Algoritmi e Strutture Dati

Capitolo 4
Ordinamento: Selection e Insertion Sort

Ordinamento

Dato un insieme S di n elementi presi da un dominio totalmente ordinato, ordinare S in ordine non crescente o non decrescente.

- Esempi: ordinare una lista di nomi alfabeticamente, o un insieme di numeri reali, o un insieme di stringhe alfanumeriche in ordine lessicografico, etc.
- Ricorda: È possibile effettuare ricerche in array ordinati di n elementi in tempo O(log n) (ricerca binaria)

Copyright © 2004 - The McGraw - Hill Companies, srl

Formalizzazione del problema dell'ordinamento (non decrescente)

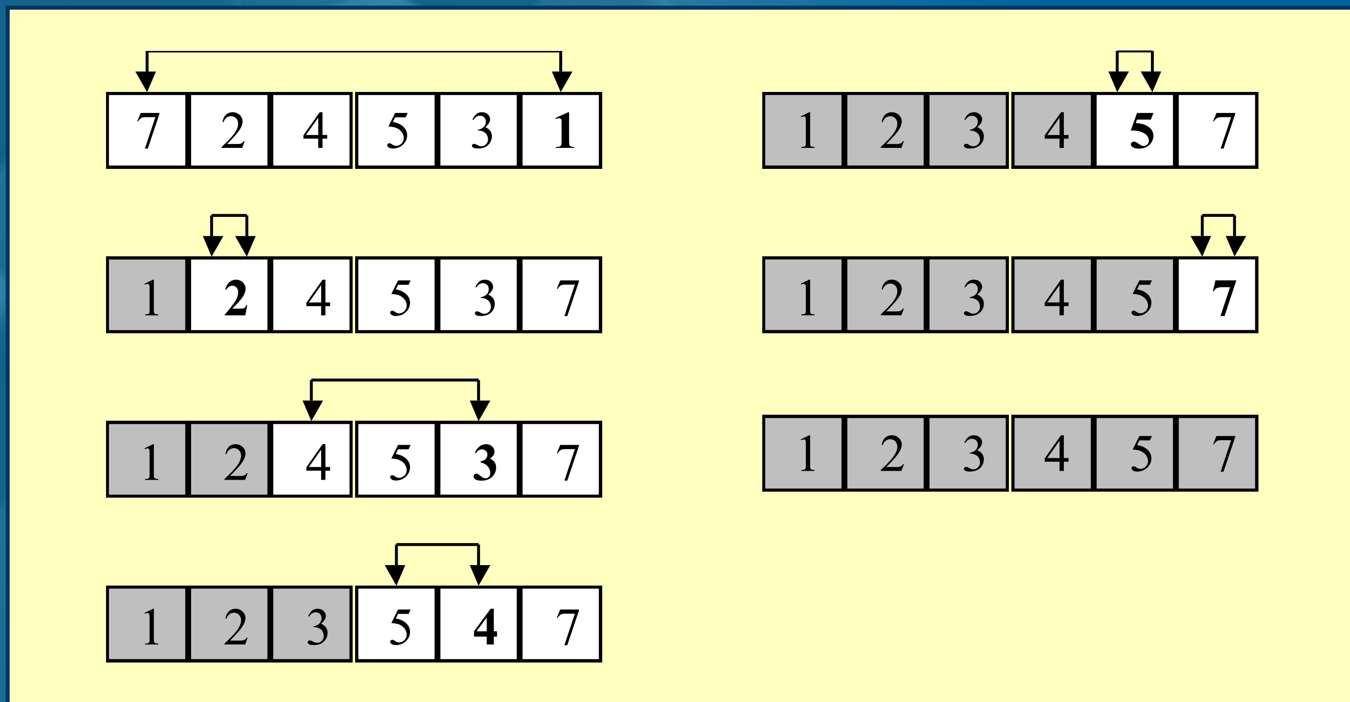
- **Input:** una sequenza di n numeri (interi) $<a_1,a_2,\ldots,a_n>$
  (NOTA: la dimensione dell'input è n)

