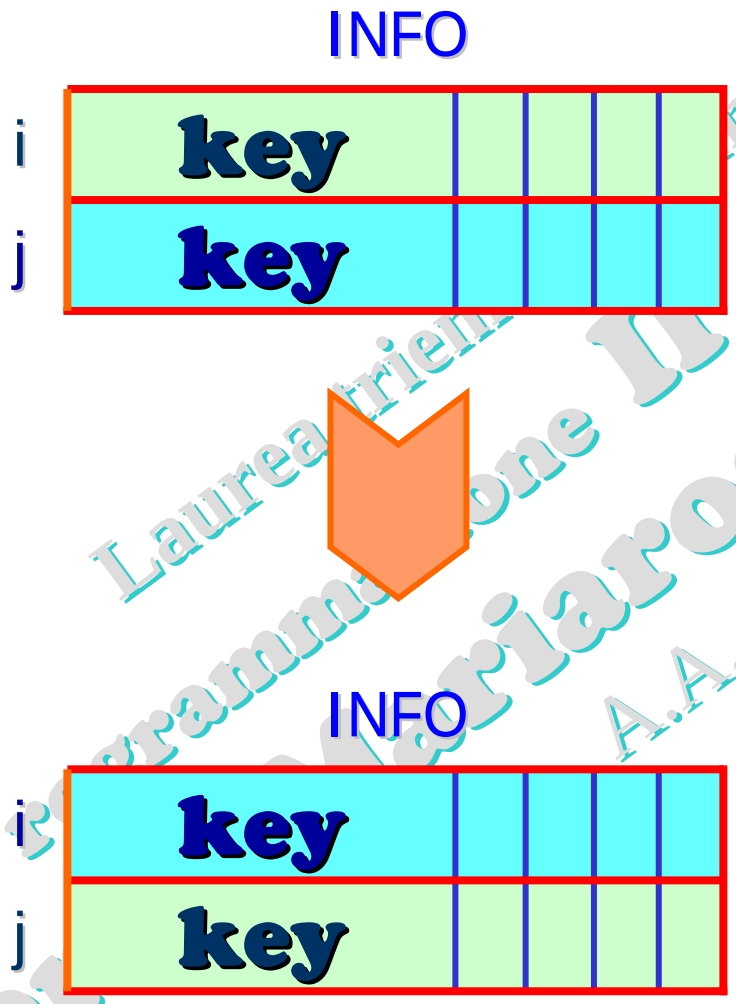
Linked list

Alcuni algoritmi di ordinamento (su array) effettuano **scambi** di informazioni:

- **Scambi reali**: i dati sono "fisicamente" scambiati di posto;
- **Scambi virtuali**: i dati non sono fisicamente scambiati perché si opera tramite **puntatori**.



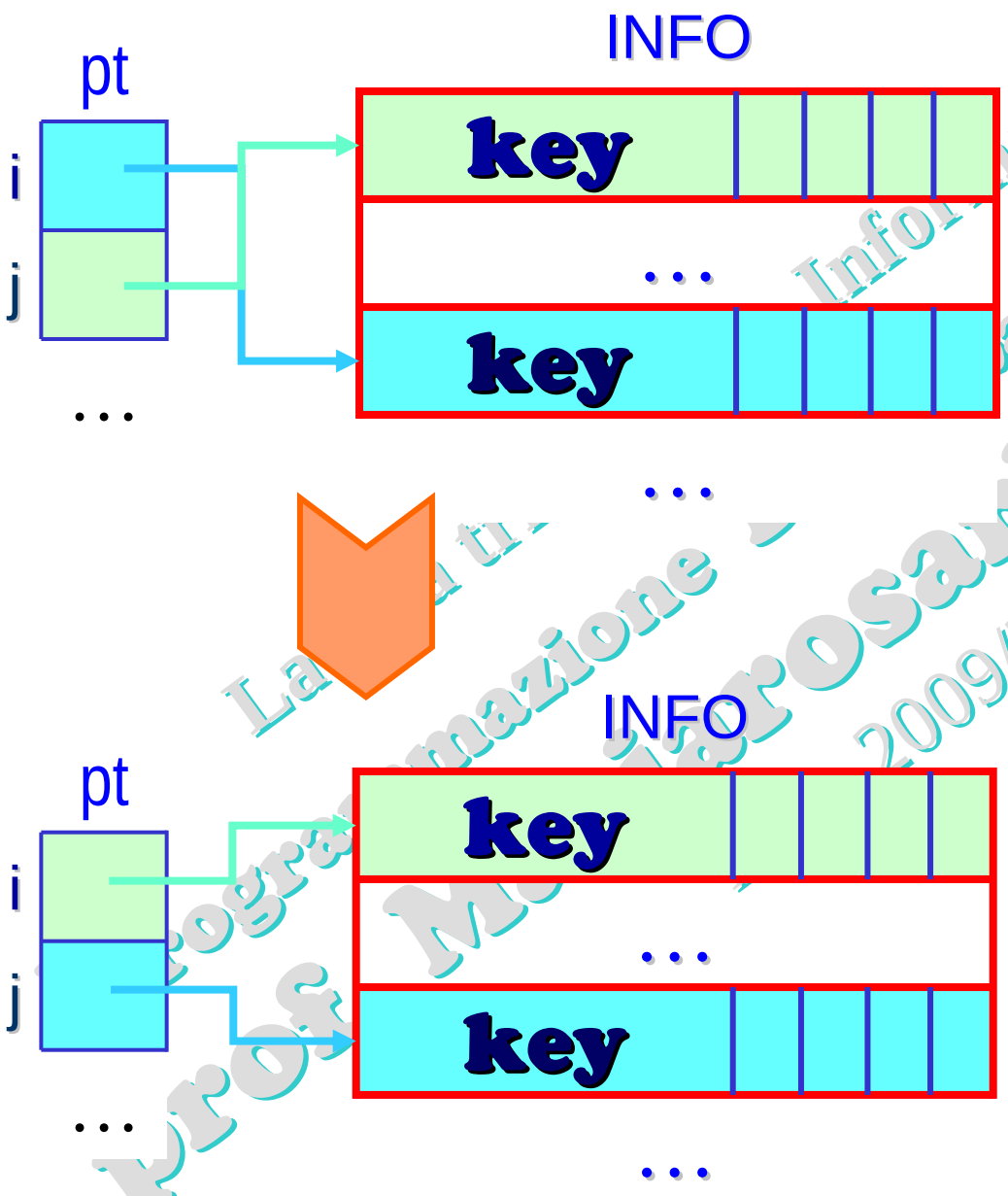
Scambi reali: esempio




```
if key_i > key_j
        Temp := INFO_i
        INFO_i := INFO_j
        INFO_j := Temp
end
```

