Ricerca

Credits A. Montenegro

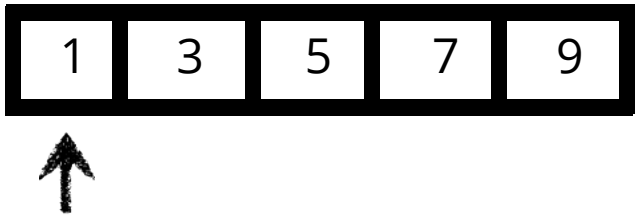
Ricerca

- Scorrimento iterativo dell’array – Caso in cui non so niente dell’array

N = 7 ?

Ricerca

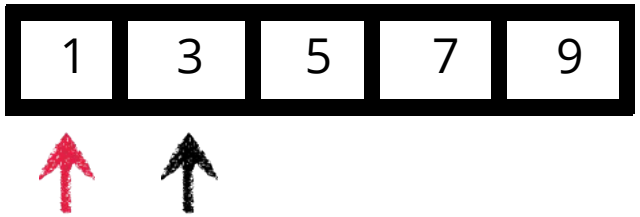
N = 7 ?



Ricerca

Scorrimento iterativo dell’array

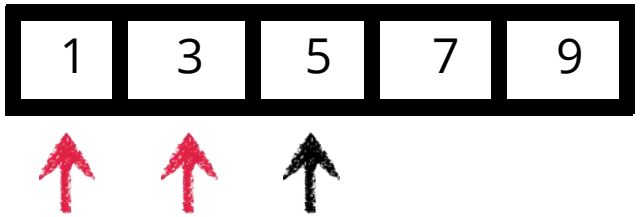
N = 7 ?



Ricerca

Scorrimento iterativo dell’array

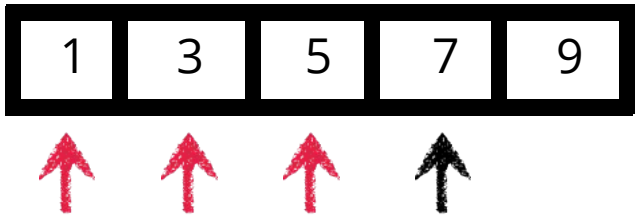
N = 7 ?



Ricerca

Scorrimento iterativo dell’array

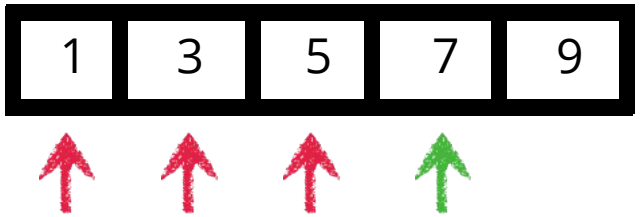
N = 7 ?



Ricerca

Scorrimento iterativo dell'array

N = 7 ?



- **Caso peggiore**: scorriamo **tutto l'array** (come quando non è presente l'argomento, complessità lineare legata al numero degli elementi)
- Funziona anche se l'array **non è ordinato**

Ricerca – E se l’array fosse ordinato?

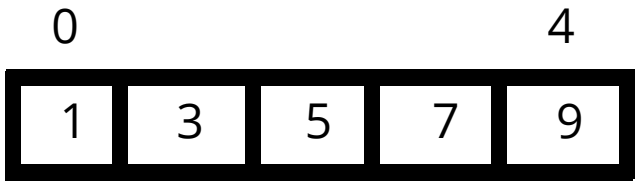
- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?



Start = 0 End = 4

m = 2

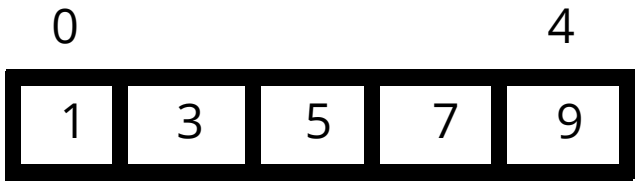
Controllo l'indice nel punto medio

$$m = \frac{start + end}{2}$$

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?



