POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Gli Algoritmi di Ordinamento iterativi quadratici

Paolo Camurati

Insertion Sort

Siano dati N interi memorizzati in un vettore A con indici compresi tra l=0 e r=N-1.

Concettualmente il vettore A è diviso in 2 sottovettori:

- di sinistra: già ordinato
- di destra: ancora disordinato.

Inizialmente il sottovettore di sinistra contiene 1 elemento, quello di destra contiene N-1 elementi.

Un vettore di un solo elemento è ordinato per definizione.

Approccio

Paradigma incrementale:

- ad ogni passo si espande il sottovettore sinistro già ordinato inserendovi un elemento preso dal sottovettore destro ancora disordinato
- l'inserzione deve garantire che il sottovettore sinistro rimanga ordinato dopo l'inserimento (invarianza della proprietà di ordinamento)
- terminazione: tutti gli elementi sono stati inseriti ordinatamente, il sottovettore sinistro contiene N elementi, quello di destra è vuoto.

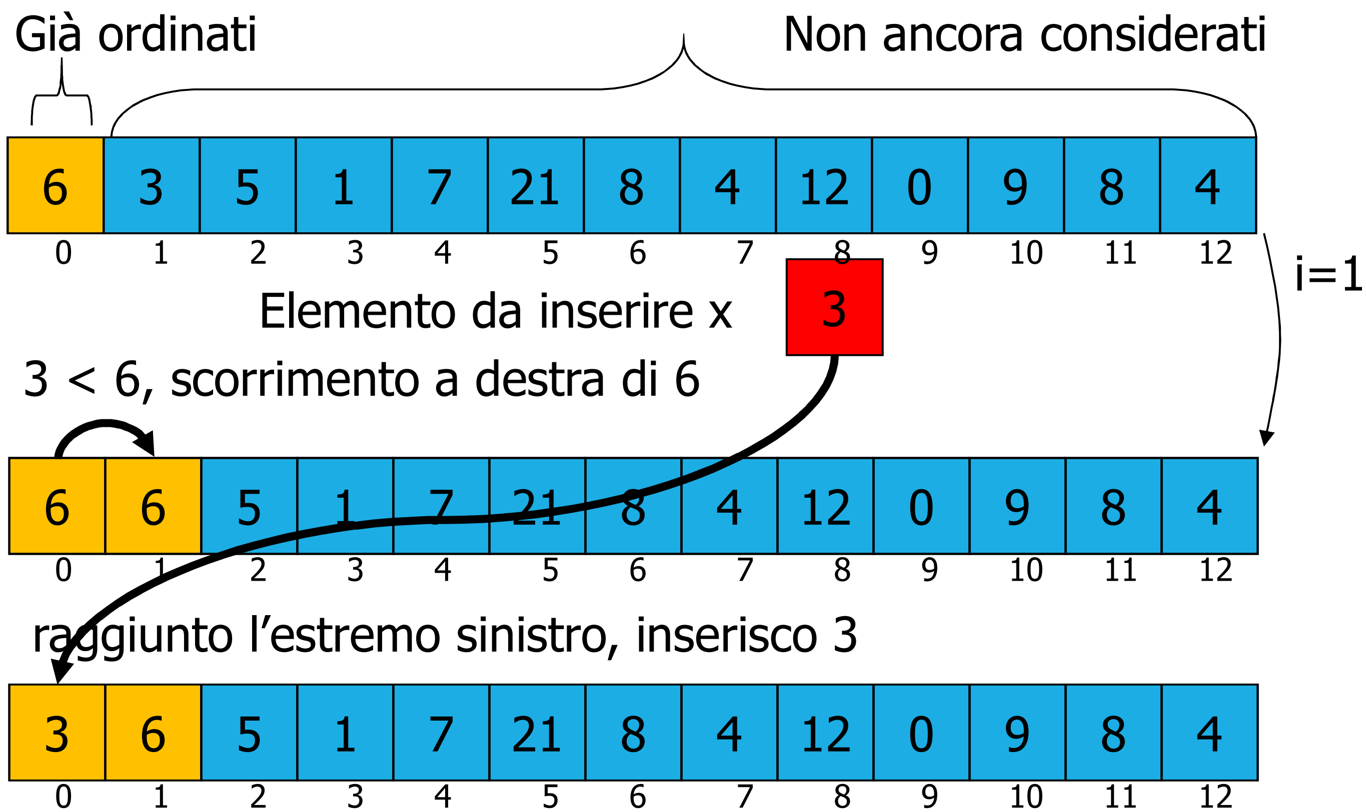
Passo i-esimo: inserimento ordinato

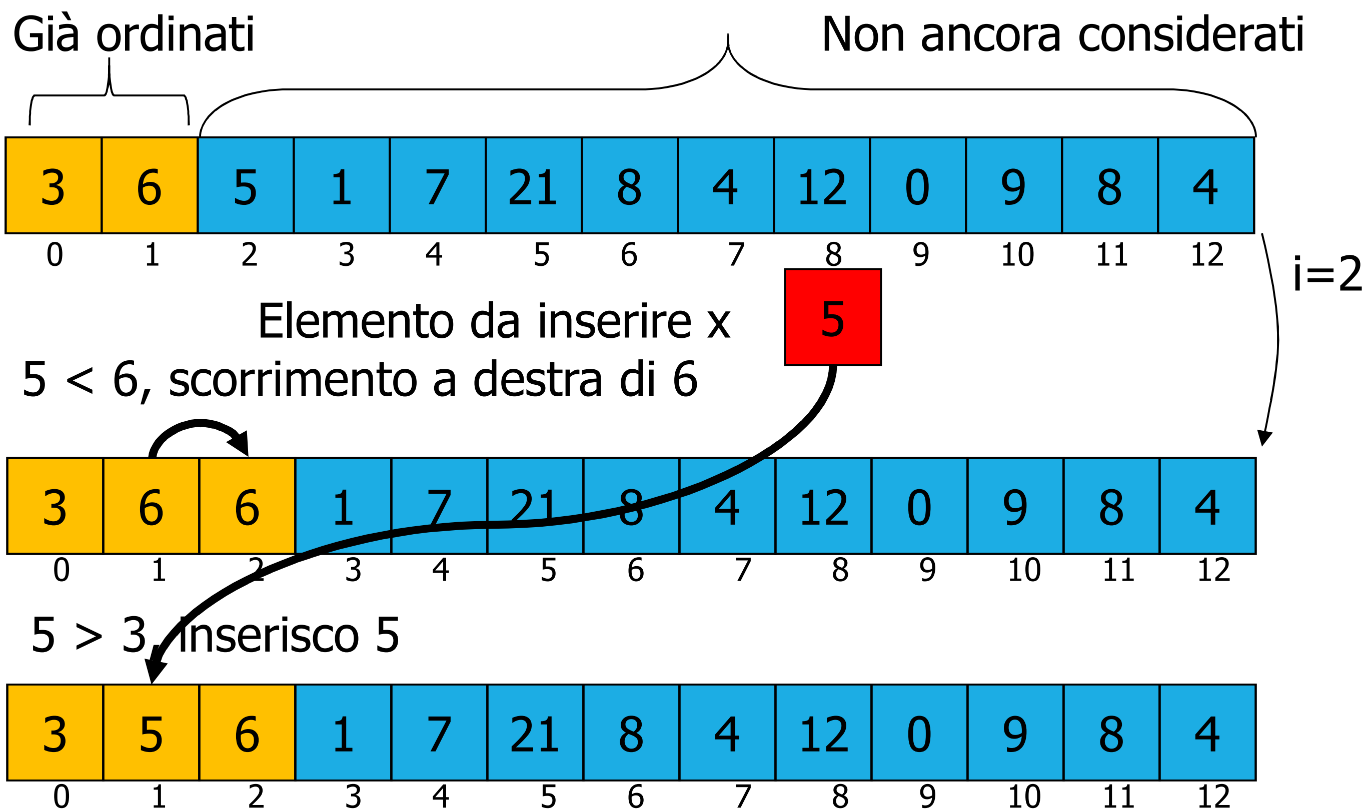
Al passo i-esimo si colloca nella posizione corretta nel sottovettore sinistro l'elemento $x = A_i$

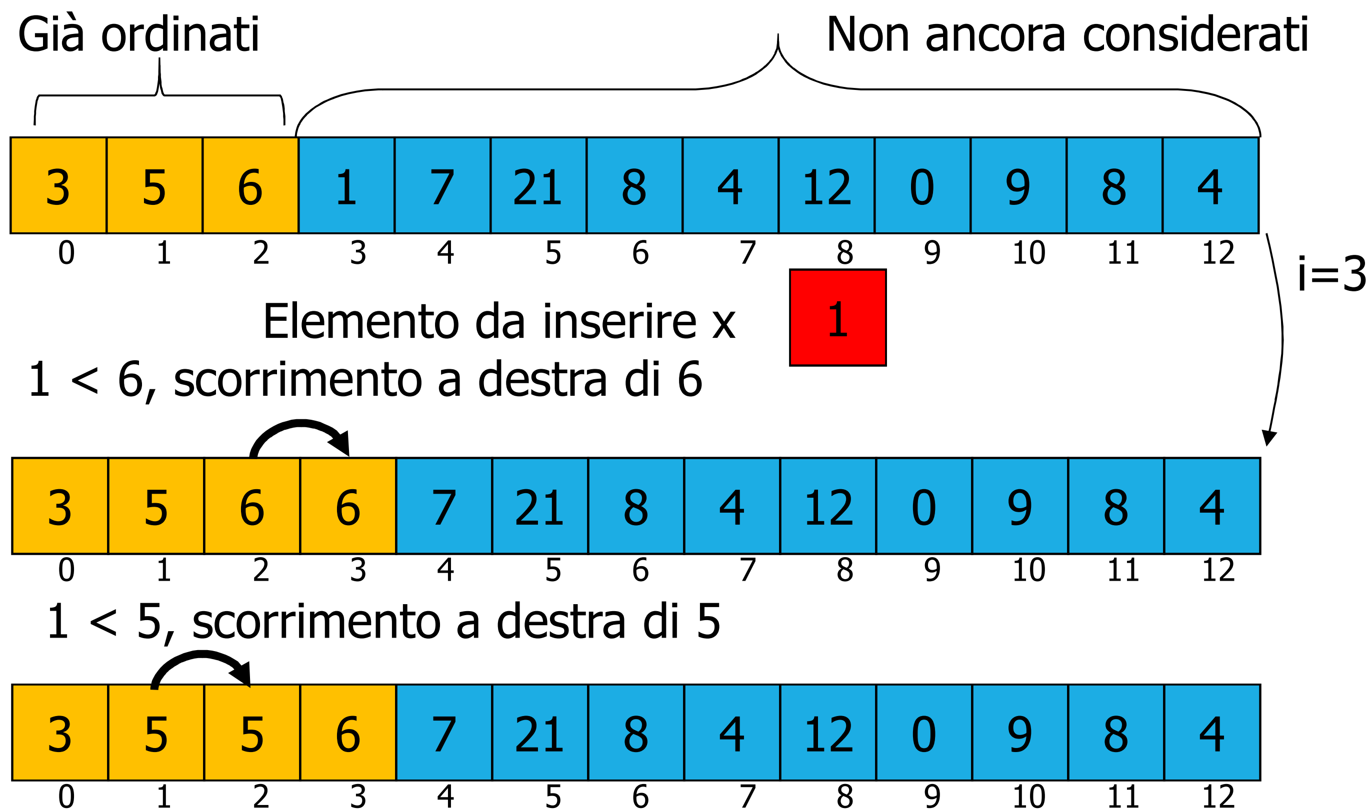
- si scandisce il sottovettore di sinistra (ordinato e compreso tra A_{i-1} e A_0) fino a trovare un elemento tale per cui $A_j > A_i$
- durante la scansione, si opera uno scorrimento (shift) a destra di una posizione degli elementi da A_j ad A_{i-1}