Le informazioni sono fisicamente scambiate

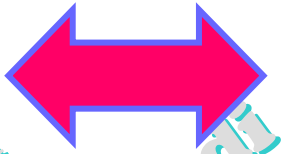
Scambi virtuali: esempio




```
if   pti→key > ptj→key
       Temp := pti
       pti := ptj
       ptj := Temp
end
```

Le informazioni non sono fisicamente scambiate.
Gli scambi coinvolgono solo i puntatori.

Algoritmo di ordinamento stabile



Quando viene mantenuto l'ordinamento relativo delle informazioni con chiavi uguali.

Esempio

1	M	5
2	A	3
3	R	2
4	S	2
5	R	4
6	O	1
7	I	6
8	Z	4

ordinamento

Algoritmo stabile

1	A	3
2	I	6
3	M	5
4	O	1
5	R	2
6	R	4
7	S	2
8	Z	4

Algoritmo instabile

1	A	3
2	I	6
3	M	5
4	O	1
5	R	4
6	R	2
7	S	2
8	Z	4

Esempio: ordinare secondo la chiave "state"

city	state
Chicago	IL
Champaign	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI
Albany	NY
Green Bay	WI
Syracuse	NY
Rockford	IL
Evanston	IL

Ordinamento per minimo

Chicago	IL
Champaign	IL
Rockford	IL
Evanston	IL
Detroit	MI
Albany	NY
Syracuse	NY
Buffalo	NY
New York	NY
Green Bay	WI
Milwaukee	WI

Non è stabile! (dipende da com'è implementato)

SelectionSort
Ordinam. per selezione (minimo)

Idea: cerca il minimo del vettore disordinato e lo inserisce al suo posto.

Legenda: riquadro rosa = posto finale del minimo; riquadro rosso = valore minimo

1	2	3	4	5	6	7	8	9	10	11	12	Passo	
M	A	R	S	O	I	Z	D	L	U	N	E	1	scambio
A	M	R	S	O	I	Z	D	L	U	N	E	2	scambio
A	D	R	S	O	I	Z	M	L	U	N	E	3	scambio
A	D	E	S	O	I	Z	M	L	U	N	R	4	scambio
A	D	E	I	O	S	Z	M	L	U	N	R	5	scambio
...												...	
A	D	E	I	L	M	N	O	R	S	U	Z	n - 1	scambio

SelectionSort
Ordinam. per selezione (massimo)