- **Output:** una permutazione $\{1,2,\ldots,n\} \rightarrow \{i_1,i_2,\ldots,i_n\}$, ovvero un riarrangiamento $<a_{i_1}, a_{i_2},\ldots, a_{i_n}>$ della sequenza di input in modo tale che $a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_n}$

Copyright © 2004 - The McGraw - Hill Companies, srl

SelectionSort

Approccio incrementale: assumendo che i primi k-1 elementi siano ordinati e siano i k-1 elementi più piccoli della sequenza, estende l'ordinamento ai primi k elementi scegliendo il **minimo** degli elementi non ancora ordinati in posizione k, k+1,…, n, e mettendolo in posizione k



Copyright © 2004 - The McGraw - Hill Companies, srl

```
SelectionSort (A)
1.  for k=1 to n-1 do
2.      m = k
3.      for j=k+1 to n do
4.          if (A[j] < A[m]) then m=j
5.      scambia A[m] con A[k]
```

NOTA: Assumiamo che il primo elemento dell’array sia in A[1]

- Linea 1: k mantiene l’indice dove andrà spostato il minimo degli elementi in posizione k, k+1, …, n.
- Linea 2: m mantiene l’indice dell’array in cui si trova il minimo corrente
- Linee 3-4: ricerca del minimo fra gli elementi A[k],…,A[n] (m viene aggiornato con l’indice dell’array in cui si trova il minimo corrente)
- Linea 5: il minimo è spostato in posizione k (si noti che questa operazione richiede 3 operazioni elementari di assegnamento)

Copyright © 2004 - The McGraw - Hill Companies, srl

Correttezza

- Si dimostra facendo vedere che alla fine del generico passo k (k=1,…, n-1) si ha che : (i) i primi k elementi sono ordinati in ordine non decresente e (ii) contengono i k elementi più piccoli dell'array
- Induzione su k:
  - k=1: Alla prima iterazione viene semplicemente selezionato l'elemento minimo dell'array ⇒ (i) e (ii) banalmente verificate.
  - k>1. All'inizio del passo k i primi k-1 elementi sono ordinati e sono i k-1 elementi più piccoli nell'array (ipotesi induttiva). Allora la tesi segue dal fatto che l'algoritmo seleziona il minimo dai restanti n-k+1 elementi e lo mette in posizione k. Infatti:
    - (i) l'elemento in posizione k non è mai più piccolo dei primi k-1 elementi, che non vengono mai spostati, e quindi per l'ipotesi induttiva i primi k elementi sono ordinati
    - (ii) i primi k elementi restano i minimi nell'array, per l'ipotesi induttiva e per come è stato scelto il k-esimo elemento

Copyright © 2004 - The McGraw - Hill Companies, srl

Complessità temporale

```
SelectionSort (A)
1.   for k=1 to n-1 do
2.       m = k
3.       for j=k+1 to n do
4.           if (A[j] < A[m]) then m=j
5.       scambia A[m] con A[k]
```

- riga 2: 1 assegnamento
- righe 3–4: n-k confronti (operaz. dominante)
- riga 5: 1 scambio (3 assegnamenti)

il tutto eseguito n-1 volte

$$T(n) = \sum_{k=1}^{n-1} (n-k) = \sum_{k=1}^{n-1} k = n\cdot(n-1)/2 = \Theta(n^2)$$

(conto solo le operazioni dominanti)

Si noti che T(n) è SEMPRE UGUALE ad un polinomio di 2° grado in n, e quindi la notazione Θ è perfettamente ESPRESSIVA del valore di T(n)