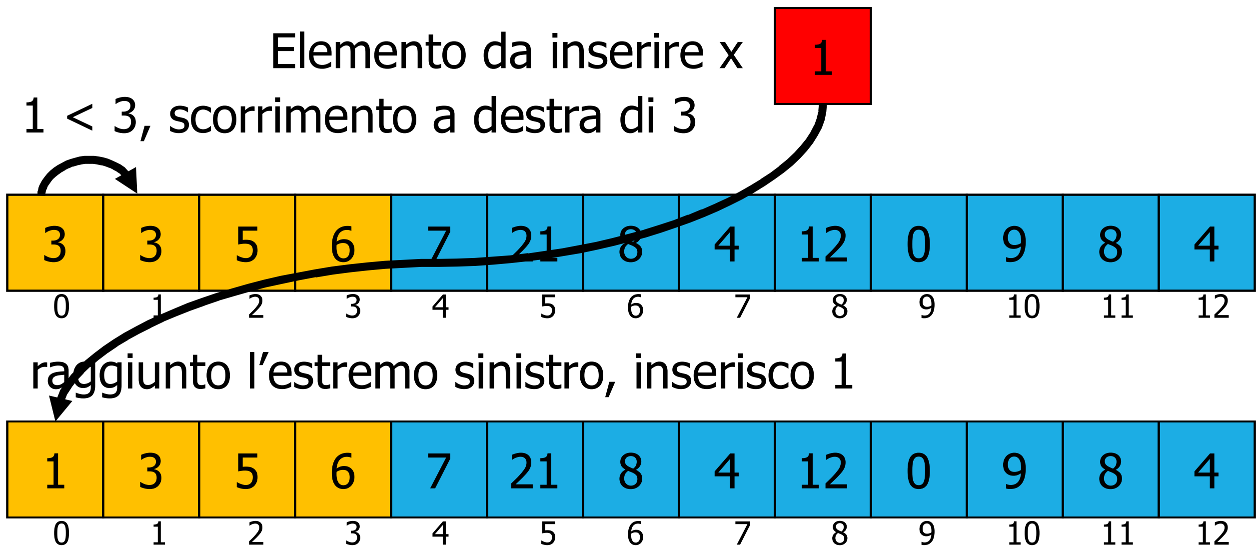
Quando il ciclo termina, si è trovata la posizione corretta in cui inserire A_i.

Esempio


Già ordinati
Non ancora considerati
6 3 5 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
i=1
Elemento da inserire x
3
3 < 6, scorrimento a destra di 6
6 6 5 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
raggiunto l’estremo sinistro, inserisco 3
3 6 5 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12


Già ordinati
Non ancora considerati
3 6 5 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
i=2
Elemento da inserire x
5
5 < 6, scorrimento a destra di 6
3 6 6 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
5 > 3, inserisco 5
3 5 6 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12

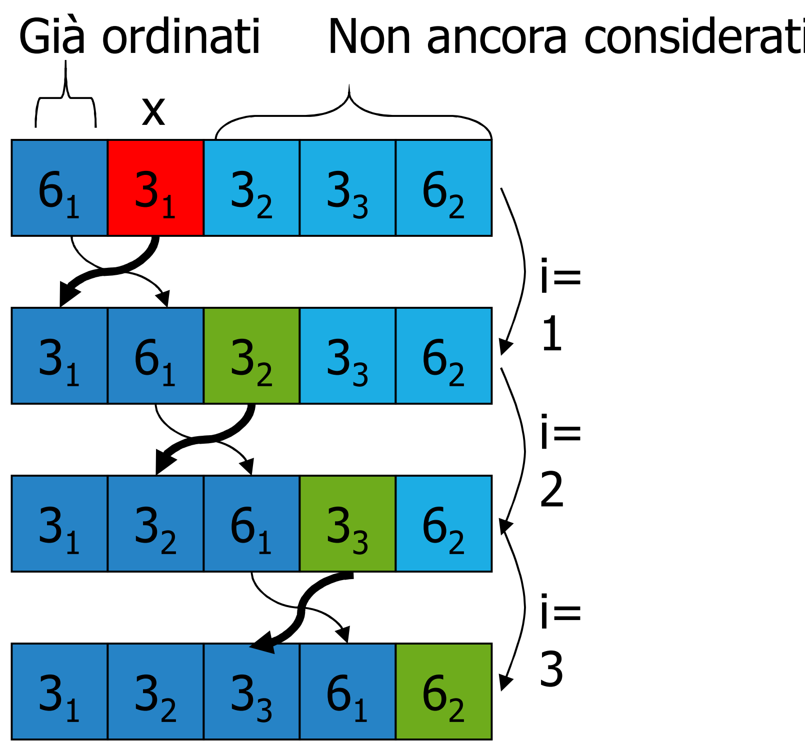

Già ordinati
Non ancora considerati
3 5 6 1 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
i=3
Elemento da inserire x
1
1 < 6, scorrimento a destra di 6
3 5 6 6 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
1 < 5, scorrimento a destra di 5
3 5 5 6 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12


Elemento da inserire x
1
1 < 3, scorrimento a destra di 3
3 3 5 6 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12
raggiunto l'estremo sinistro, inserisco 1
1 3 5 6 7 21 8 4 12 0 9 8 4
0 1 2 3 4 5 6 7 8 9 10 11 12

```
void InsertionSort(int A[], int N) {
  int i, j, l=0, r=N-1, x;
  for (i = l+1; i <= r; i++) {
    x = A[i];
    j = i - 1;
    while (j >= l && x < A[j]){
      A[j+1] = A[j];
      j--;
    }
    A[j+1] = x;
  }
}
```

Caratteristiche dell'Insertion sort

- in loco: oltre al vettore A si usa solo la variabile x
- stabile: se l'elemento da inserire è una chiave duplicata, non può mai «scavalcare» a SX un'occorrenza precedente della stessa chiave:



Analisi di complessità dell'Insertion sort

Analisi di complessità asintotica di caso peggiore:

- sulla base dei singoli passi delle istruzioni utilizzate
- sulla base del comportamento di alto livello dei costrutti del linguaggio usati

per determinare

- il numero di confronti
- il numero di scambi.