Start = 0 End = 4

m = 2

Controllo l'indice nel punto medio

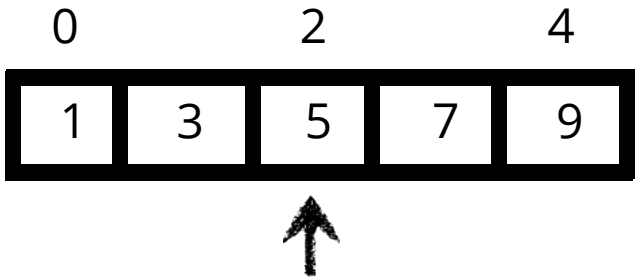
$$m = \frac{start + end}{2}$$

Complessità logaritmica, perché vado a dividere l'array in parti

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?
Start = 0
End = 4
m = 2



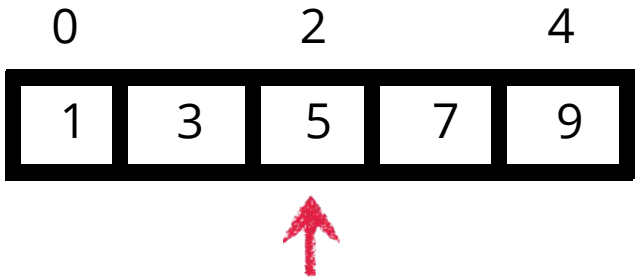
Controllo l'indice nel punto medio

$$m = \frac{start + end}{2}$$

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?
Start = 0
End = 4
m = 2



Controllo l'indice nel punto medio

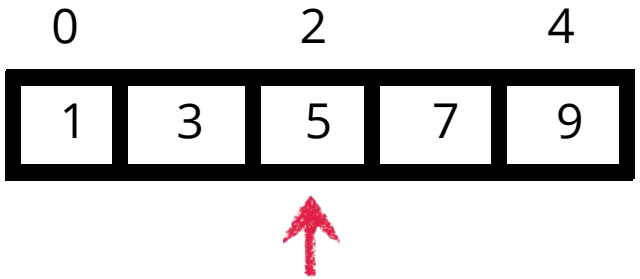
$$m = \frac{start + end}{2}$$

L'elemento in posizione m non è quello che cerchiamo
L'elemento in posizione m è più piccolo di N e l'array è ordinato

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?
Start = 0
End = 4
m = 2



Controllo l'indice nel punto medio

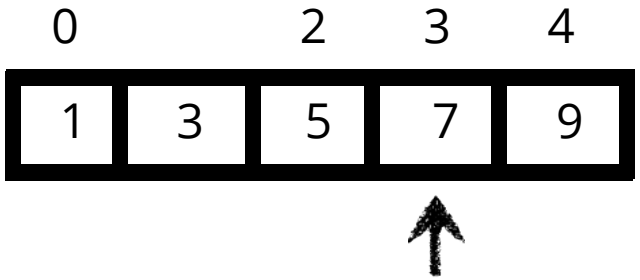
$$m = \frac{start + end}{2}$$

Cerchiamo nella metà di destra dove ci sono gli elementi più grandi di quello in posizione m (quindi forse c'è N)
Start = m+1, End = End (nel caso opposto Start = Start e End = m-1)

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?
Start = 3
End = 4
m = 3



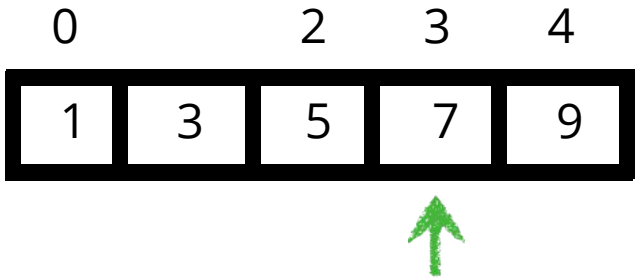
Controllo l'indice nel punto medio

$$m = \frac{start + end}{2}$$

Ricerca – E se l'array fosse ordinato?

- Ricerca Binaria su Array Ordinato – Più efficiente, bisezione!

N = 7 ?
Start = 3
End = 4
m = 3



Controllo l'indice nel punto medio

$$m = \frac{start + end}{2}$$

Possibilità: troviamo N oppure Start > End (non troviamo N)

Ordinamento

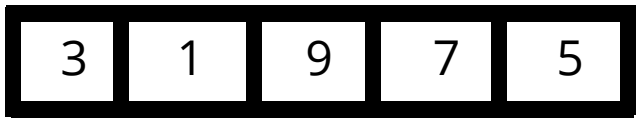
Credits A. Montenegro

Ordinamento

Vedremo due algoritmi iterativi, ed uno ricorsivo.