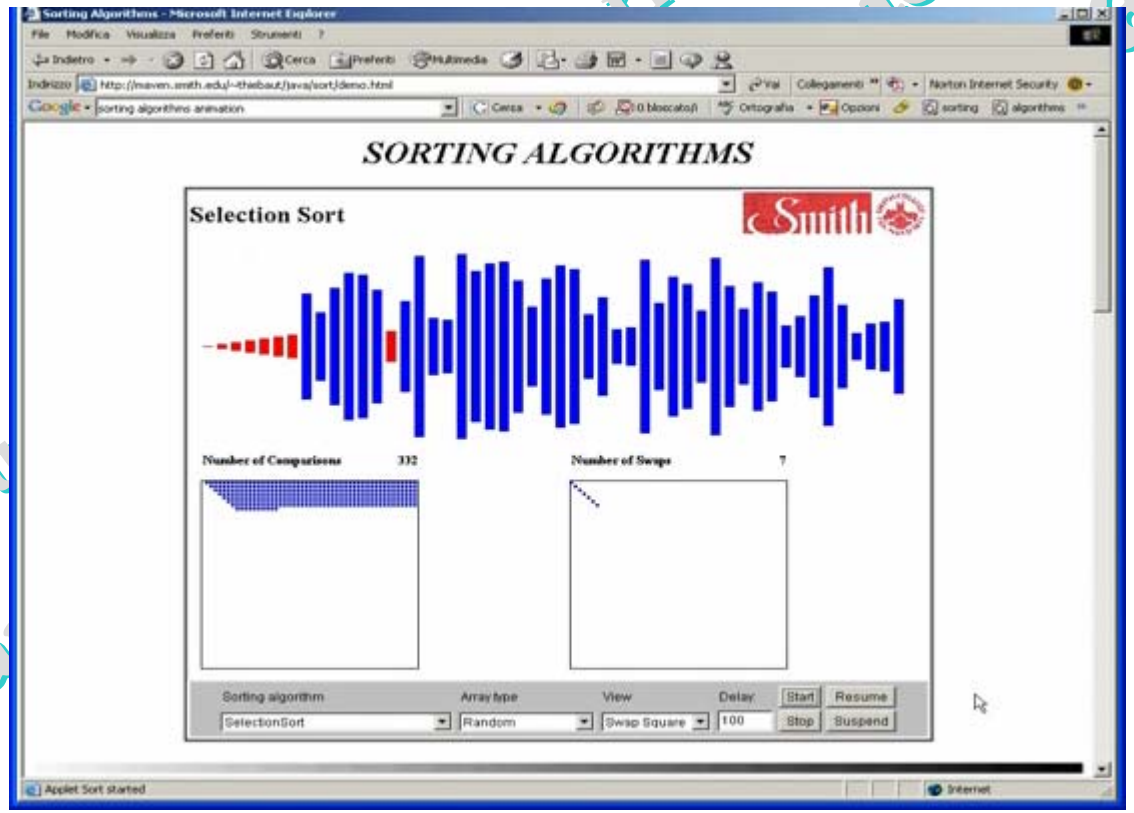
Idea: cerca il massimo del vettore disordinato e lo inserisce al suo posto.

valore massimo

1	2	3	4	5	6	7	8	9	10	11	12	Passo
M	A	R	S	O	I	Z	D	L	U	N	E	1
M	A	R	S	O	I	E	D	L	U	N	Z	2
M	A	R	S	O	I	E	D	L	N	U	Z	3
M	A	R	N	O	I	E	D	L	S	U	Z	4
M	A	L	N	O	I	E	D	R	S	U	Z	5
...												...
A	D	E	I	L	M	N	O	R	S	U	Z	*n* - 1

posto del massimo

Selection Sort java applet



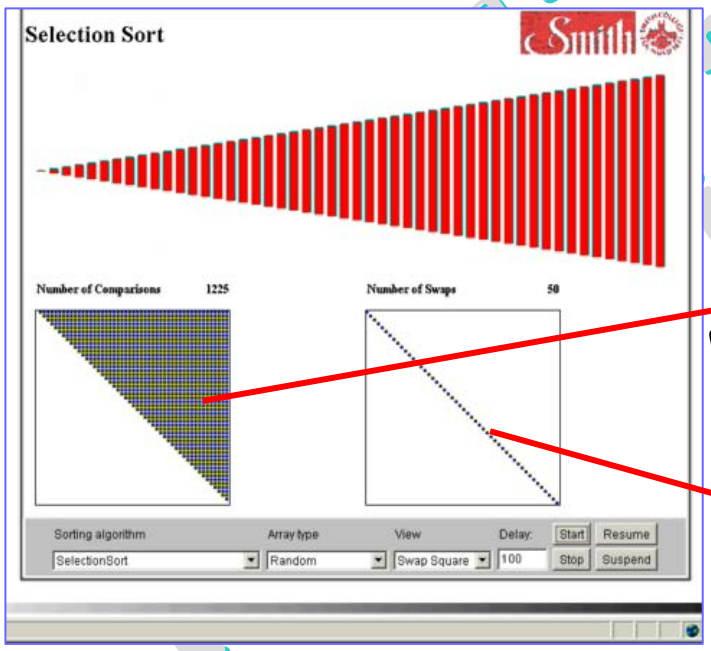
URL: http://maven.smith.edu/~thiebaut/java/sort/demo.html