Analisi di dettaglio del numero di confronti

Ipotesi:

- tutte le istruzioni hanno costo unitario
- i parte da 1 e cresce fino a N-1 (per comodità non si usano r e l)

Istruzioni	# esecuzioni
`for(i=1; i<N; i++) {`	2N

- 2 istruzioni a ciascun passo (< e +)
- N esecuzioni in quanto si tiene conto anche di quella finale quando la condizione i<N fallisce

Istruzioni	# esecuzioni
`for(i=1; i<N; i++) {`	2N
`x = A[i];`	N-1
`j = i-1;`	N-1

N-1 esecuzioni in quanto eseguite solo se la condizione i<N è vera

Istruzioni	# esecuzioni
`for(i=1; i<N; i++) {`	2N
`x = A[i];`	N-1
`j = i-1;`	N-1
`while (j>=0 && x<A[j]){`	$2\sum_{j=2}^{N} j$

- 2 istruzioni a ciascun passo (>=e <)
- N-1 esecuzioni del ciclo nel caso peggiore (vettore ordinato decrescente)
- j operazioni in ogni esecuzione del ciclo in quanto si tiene conto anche di quella finale quando la condizione j>=0 fallisce

Istruzioni	# esecuzioni
`for(i=1; i<N; i++) {`	2N
`  x = A[i];`	N-1
`  j = i-1;`	N-1
`  while (j>=0 && x<A[j]){`	$2\sum_{j=2}^{N} j$
`    A[j+1] = A[j];`	$\sum_{j=2}^{N}(j-1)$
`    j--;`	$\sum_{j=2}^{N}(j-1)$
`  }`	

- N-1 esecuzioni del ciclo nel caso peggiore (vettore ordinato decrescente)
- j-1 operazioni in ogni esecuzione del ciclo in quanto eseguite solo se la condizione del ciclo è vera

Istruzioni	# esecuzioni
`for(i=1; i<N; i++) {`	2N
`  x = A[i];`	N-1
`  j = i-1;`	N-1
`  while (j>=0 && x<A[j]){`	$2\sum_{j=2}^{N} j$
`    A[j+1] = A[j];`	$\sum_{j=2}^{N}(j-1)$
`    j--;`	$\sum_{j=2}^{N}(j-1)$
`  }`	
`  A[j+1] = x;`	N - 1
`}`	

N-1 esecuzioni in quanto eseguite solo se la condizione i<N è vera

Quindi:

$T(N) = 2N+(N-1)+(N-1)+2\sum_{j=2}^{N} j+\sum_{j=2}^{N}(j-1) + \sum_{j=2}^{N}(j-1)+(N-1)$

Ricordando che:

$\sum_{j=2}^{N} j = 2+3+\ldots N = N(N+1)/2 -1$

$\sum_{j=2}^{N}(j-1) = N(N-1)/2$

$T(N) = 2N + 3(N-1) + 2(N(N+1)/2 -1) + 2(N(N-1)/2) = 2N^2 + 6N -6$

$T(N) = O(N^2)$: il numero di confronti nel caso peggiore cresce quadraticamente.

Analisi di alto livello

Due cicli annidati:

- esterno: N-1 esecuzioni
- interno nel caso peggiore: i esecuzioni all'i-esima iterazione di quello esterno

Complessità:

$T(N) = 1+2+3+ \ldots +(N-2)+(N-1)$

$= \Sigma_{1 \leq i < N}\, i = N(N-1)/2$

T(N) cresce quadraticamente in N.

progressione aritmetica finita di ragione 1 (Gauss, fine XVII sec.)

$T(N) = O(N^2)$: il numero di scambi e di confronti nel caso peggiore cresce quadraticamente.

Bubble/Exchange sort

Siano dati N interi memorizzati in un vettore A con indici compresi tra l=0 e r=N-1.
Concettualmente il vettore A è diviso in 2 sottovettori:

- di destra: già ordinato
- di sinistra: ancora disordinato

Inizialmente il sottovettore di destra è vuoto, quello di sinistra contiene N elementi.

L'operazione elementare è un confronto tra elementi successivi del vettore A[j] e A[j+1], scambio se A[j] > A[j+1].

Approccio

Paradigma incrementale:

- ad ogni passo si espande il sottovettore destro già ordinato inserendovi un elemento preso dal sottovettore sinistro ancora disordinato
- l'inserzione deve garantire che il sottovettore destro rimanga ordinato dopo l'inserimento (invarianza della proprietà di ordinamento)
- all'iterazione i si inserisce l'elemento massimo del sottovettore sinistro (A_l ... A_{r-i+l}) nell'estremo sinistro del sottovettore destro A[r-i+l]. Il sottovettore destro ordinato cresce di 1 posizione verso sinistra, dualmente quello di sinistra disordinato decresce di 1 posizione
- terminazione: tutti gli elementi sono stati inseriti ordinatamente, il sottovettore destro contiene N elementi, quello di sinistra è vuoto.