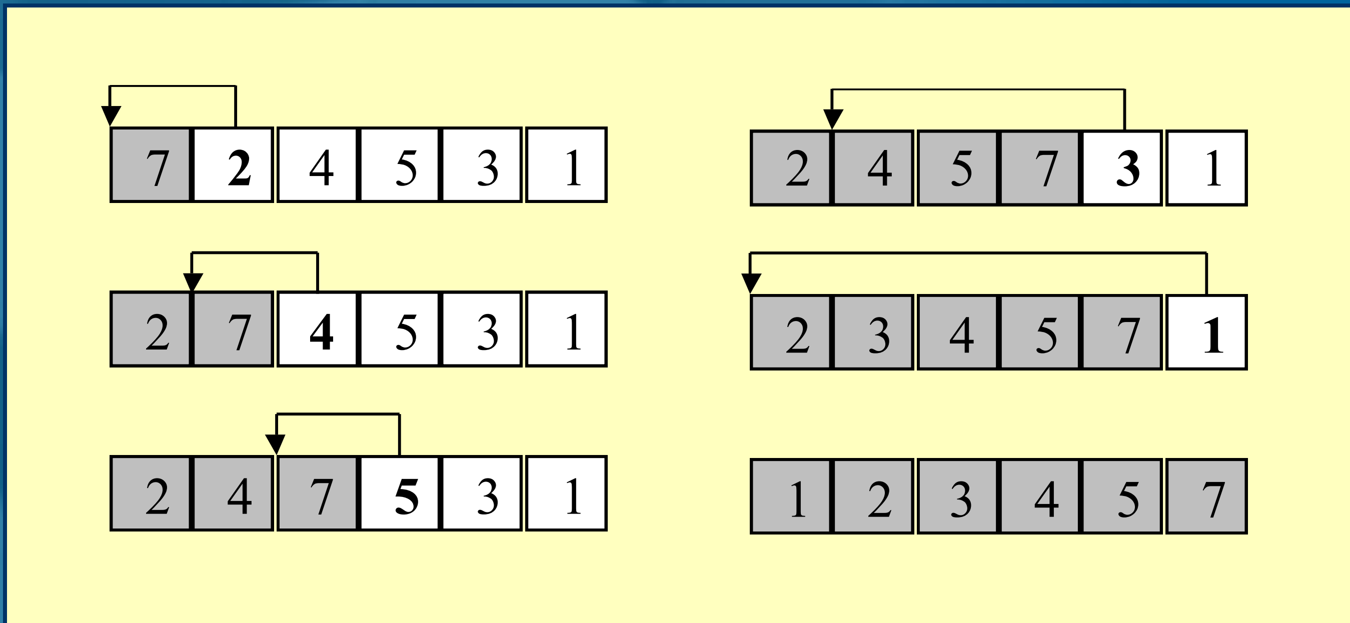
$$\Rightarrow T_{worst}(n) = T_{best}(n) = T_{avg}(n) = \Theta(n^2)$$

Possiamo fare meglio?

Copyright © 2004 - The McGraw - Hill Companies, srl
Mc Graw Hill

InsertionSort

Approccio incrementale: assumendo che i primi k elementi siano ordinati (attenzione, non necessariamente i k elementi più piccoli della sequenza!), estende l'ordinamento ai primi k+1 elementi, inserendo l'elemento in posizione k+1-esima nella giusta posizione rispetto ai primi k elementi



Copyright © 2004 - The McGraw - Hill Companies, srl

Una prima versione dell'InsertionSort

```
InsertionSort (A)
1.      for k=1 to n-1 do
2.          x = A[k+1]
3.          for j=1 to k+1 do
4.              if (A[j] > x) then break
5.          if (j < k+1) then
6.              for t=k downto j do A[t+1]= A[t]
7.              A[j]=x
```

- Linea 2: elemento x=A[k+1] da inserire nella posizione che gli compete
- Linee 3 e 4: individuano la posizione j in cui va messo x
- Linee 5 e 6: se la posizione j è diversa da k+1, si fa spazio per inserire x, "shiftando" tutti gli elementi da j a k verso destra