SelectionSort

Complessità di spazio = $O(n)$

in place

Complessità di tempo



Numero confronti

matrice triangolare

= $O(\frac{1}{2}n^2)$

Numero scambi

matrice diagonale

= al più $O(n)$

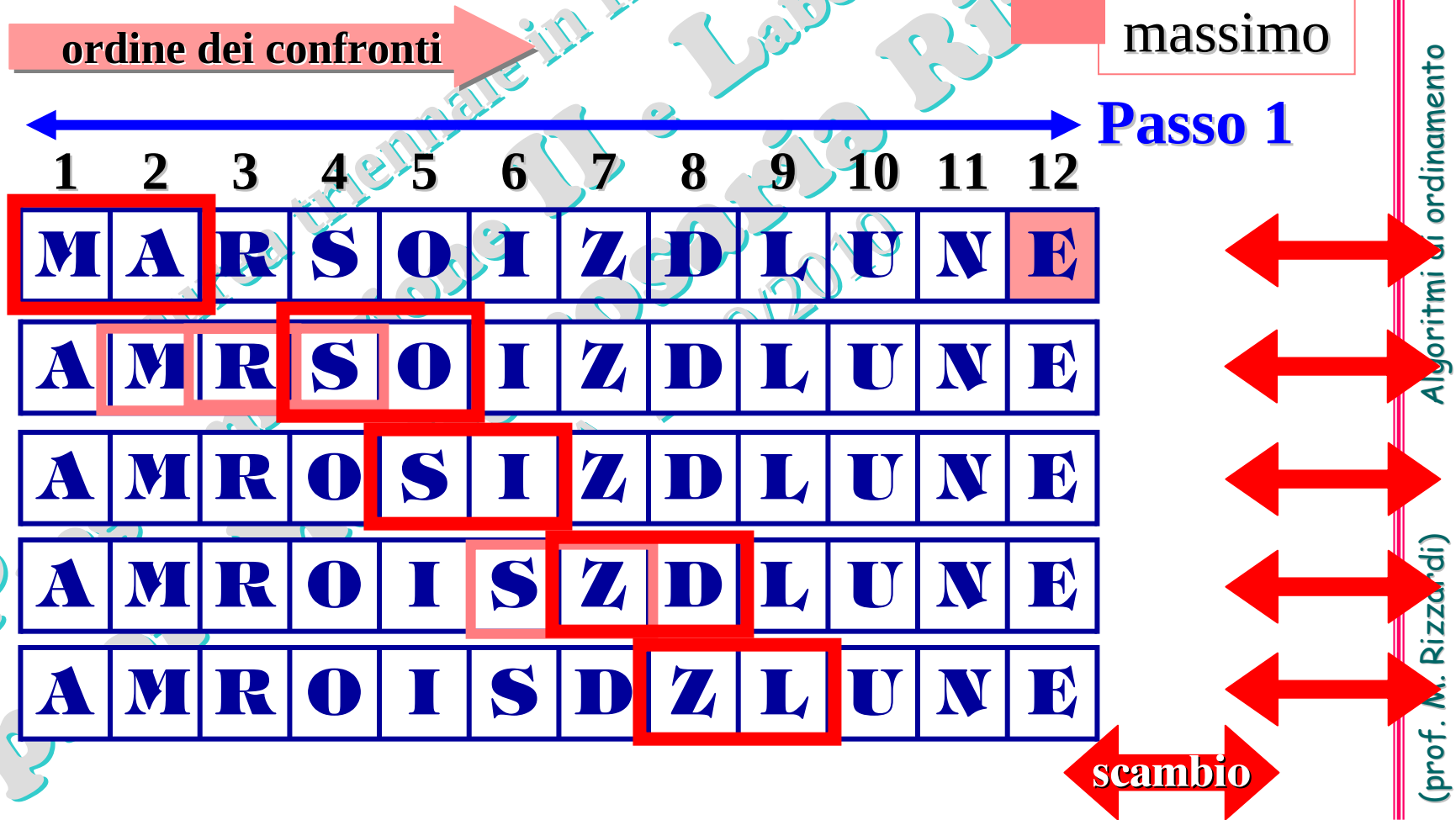
ExchangeSort

Ordinam. per scambi

detto anche

BubbleSort

Idea: confronta due componenti adiacenti del vettore disordinato e le scambia se non sono ordinate.



Exchange sort (cont.)

1	2	3	4	5	6	7	8	9	10	11	12
A	M	R	O	I	S	D	L	Z	U	N	E
A	M	R	O	I	S	D	L	U	Z	N	E
A	M	R	O	I	S	D	L	U	N	Z	E
A	M	R	O	I	S	D	L	U	N	E	Z

Passo 1

massimo a posto

ricomincia a confrontare dalla prima componente

1	2	3	4	5	6	7	8	9	10	11	12
A	M	R	O	I	S	D	L	U	N	E	Z
A	M	O	R	I	S	D	L	U	N	E	Z
A	M	O	I	R	S	D	L	U	N	E	Z
A	M	O	I	R	D	S	L	U	N	E	Z
A	M	O	I	R	D	L	S	N	E	U	Z

Passo 2

...