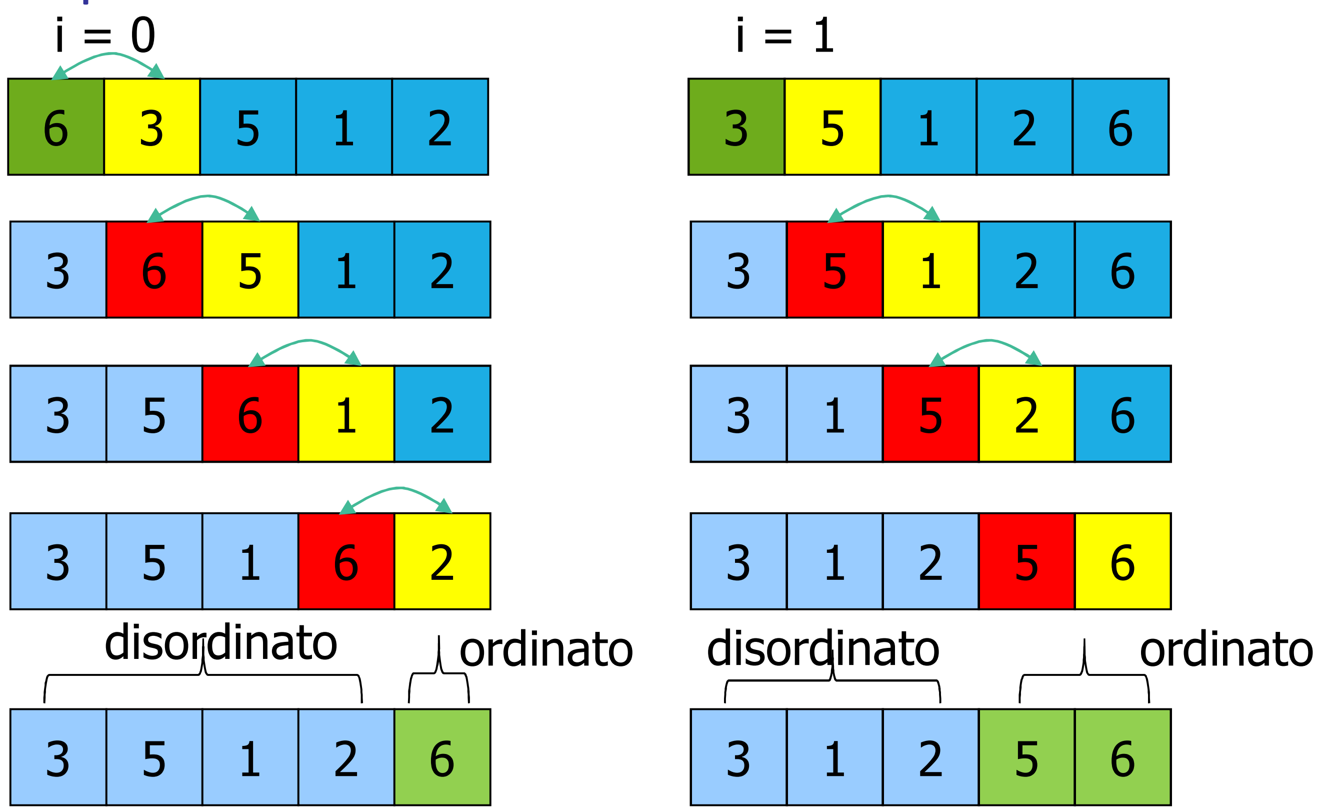
Passo i-esimo: identificazione del massimo

Al passo i-esimo si identifica il massimo del sottovettore sinistro

- si scandisce il sottovettore di sinistra confrontando le coppie di elementi A[j] e A[j+1] e scambiandole se A[j] > A[j+1]

Quando il ciclo termina, il massimo è «galleggiato» come una bolla («bubble») fino alla posizione corretta all'estremo sinistro del sottovettore ordinato destro.

Esempio

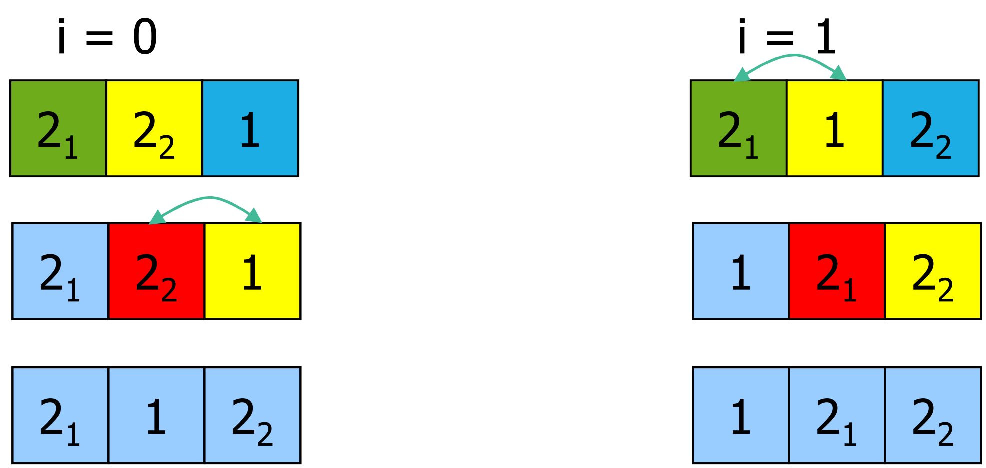

i = 0
6 3 5 1 2
3 6 5 1 2
3 5 6 1 2
3 5 1 6 2
disordinato
ordinato
3 5 1 2 6
i = 1
3 5 1 2 6
3 5 1 2 6
3 1 5 2 6
3 1 2 5 6
disordinato
ordinato
3 1 2 5 6

```
void BubbleSort(int A[], int N){
  int i, j, l=0, r=N-1;
  int temp;
  for (i = l; i < r; i++) {
    for (j = l; j < r - i +l; j++)
      if (A[j] > A[j+1]) {
        temp = A[j];
        A[j] = A[j+1];
        A[j+1] = temp;
      }
  }
  return;
}
```

i-l è la dimensione del sottovettore destro già ordinato

Caratteristiche del Bubble/Exchange sort

- in loco: oltre al vettore A si usa solo la variabile temp
- stabile: tra più chiavi duplicate quella più a destra prende la posizione più a destra e non viene mai «scavalcata» a destra da un’altra chiave uguale:



Analisi di complessità ad alto livello

Due cicli annidati:

- esterno: eseguito sempre N-1 volte
- interno: all'i-esima iterazione viene eseguito sempre N-1-i volte

$T(N) = (N-1) + (N-2) + \ldots\ 2 + 1$

$= \Sigma_{1 \leq i < N}\, i = N(N-1)/2$

progressione aritmetica finita di ragione 1 (Gauss, fine XVII sec.)

$T(N) = \Theta(N^2)$.

- scambi nel caso peggiore: $O(N^2)$: non sempre lo scambio avviene
- confronti nel caso peggiore: $\Theta(N^2)$: il confronto avviene sempre

Ottimizzazione

- si introduce un flag che indica se vi sono stati scambi, l'esecuzione prosegue solo se vi sono stati scambi:
  - il ciclo esterno viene eseguito al massimo N-1 volte
  - il ciclo interno all'i-esima iterazione viene eseguito sempre N-1-i volte
- confronti nel caso peggiore: $O(N^2)$: il confronto non sempre avviene
- si migliora il caso medio, ma non cambia la complessità di caso peggiore.