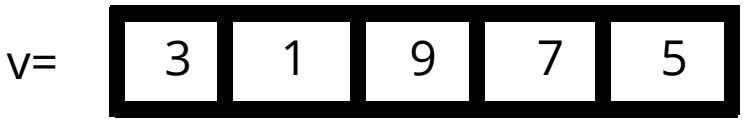
Selection Sort

- Ordinamento Iterativo 1



Selection Sort

- Ordinamento Iterativo 1

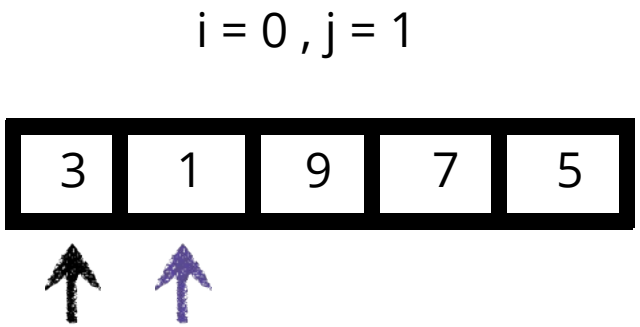


Logica

- Fisso un elemento in posizione i fino a LEN
- Scorro l’array in avanti a partire da quello in posizione j = i + 1 fino a LEN - 1
- Se V[i] > V[j] li scambio
- Alla fine del secondo ciclo fino a i siamo sicuri che l’i-esimo elemento più piccolo è in posizione i