max. relat. a posto

ricomincia a confrontare dalla prima componente

Passo 3

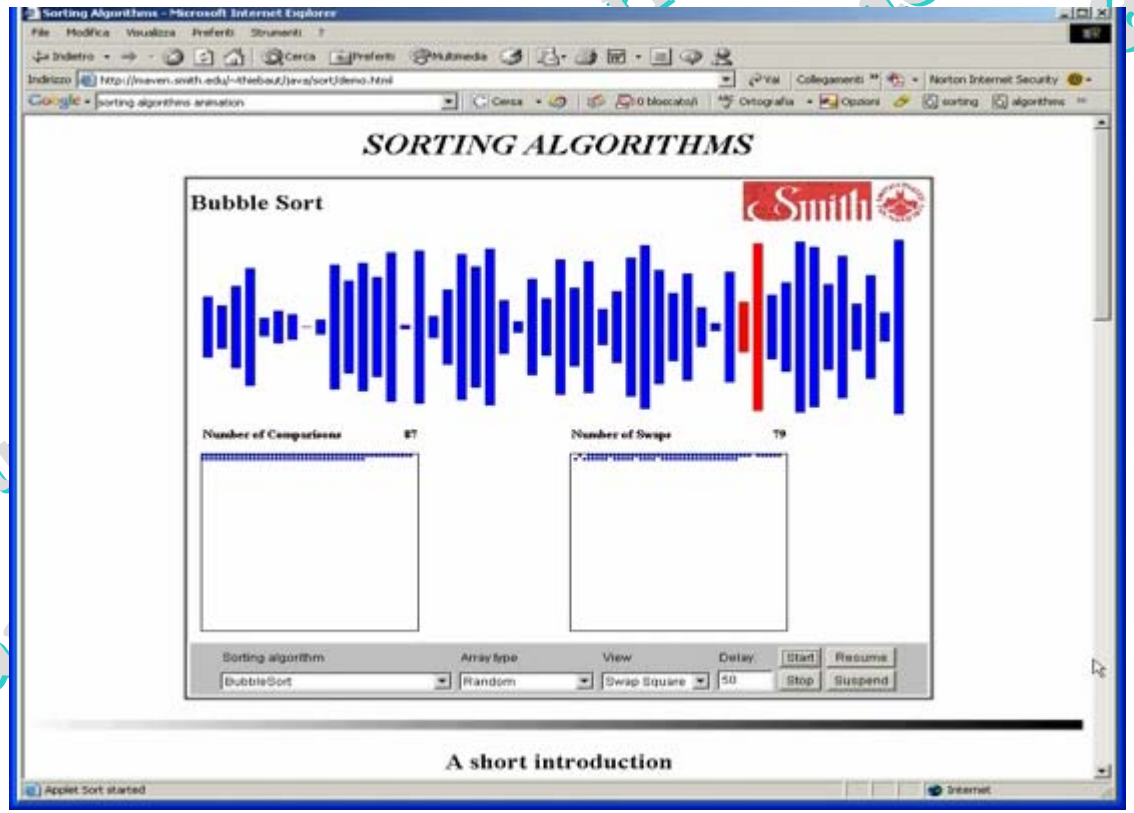
1	2	3	4	5	6	7	8	9	10	11	12	
A	M	O	I	R	D	L	S	N	E	U	Z	
A	M	O	I	D	R	S	L	N	E	U	Z	
A	M	O	I	D	R	L	S	N	E	U	Z	
A	M	O	I	D	R	L	N	S	E	U	Z	
A	M	O	I	D	R	L	N	E	S	U	Z	max. relat. a posto

...

...

A	D	E	I	L	M	N	O	R	S	U	Z

Bubble Sort java applet



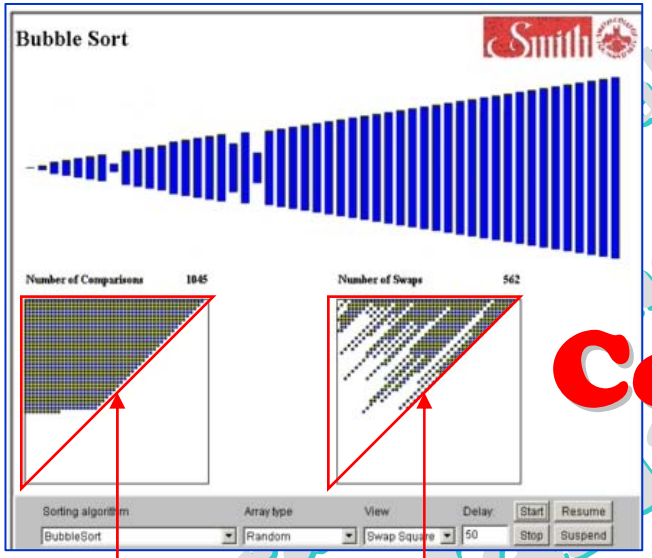
Esiste anche una versione bidirezionale (bidirectional bubble)

URL: http://maven.smith.edu/~thiebaut/java/sort/demo.html

ExchangeSort (BubbleSort)

Complessità di spazio = $O(n)$

in place



matrice triangolare

Complessità di tempo

	(caso migliore)	(caso peggiore)
Numero confronti	$= O(n)$	$O(\frac{1}{2}n^2)$
Numero scambi	$= -$	$O(\frac{1}{2}n^2)$