Esempio

flag = 1
i = 0 eseguo ciclo esterno

flag = 0

j = 0 eseguo ciclo interno	4	2	1	5	3	⇨	2	4	1	5	3	flag = 1
j = 1 eseguo ciclo interno	2	4	1	5	3	⇨	2	1	4	5	3	flag = 1
j = 2 eseguo ciclo interno	2	1	4	5	3	⇨	2	1	4	5	3	flag = 1
j = 3 eseguo ciclo interno	2	1	4	5	3	⇨	2	1	4	3	5	flag = 1

i = 1 eseguo ciclo esterno

flag = 0

j = 0 eseguo ciclo interno	2	1	4	3	5	⇨	1	2	4	3	5	flag = 1
j = 1 eseguo ciclo interno	1	2	4	3	5	⇨	1	2	4	3	5	flag = 1
j = 2 eseguo ciclo interno	1	2	4	3	5	⇨	1	2	3	4	5	flag = 1
j = 3 eseguo ciclo interno	1	2	3	4	5	⇨	1	2	3	4	5	flag = 1

i = 2 eseguo ciclo esterno

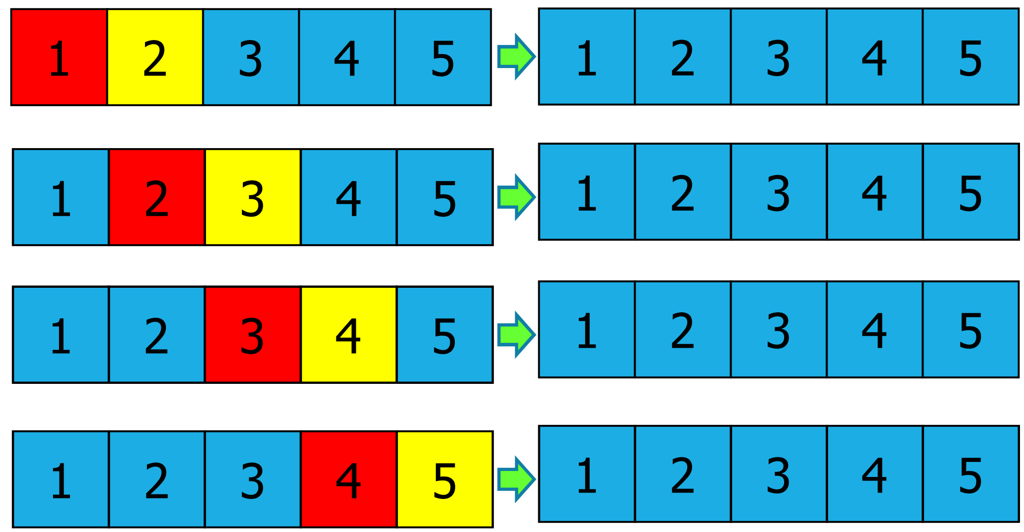
flag = 0

j = 0 eseguo ciclo interno

j = 1 eseguo ciclo interno

j = 2 eseguo ciclo interno

j = 3 eseguo ciclo interno



i = 3 ma flag = 0 in quanto non ci sono stati scambi, non eseguo ciclo esterno

```
void OptBubbleSort(int A[], int N) {
  int i, j, l=0, r=N-1, flag=1;
  int temp;

  for (i = l; i < r && flag==1; i++) {
    flag = 0;
    for (j = l; j < r - i + l; j++)
      if (A[j] > A[j+1]) {
        flag = 1;
        temp = A[j];
        A[j] = A[j+1];
        A[j+1] = temp;
      }
  }
  return;
}
```

Selection sort

Siano dati N interi memorizzati in un vettore A con indici compresi tra l=0 e r=N-1.

Concettualmente il vettore A è diviso in 2 sottovettori:

- di sinistra: già ordinato
- di destra: ancora disordinato

Inizialmente il sottovettore di sinistra è vuoto, quello di destra contiene N elementi.

Approccio

Paradigma incrementale:

- ad ogni passo si espande il sottovettore sinistro già ordinato inserendovi un elemento preso dal sottovettore destro ancora disordinato
- l’inserzione deve garantire che il sottovettore sinistro rimanga ordinato dopo l’inserimento (invarianza della proprietà di ordinamento). Questo è garantito identificando il minimo del sottovettore destro (A_i ... A_r) ed assegnandolo a A[i]. Il sottovettore sinistro ordinato cresce di 1 posizione verso destra, dualmente quello di destra disordinato decresce di 1 posizione
- terminazione: tutti gli elementi sono stati inseriti ordinatamente, il sottovettore sinistro contiene N elementi, quello di destra è vuoto.

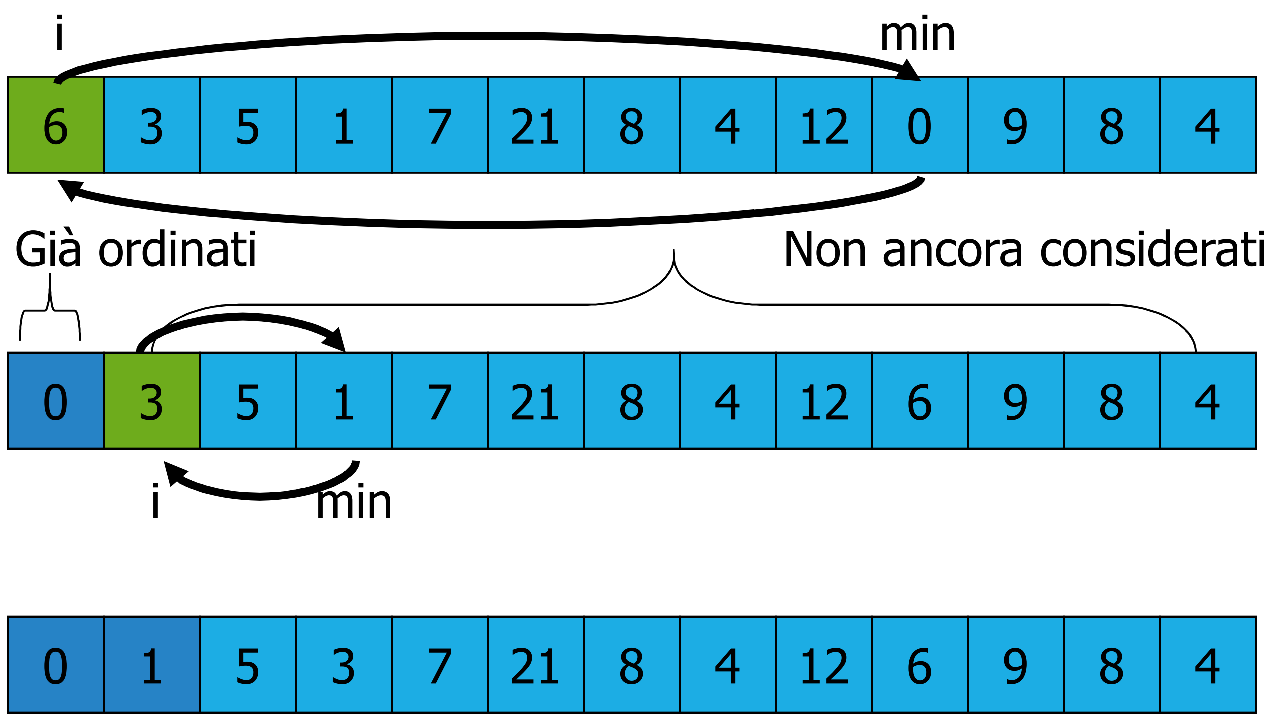
Passo i-esimo: identificazione del minimo