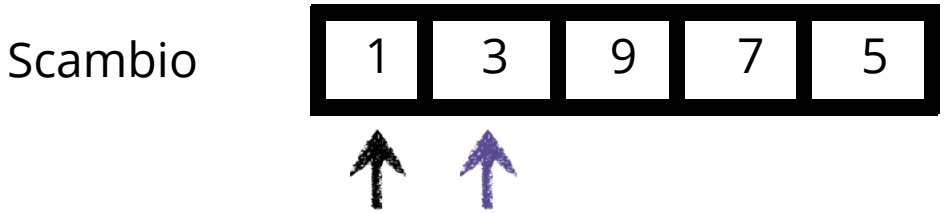
Selection Sort

- Ordinamento Iterativo 1



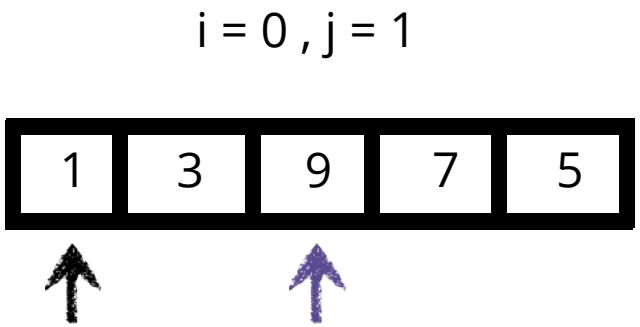
Selection Sort

- Ordinamento Iterativo 1



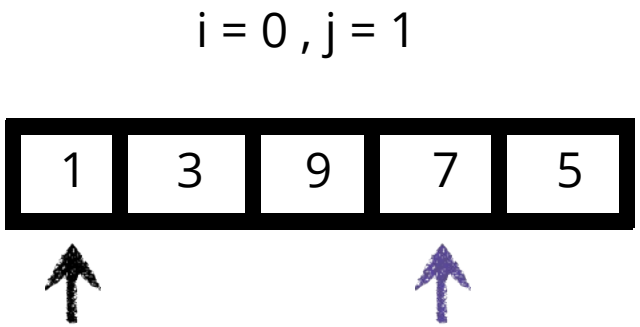
Selection Sort

- Ordinamento Iterativo 1



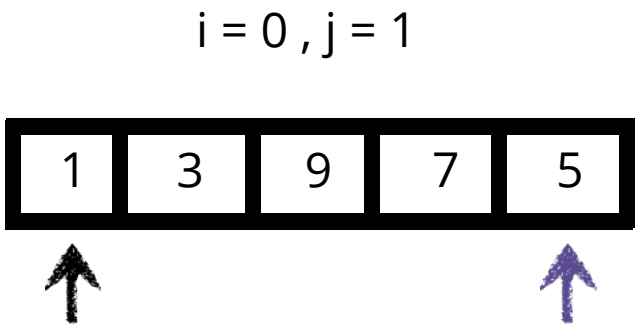
Selection Sort

- Ordinamento Iterativo 1



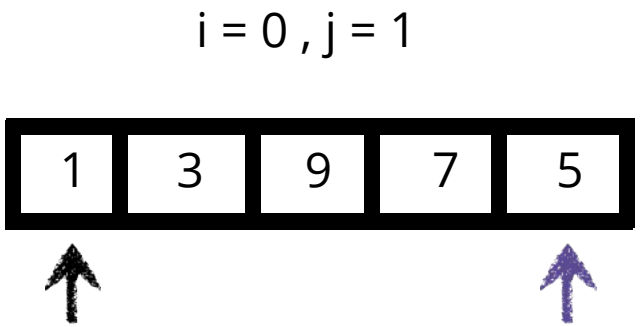
Selection Sort

- Ordinamento Iterativo 1



Selection Sort

- Ordinamento Iterativo 1

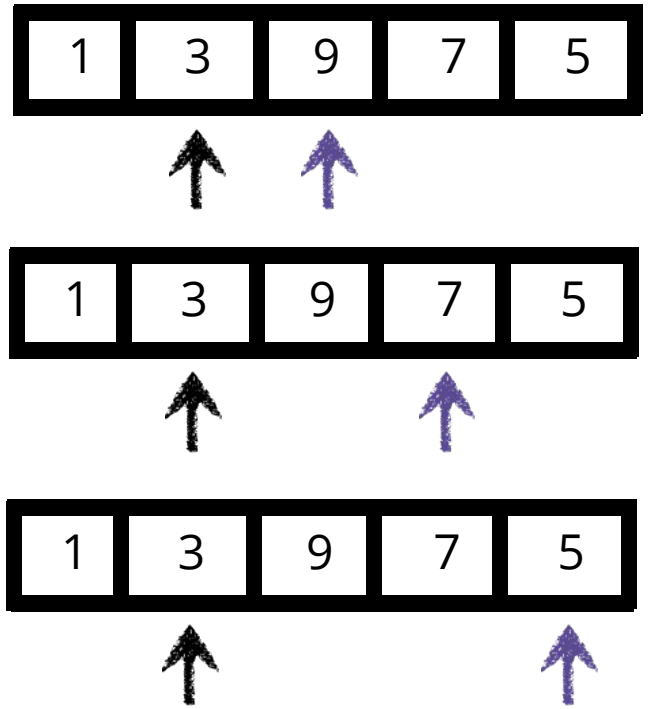


- Ho finito il loop interno, so che l’elemento più piccolo è in prima posizione. Ora rientro nel loop più esterno ed incremento i