InsertionSort

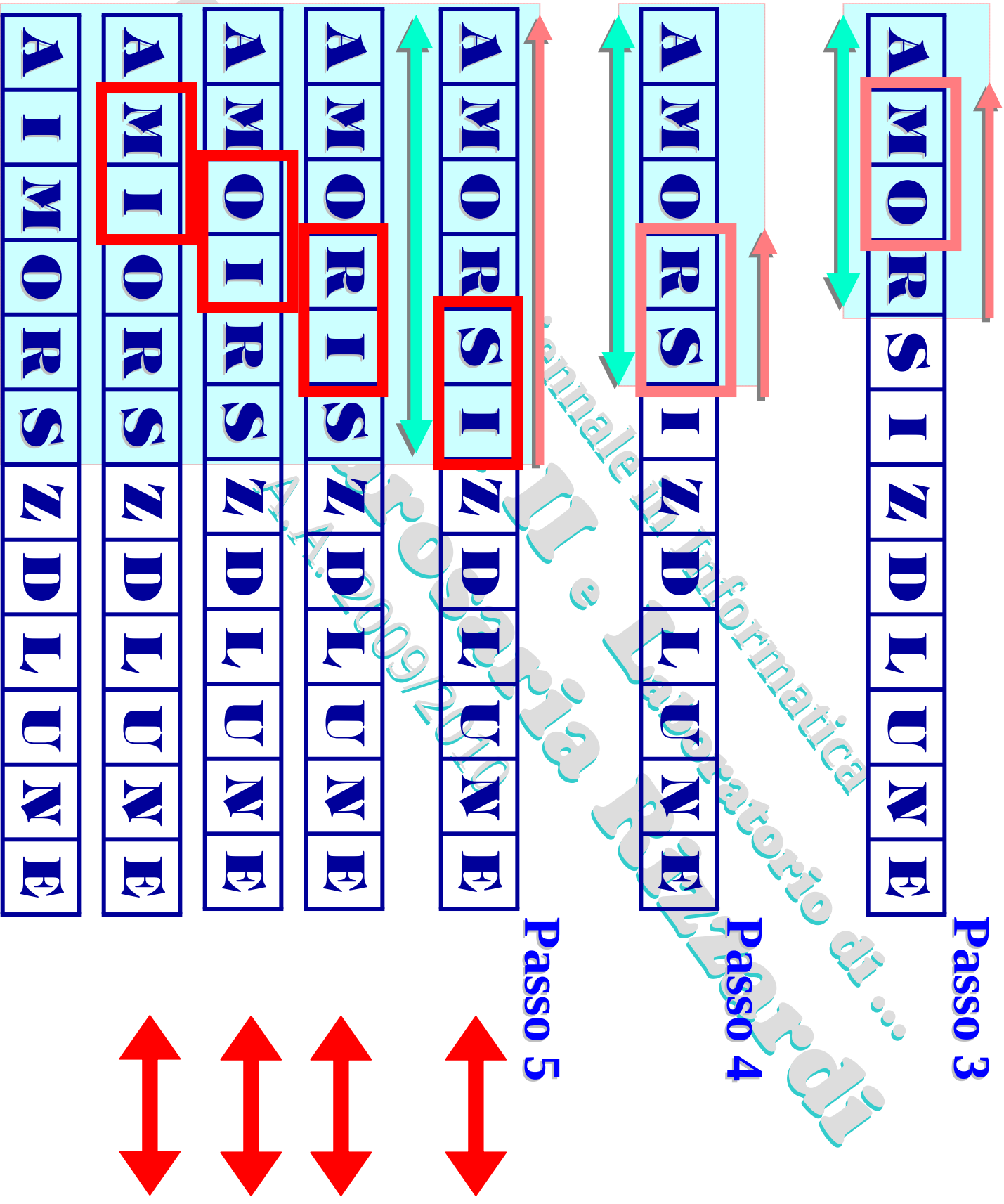
ordine dei confronti

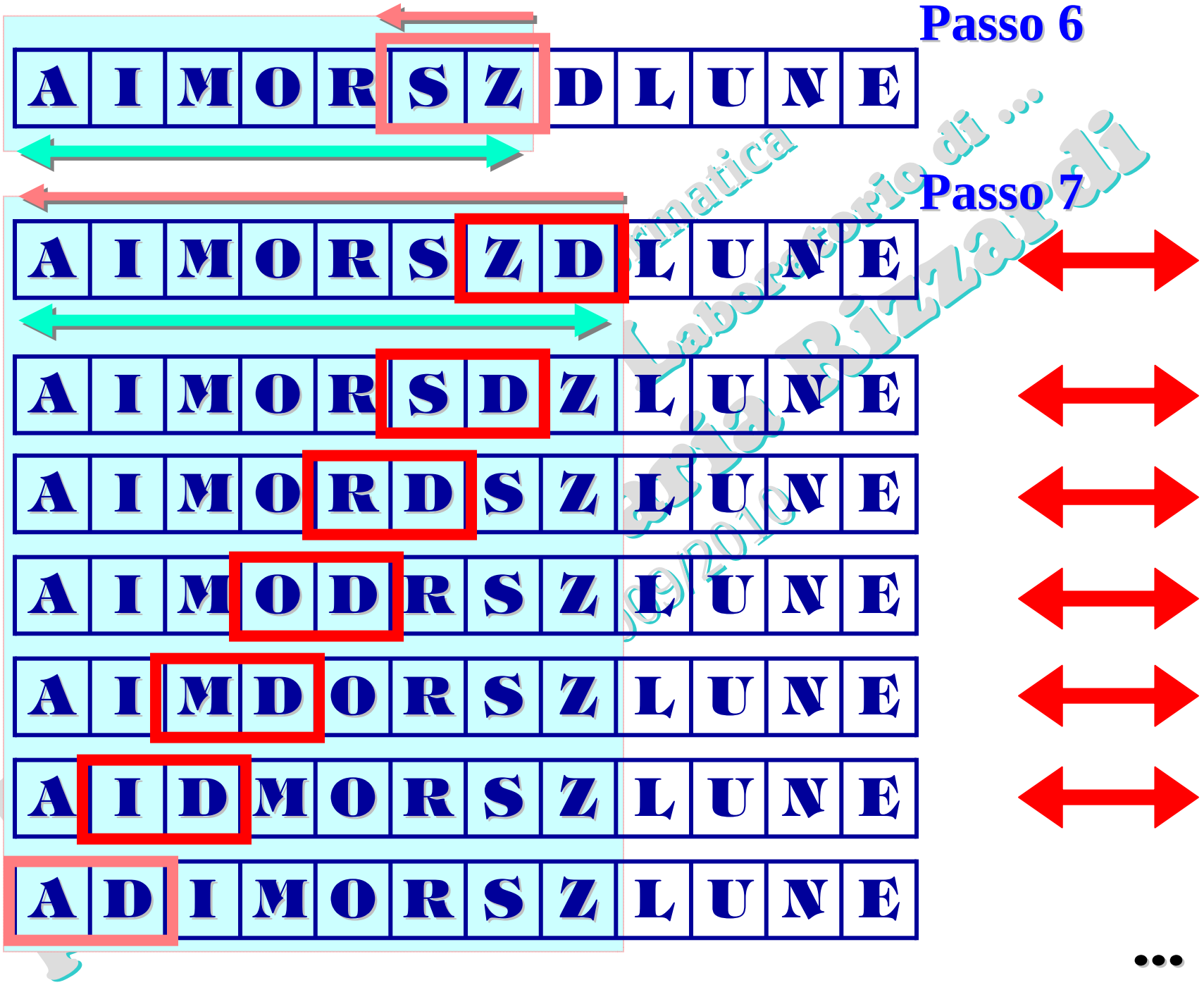
sottovettore da ordinare

Idea: ordina porzioni del vettore disordinato tramite una modifica del metodo *ExchangeSort* per migliorarne l'efficienza nel caso di vettore poco disordinato.

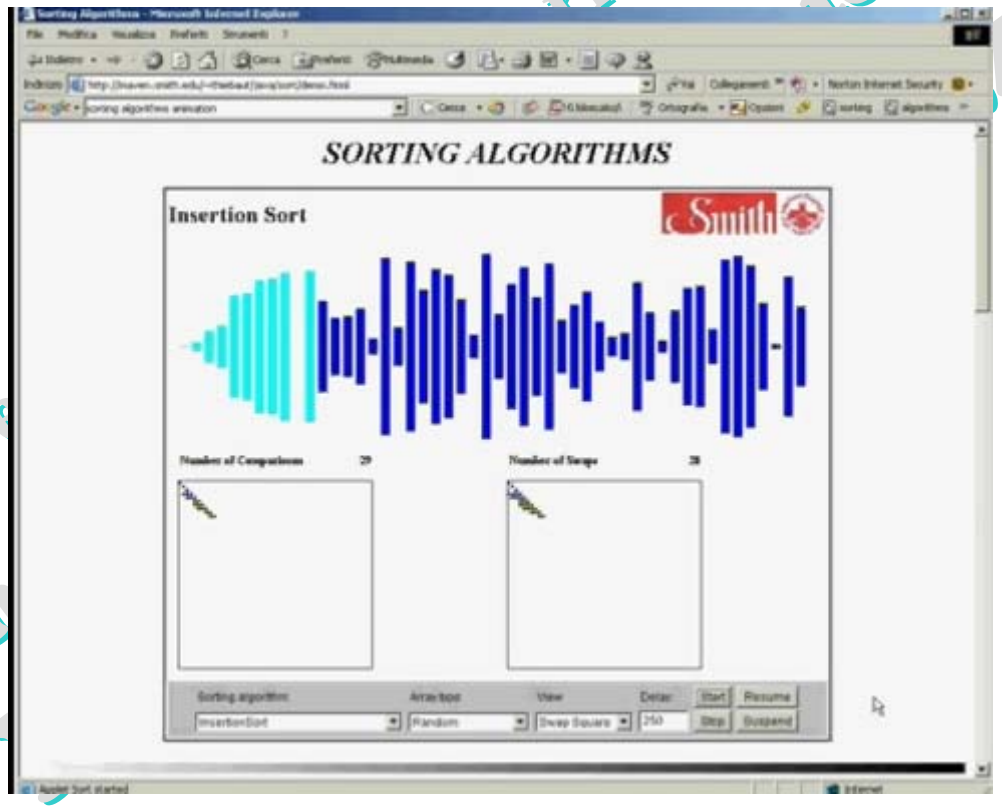
se non c'è scambio si va al passo successivo