Al passo i-esimo si identifica il minimo del sottovettore destro:

- si scandisce il sottovettore di destra, ipotizzando il minimo in A[i] e aggiornando il minimo ad ogni confronto con gli elementi successivi

Quando il ciclo termina, è stato identificato il minimo tramite la sua posizione (indice in A) e lo si scambia con A[i].

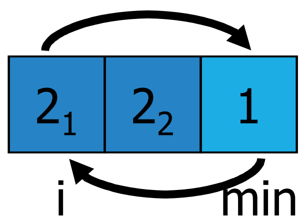
Esempio

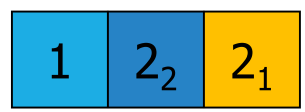

i
min
6 3 5 1 7 21 8 4 12 0 9 8 4
Già ordinati
Non ancora considerati
0 3 5 1 7 21 8 4 12 6 9 8 4
i
min
0 1 5 3 7 21 8 4 12 6 9 8 4

```
void SelectionSort(int A[], int N) {
  int i, j, l=0, r=N-1, min;
  int temp;
  for (i = l; i < r; i++) {
    min = i;
    for (j = i+1; j <= r; j++)
      if (A[j] < A[min])
          min = j;
    if (min != i) {
       temp = A[i];
      A[i] = A[min];
      A[min] = temp;
     }
  }
  return;
}
```

Caratteristiche del Selection sort

- in loco: oltre al vettore A si usa solo la variabile temp
- non stabile: uno scambio tra elementi «lontani» può far sì che un'occorrenza di una chiave duplicata si sposti a sinistra di un'occorrenza precedente della stessa chiave «scavalcandola»:





Analisi di complessità ad alto livello

Due cicli annidati:

- esterno: eseguito sempre N-1 volte
- interno: all'i-esima iterazione viene eseguito sempre N-1-i volte

$$T(N) = (N-1) + (N-2) + \dots 2 + 1$$
$$= \Sigma_{1 \leq i < N}\, i = N(N-1)/2$$

progressione aritmetica finita di ragione 1 (Gauss, fine XVII sec.)

$T(N) = \Theta(N^2)$.

- scambi nel caso peggiore: $O(N)$: al massimo può succedere di dover sempre scambiare con il minimo corrente, ma questo può avvenire al massimo N volte
- confronti nel caso peggiore: $\Theta(N^2)$: il confronto avviene sempre
- si tratta comunque di un algoritmo quadratico, in quanto la complessità è determinata dai confronti, non dagli scambi.

Shell sort (Shell, 1959)

Limite dell'Insertion sort: il confronto, quindi lo scambio, avviene solo tra elementi adiacenti.

Idea dello Shell sort:

- confrontare, quindi eventualmente scambiare, elementi a distanza h tra di loro
- definendo una sequenza decrescente di interi h che termina con 1, riconducendosi quindi all'Insertion sort all'ultimo passo.

Sequenze lineari

- Insieme finito di elementi consecutiin cui a ogni elemento è associato univocamente un indice

$$a_0, a_1, \dots , a_i, \dots, a_{n-1}$$

- Sulle coppie di elementi contigui è definita una relazione predecessore/successore:

$$a_{i+1} = succ(a_i) \qquad\qquad a_i = pred(a_{i+1})$$

- Memorizzazione e accesso:
  - vettore: dati contigui in memoria con accesso diretto:
    - dato l’indice i, si accede all’elemento a_i senza dover scorrere la sequenza lineare
    - il costo dell’accesso non dipende dalla posizione dell’elemento nella sequenza lineare, quindi è $O(1)$
  - lista: dati non contigui in memoria con accesso sequenziale, trattata nel Corso del II anno.

Sottovettori e sottosequenze

Data una sequenza lineare di N interi memorizzata in un vettore A

$$A = (a_0, a_1, \dots a_{N-1})$$

si definisce **sottosequenza** di A di lunghezza k ($k \leq N$) un qualsiasi n-upla Y di k elementi di A con indici crescenti non necessariamente contigui $i_0, i_1, \cdots, i_{k-1}$

si definisce **sottovettore** di A di lunghezza k ($k \leq N$) un qualsiasi n-upla Y di k elementi di A con indici crescenti e contigui $i_0, i_1, \cdots, i_{k-1}$.