Copyright © 2004 - The McGraw - Hill Companies, srl

Correttezza

- Si dimostra facendo vedere che alla fine del generico passo k (k=1,…, *n*-1) i primi k+1 elementi sono ordinati in ordine non decrescente (si noti la differenza con il Selection Sort, in cui invece dovevamo far vedere che i primi k elementi erano i più piccoli)
- Induzione su k:
  - k=1: banale: si riordinano A[1] e A[2];
  - k>1: All'inizio del passo k i primi k elementi sono ordinati (ipotesi induttiva). Allora la tesi segue dal fatto che l'algoritmo inserisce A[k+1] nella giusta posizione rispetto alla sequenza A[1],…,A[k]

Copyright © 2004 - The McGraw - Hill Companies, srl

Complessità temporale

```
InsertionSort (A)
1.    for k=1 to n-1 do
2.        x = A[k+1]
3.        for j=1 to k+1 do
4.            if (A[j] > x) then break
5.        if (j < k+1) then
6.            for t=k downto j do A[t+1]= A[t]
7.            A[j]=x
```

- $j^* \le k+1$ confronti
- $k+1-j^*$ assegnamenti
- → $k+1$ oper. (oper. dom.)
- → il tutto eseguito n-1 volte

$$T(n) = \sum_{k=1}^{n-1} (k+1) = n(n+1)/2 - 1 = \Theta(n^2)$$

(conto solo le oper. dominanti)

$$T_{worst}(n) = T_{best}(n) = T_{avg}(n) = \Theta(n^2)$$

Possiamo fare meglio?

Copyright © 2004 - The McGraw - Hill Companies, srl

Una variante dell'IS più efficiente

```
InsertionSort2 (A)
1.    for k=1 to n-1 do
2.        x = A[k+1]
3.        j = k
4.        while j > 0 e A[j] > x do
5.            A[j+1] = A[j]
6.            j= j-1
7.        A[j+1]=x
```

$t_k \leq 2k$ assegnam. (oper. dom.)