InsertionSort (cont.)





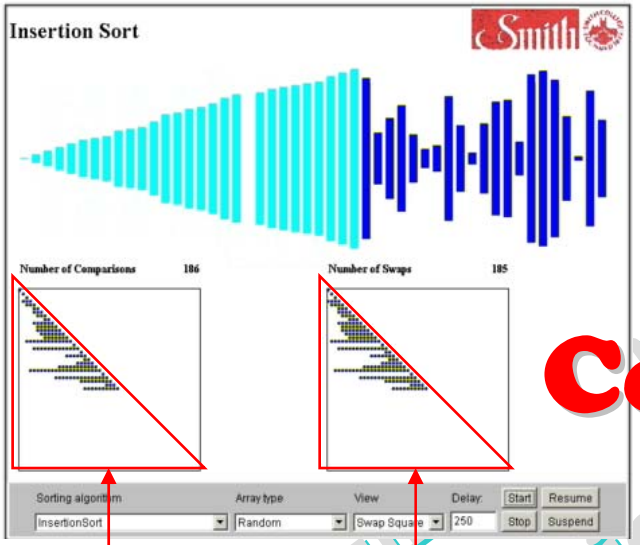
Insertion Sort java applet



URL: http://maven.smith.edu/~thiebaut/java/sort/demo.html

InsertionSort

Complessità di spazio = $O(n)$ in place



Complessità di tempo

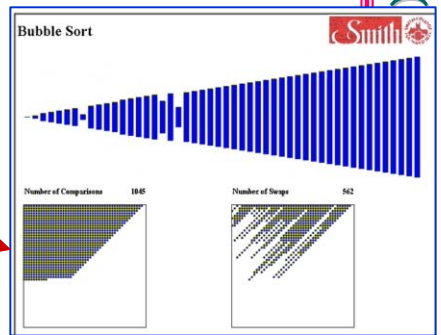
	(caso migliore)	(caso peggiore)
Numero confronti =	$O(n)$	$O(\frac{1}{2}n^2)$
Numero scambi =	-	$O(\frac{1}{2}n^2)$