Selection Sort

- Ordinamento Iterativo 1

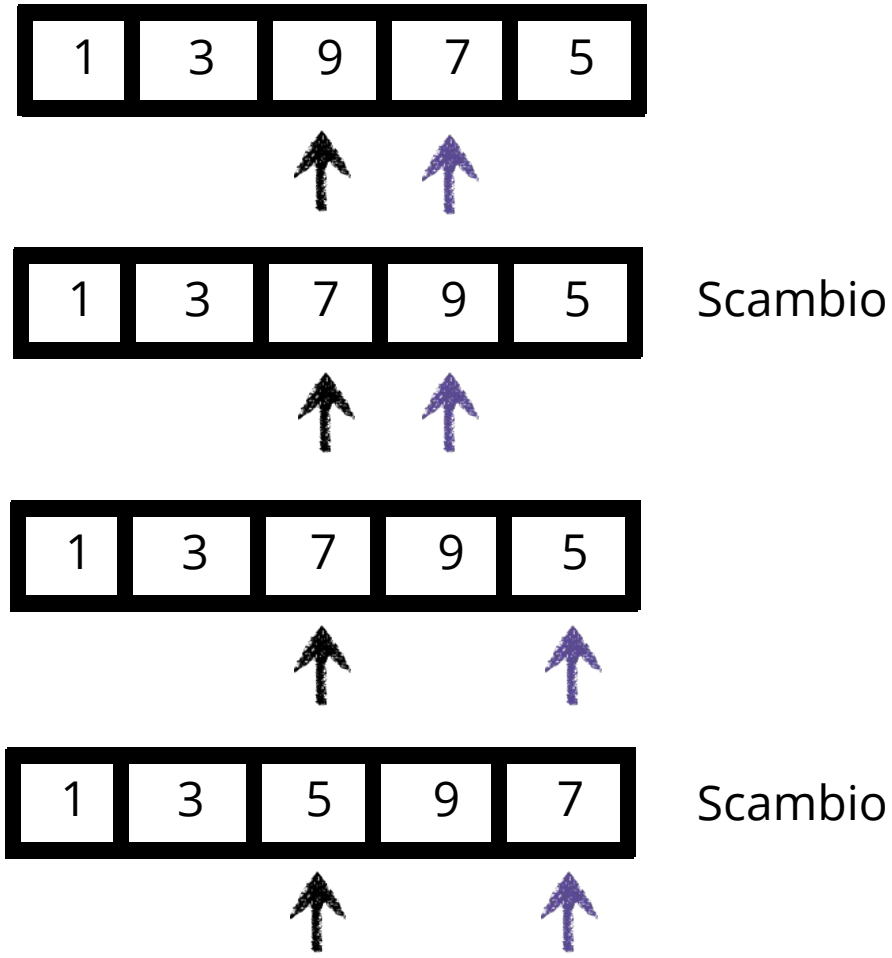
i = 1 , j = 2



Selection Sort

- Ordinamento Iterativo 1

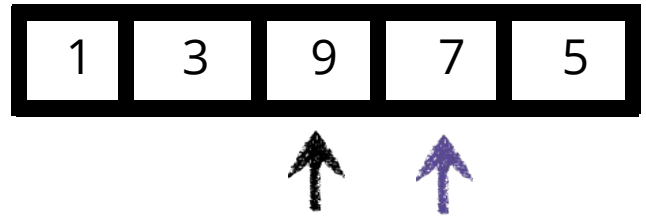
i = 2, j = 3

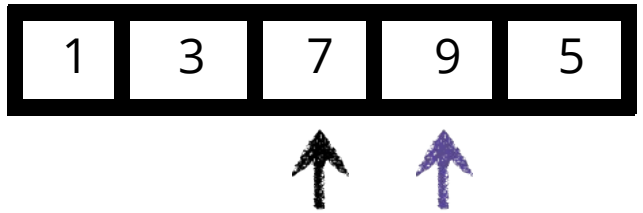


Selection Sort

- Ordinamento Iterativo 1

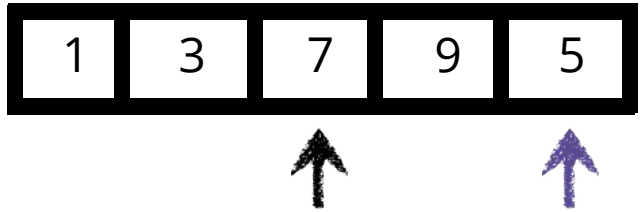
i = 2, j = 3

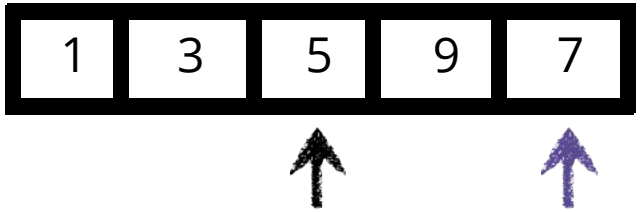




Scambio

Il terzo elemento più piccolo è in terza posizione



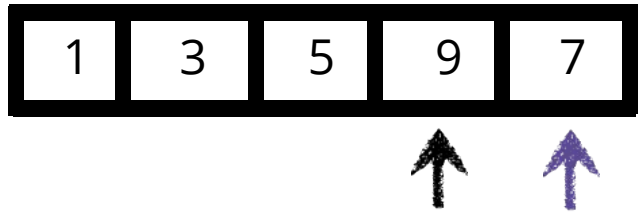


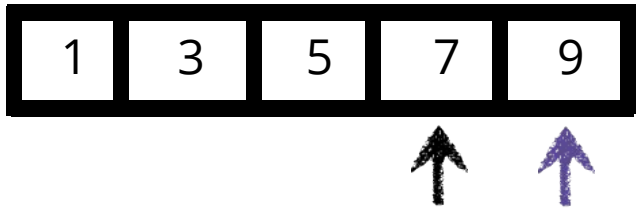
Scambio

Selection Sort

- Ordinamento Iterativo 1

i = 3, j = 4



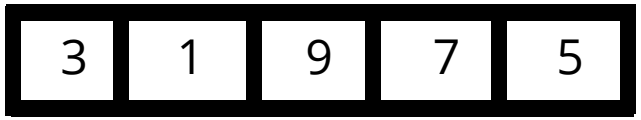


Scambio

Fine!