il tutto eseguito n-1 volte

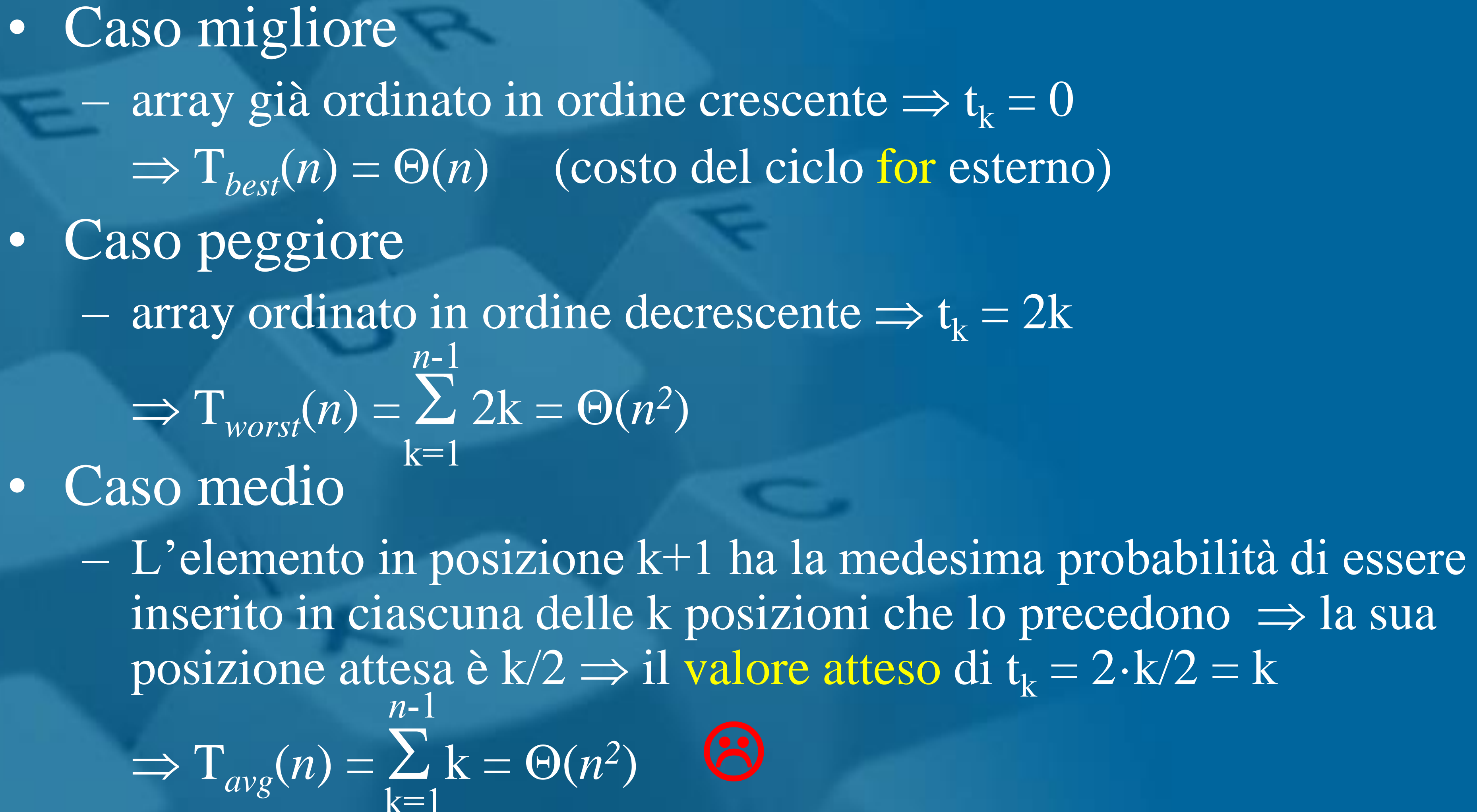
$$T(n) = \sum_{k=1}^{n-1} t_k \leq \sum_{k=1}^{n-1} 2k = n(n-1)$$

$$\Rightarrow T(n) = O(n^2)$$

Si noti che T(n) è AL PIÙ UGUALE ad un polinomio di 2° grado in n, e quindi la notazione O è perfettamente ESPRESSIVA del valore di T(n)

Copyright © 2004 - The McGraw - Hill Companies, srl

Caso migliore, peggiore, medio



- Caso migliore
  - array già ordinato in ordine crescente $\Rightarrow t_k = 0$

    $\Rightarrow T_{best}(n) = \Theta(n)$ (costo del ciclo for esterno)
- Caso peggiore
  - array ordinato in ordine decrescente $\Rightarrow t_k = 2k$

    $$\Rightarrow T_{worst}(n) = \sum_{k=1}^{n-1} 2k = \Theta(n^2)$$
- Caso medio
  - L'elemento in posizione k+1 ha la medesima probabilità di essere inserito in ciascuna delle k posizioni che lo precedono $\Rightarrow$ la sua posizione attesa è k/2 $\Rightarrow$ il valore atteso di $t_k = 2 \cdot k/2 = k$

    $$\Rightarrow T_{avg}(n) = \sum_{k=1}^{n-1} k = \Theta(n^2)$$ ☹

Copyright © 2004 - The McGraw - Hill Companies, srl

Legge di Murphy?

« Se qualcosa può andar male, lo farà. »

In realtà, negli algoritmi iterativi il caso medio costa spesso come il caso peggiore (asintoticamente), in quanto le strutture di controllo fondamentali di tali algoritmi sono i cicli, e spesso il caso medio implica l'esecuzione della metà delle istruzioni di un ciclo, senza quindi avere un abbattimento asintotico della complessità.