L'Insertion Sort mediamente risulta

2 volte più veloce del Bubble

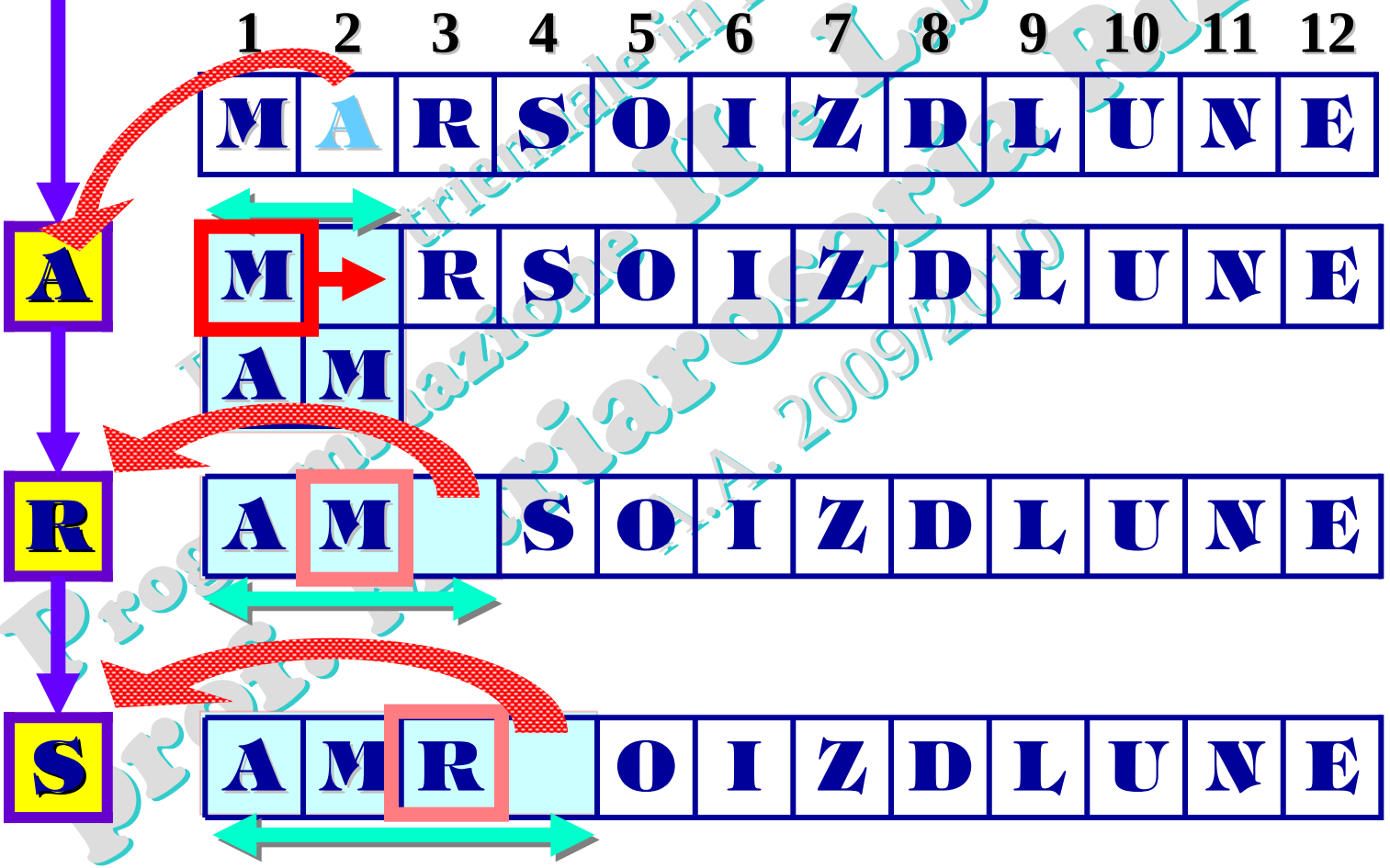
e

40% più veloce del Selection



InsertionSort

Per migliorare l'efficienza si possono eliminare gli scambi introducendo un nodo temporaneo



Esercizi: implementare in *C*…

1 l'algoritmo *SelectionSort* su un array di struttura
- in versione iterativa e ricorsiva;
- mediante scambi reali e scambi virtuali.

2 l'algoritmo *SelectionSort* su una lista bidirezionale. [liv.3]

3 l'algoritmo *ExchangeSort (BubbleSort)* su un array di struttura in versione iterativa mediante scambi reali e scambi virtuali.

4 l'algoritmo *InsertionSort* su un array di struttura.