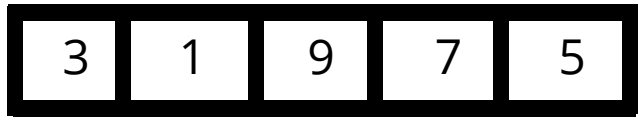
Bubble Sort

- Ordinamento Iterativo 2



Bubble Sort

- Ordinamento Iterativo 2

V = 

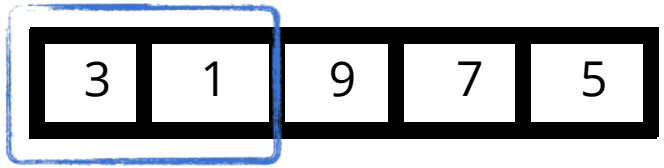
Logica

- Fisso un elemento in posizione i (scorro fino a LEN - 1)
- Scorro l'array in avanti a partire da quello in posizione j = 0 e fino alla posizione LEN - i - 1
- Se V[j] > V[j+1] li scambio (si usa una finestra di 2 elementi)
- Alla fine del secondo ciclo siamo sicuri che gli elementi da i in poi sono ordinati
- Ottimizzazione: se in una passata non ho cambiato nulla posso fermarmi

Bubble Sort

- Ordinamento Iterativo 2

i = 0 -> LEN - 1, j = 0 -> LEN - 1

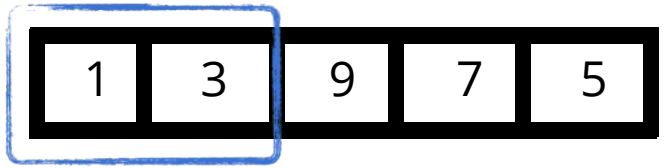


Bubble Sort

- Ordinamento Iterativo 2

i = 0 -> LEN - 1, j = 0 -> LEN - 1

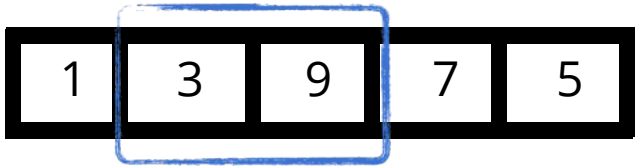
Scambio



Bubble Sort

- Ordinamento Iterativo 2

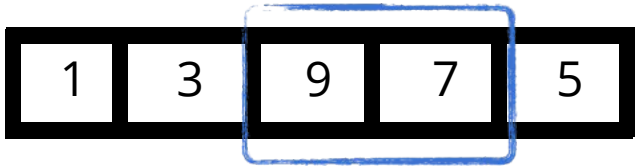
i = 0 -> LEN - 1, j = 0 -> LEN - 1



Bubble Sort

- Ordinamento Iterativo 2

i = 0 -> LEN - 1, j = 0 -> LEN - 1

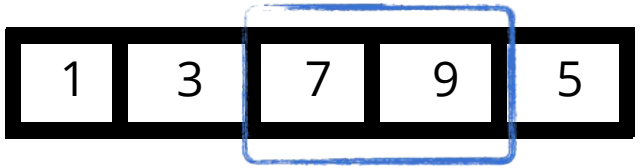


Bubble Sort