Mc Graw Hill

Complessità spaziale

Ricordiamo che oltre alla complessità temporale dobbiamo valutare anche la complessità spaziale di un algoritmo, ovvero lo spazio di memoria necessario per ospitare i dati manipolati dall'algoritmo (input, output e memoria di lavoro).

La complessità spaziale del Selection Sort e dell'Insertion Sort è $\Theta(n)$

Nota: Se la complessità spaziale per la rappresentazione dell'input di un certo algoritmo è $\Theta(g(n))$, e se tale algoritmo deve ispezionare l'intero input per essere corretto, allora la complessità temporale dell'algoritmo è $\Omega(g(n))$, ovviamente.

Copyright © 2004 - The McGraw - Hill Companies, srl

Conseguenze per il problema dell'ordinamento

La dimensione dell'input per il problema dell'ordinamento è $\Theta(n)$, e **qualsiasi** algoritmo che risolve il problema dell'ordinamento deve ovviemante ispezionare **tutti** i dati in ingresso per ordinarli, e quindi avrà complessità temporale $T(n)=\Omega(n)$

Tutti gli algoritmi che risolveranno il problema dell'ordinamento avranno una complessità temporale $\Omega(n)$

Nota: Si ricordi che per il problema della ricerca, la dimensione dell'input è $\Theta(n)$, ma se assumiamo che l'input è ordinato, allora non è necessario ispezionare l'intero input per trovare un certo elemento (infatti, ad esempio la ricerca binaria costa $O(\log n)$)

Copyright © 2004 - The McGraw - Hill Companies, srl

Riepilogo

	Caso migliore	Caso medio	Caso peggiore	T(n)	S(n)
`Selection Sort`	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
`Insertion Sort 1`	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
`Insertion Sort 2`	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(n^2)$	$\Theta(n)$

$\Rightarrow$ Lower bound **temporale**: $\Omega(n)$"banale" (esplorazione input)
Upper bound **temporale**: $O(n^2)$ `Insertion Sort 2`

Approfondimento: Analizzare la complessità computazionale di un'ulteriore variante dell'IS in cui la fase di ricerca della giusta posizione in cui inserire un elemento viene eseguita utilizzando la ricerca binaria.

Copyright © 2004 - The McGraw - Hill Companies, srl

Esercizi di approfondimento per casa

- Illustrare l'evoluzione di Insertion-Sort 2 applicata all'array A=<31,41,59,26,41,58>

- Riscrivere la procedura Insertion-Sort 1 per ordinare in modo non crescente

- Riscrivere la procedura Insertion-Sort 2 per ordinare l'array da destra verso sinistra

Copyright © 2004 - The McGraw - Hill Companies, srl

Mc Graw Hill

Approfondimento: Algoritmo Bubble Sort

Definizione

Sia $A=\langle a_1,a_2,\ldots \ldots a_n\rangle$ una sequenza di n numeri. La coppia (a_i,a_j) è chiamata inversione (rispetto ad un ordinamento **crescente**) se $i<j$ ed $a_i > a_j$.

L'algoritmo Bubble Sort risolve il problema dell'ordinamento seguendo la seguente strategia:

✓ 1) Si scorre la sequenza A[1],… …,A[n], eliminando inversioni contigue (in tal modo si porta il massimo degli n elementi considerati nella posizione A[n]);

✓ 2) Si scorre la sequenza A[1],……,A[n-1], eliminando inversioni contigue (in tal modo si porta il massimo degli n-1 elementi considerati nella posizione A[n-1])

✓ n-1) Si scorre la sequenza A[1],A[2], eliminando inversioni contigue (in tal modo si porta il massimo dei due elementi considerati nella posizione A[2]).

Fornire un'implementazione del Bubble Sort che esegua esattamente i passi appena descritti, e analizzarne la complessità computazionale.