- Ordinamento Iterativo 2

i = 0 -> LEN - 1, j = 0 -> LEN - 1

Scambio



Bubble Sort

- Ordinamento Iterativo 2

i = 0 -> LEN - 1, j = 0 -> LEN - 1

Bubble Sort

- Ordinamento Iterativo 2

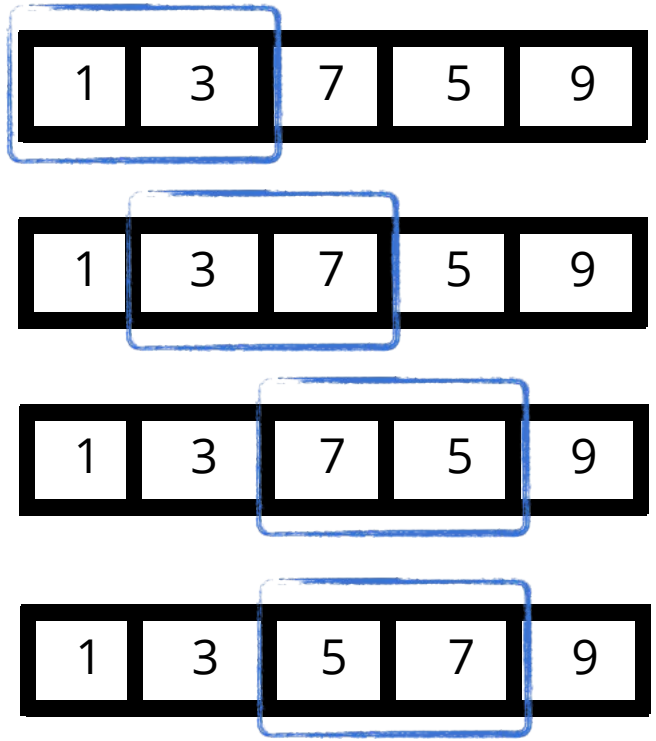
i = 0 -> LEN - 1, j = 0 -> LEN - 1

Scambio

Bubble Sort

- Ordinamento Iterativo 2

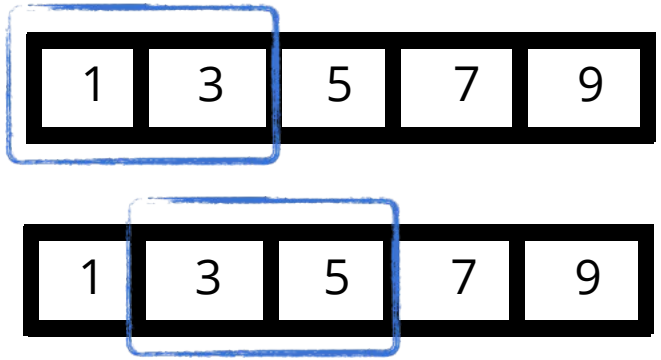
i = 1 -> LEN - 1, j = 0 -> LEN - 2



Bubble Sort

- Ordinamento Iterativo 2

i = 1 -> LEN - 1, j = 0 -> LEN - 3



Fine!
Nell’ultima passata dell’array non ho effettuato scambi.

Selection Sort vs Bubble Sort

Ordinamento Iterativo

- Entrambi non sono algoritmi ottimi (hanno la stessa complessità nel caso peggiore)
- Bubble Sort fa più scambi di Selection Sort
- Se i dati da scambiare sono pesanti, è preferibile Selection Sort
- Bubble Sort può essere **ottimizzato** fermandosi se non sono avvenuti scambi

Merge Sort

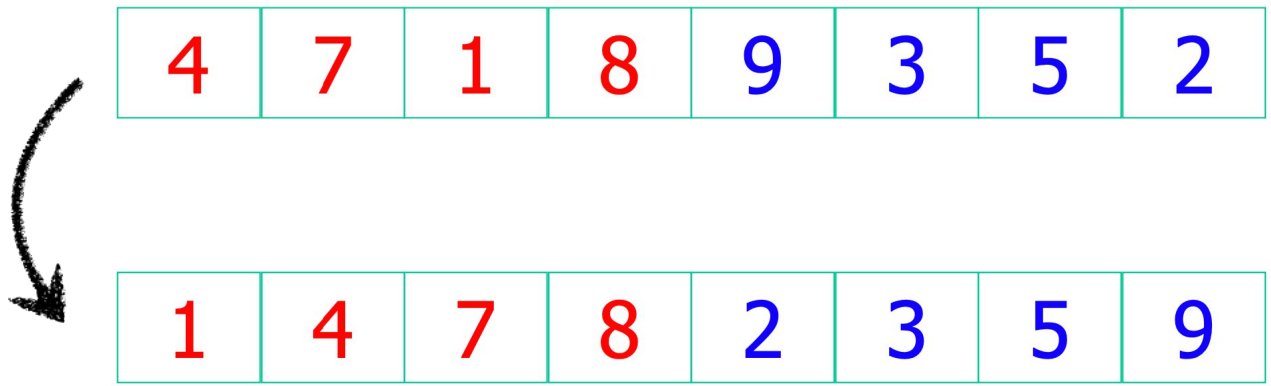
Ordinamento Ricorsivo

Logica

- Si divide l’array in due
- Si copia in un nuovo array in modo ordinato, rendendo gli elementi da entrambe le metà dell’array

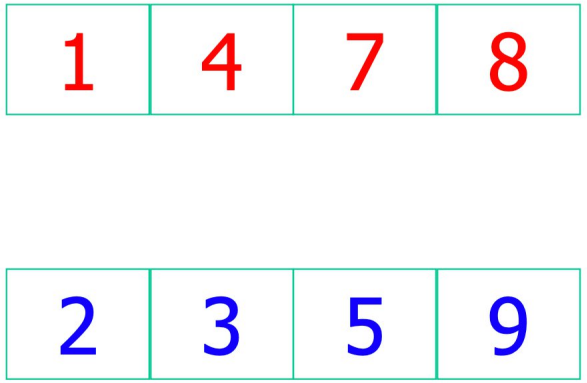
Merge Sort

Ordinamento Ricorsivo



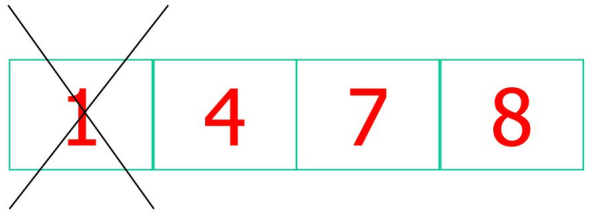
Merge Sort

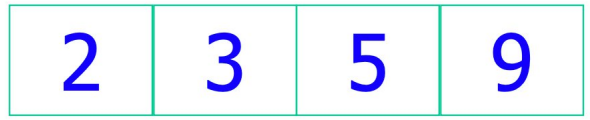
Ordinamento Ricorsivo



Merge Sort

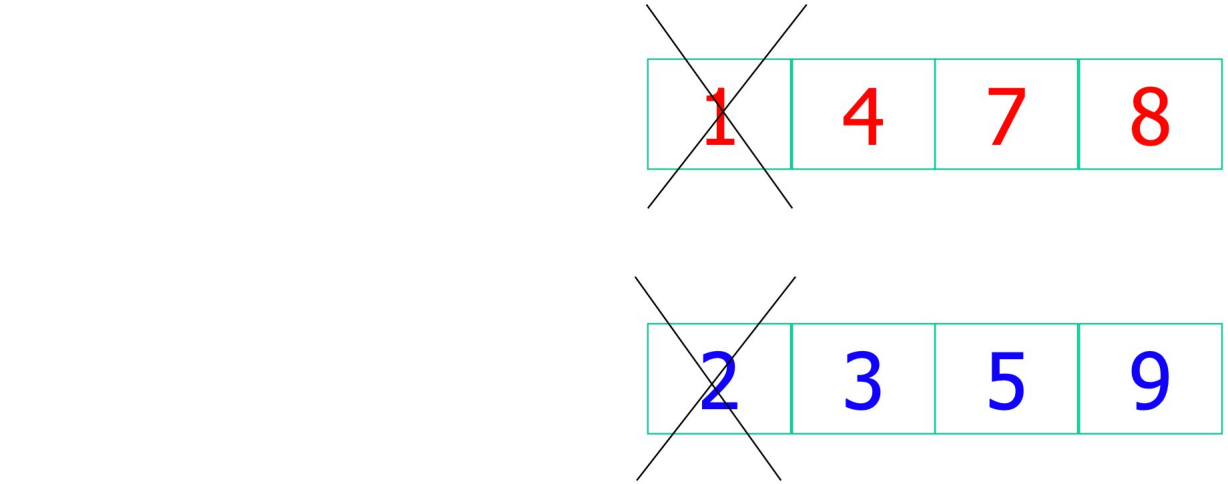
Ordinamento Ricorsivo





Merge Sort

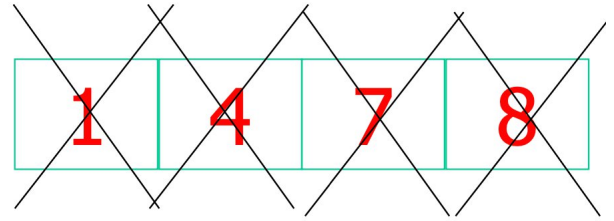
Ordinamento Ricorsivo

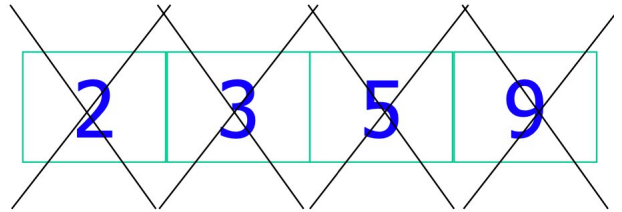


Merge Sort

Ordinamento Ricorsivo

[...]





1	2	3	4	5	7	8	9

Merge Sort

Ordinamento Ricorsivo

Divisione a metà

Caso base: porzioni di dimensione 1

4	7	1	8	9	3	5	2
4	7	1	8	9	3	5	2
4	7	1	8	9	3	5	2
4	7	1	8	9	3	5	2
4	7	1	8	3	9	2	5
1	4	7	8	2	3	5	9
1	2	3	4	5	7	8	9

Merge di tutte le porzioni

Array ordinato!

Thank You!

Mail: luca1.alessandrini@polimi.it

Website: https://alessandriniluca.github.io/