Esempio

A	6	3	5	1	7	21	8	4	12	0	9	8	4
	0	1	2	3	4	5	6	7	8	9	10	11	12

A	6	3	5	1	**7**	**21**	**8**	**4**	12	0	9	8	4
	0	1	2	3	4	5	6	7	8	9	10	11	12

sottovettore di A di k=4 elementi con indici crescenti e contigui i_4, i_5, i_6, i_7

A	6	**3**	5	1	**7**	21	**8**	4	12	0	9	**8**	4
	0	1	2	3	4	5	6	7	8	9	10	11	12

sottosequenza di A di k=4 elementi con indici crescenti ma non contigui i_1, i_4, i_6, i_{11}

Le sottosequenze nello Shell sort

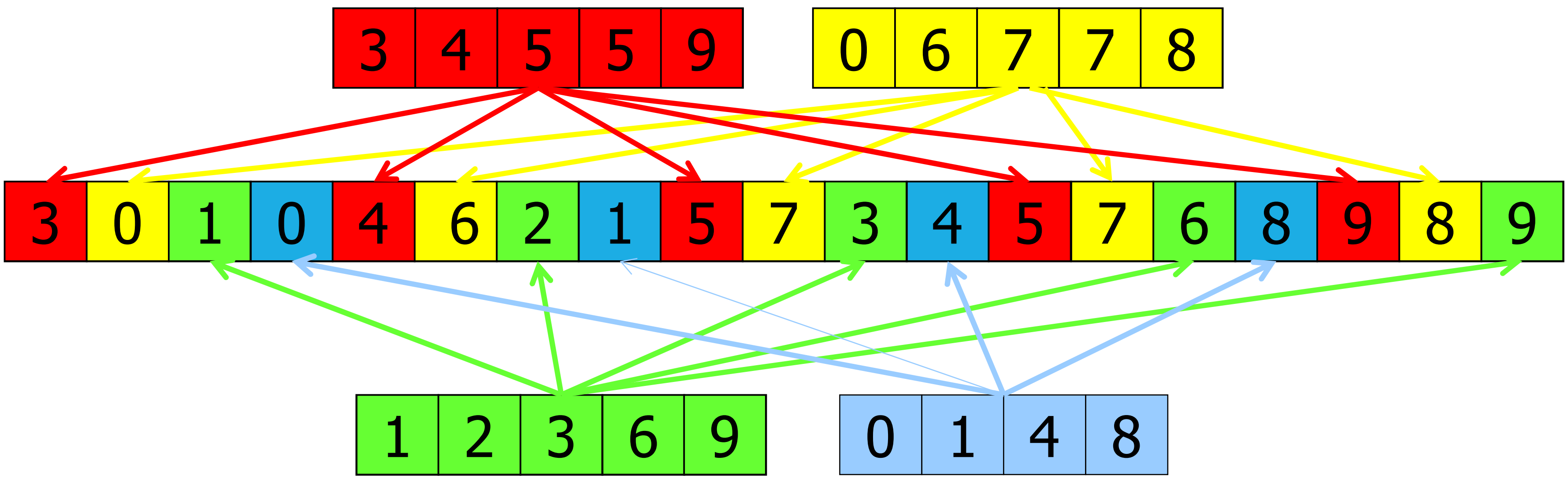
Nello Shell sort si considerano le sottosequenze formate dagli elementi del vettore i cui indici distano h.

Esempio: h=4

5	8	9	0	4	6	3	1	3	7	6	4	5	7	1	8	9	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Se le sottosequenze formate da elementi a distanza h sono ordinate, il vettore si dice h-ordinato.

Esempio: h=4



Per ognuna delle sottosequenze si applica l'insertion sort. Gli elementi della sottosequenza sono quelli a distanza h da quello corrente.

```
for i = l+h; i <= r; i++) {
  j = i; x = A[i];
  while (j>= l+h && x < A[j-h]) {
    A[j] = A[j-h];
    j -=h;
  }
  A[j] = x;
}
```

Esempio

sequenza h: 13, 4, 1

Passo 1: h=13

5	8	9	0	4	6	3	1	3	7	6	4	5	7	1	8	9	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

5	8	9	0	4	6	3	1	3	7	6	4	5	7	1	8	9	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

5	1	9	0	4	6	3	1	3	7	6	4	5	7	8	8	9	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

5	1	8	0	4	6	3	1	3	7	6	4	5	7	8	9	9	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

5	1	8	0	4	6	3	1	3	7	6	4	5	7	8	9	9	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

5	1	8	0	0	6	3	1	3	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

Passo 2: h=4

5	1	8	0	0	6	3	1	3	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	8	0	5	6	3	1	3	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Passo 2: h=4

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

0	1	8	0	3	6	3	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	8	1	5	7	6	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Passo 2: h=4

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

Passo 2: h=4

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

Passo 3: h=1

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	3	0	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	3	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	3	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	3	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

Passo 3: h=1

0	0	1	3	3	6	6	1	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	6	6	5	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	5	6	6	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	5	6	6	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	5	6	6	7	8	4	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	4	5	6	6	7	8	5	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

sequenza h: 13, 4, 1

Passo 3: h=1

0	0	1	1	3	3	4	5	5	6	6	7	8	7	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	4	5	5	6	6	7	7	8	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	4	5	5	6	6	7	7	8	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	4	5	5	6	6	7	7	8	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	4	5	5	6	6	7	7	8	8	9	9	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	0	1	1	3	3	4	4	5	5	6	6	7	7	8	8	9	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Sequenze

- Sequenza di Shell (1959) $h_i = 2^{i-1}$

$h_1 = 1$, $h_2 = 2$, $h_3 = 4$, $h_4 = 8$, $h_5 = 16$, …

- Sequenza di Hibbard (1963): $h_i = 2^i - 1$

$h_1 = 1$, $h_2 = 3$, $h_3 = 7$, $h_4 = 15$, $h_5 = 31$, …

- Sequenza di Pratt (1971):

1, 2, 3, 4, 6, 8, 9, 12, 16, …, $2^p 3^q$, …

- Sequenza di Knuth (1973): con h_0=0 inizialmente e $h_i = 3h_{i-1}+1$

$h_1 = 1$, $h_2 = 4$, $h_3 = 13$, $h_4 = 40$, $h_5 = 121$, …

- Sequenza di Sedgewick (1986):

1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, …

```
void ShellSort(int A[], int N) {
  int i, j, x, l=0, r=N-1, h=1;
  while (h < N/3)
    h = 3*h+1;
  while(h >= 1) {
    for (i = l + h; i <= r; i++) {
      j = i;
      x = A[i];
      while(j >= l + h  && x < A[j-h]) {
        A[j] = A[j-h];
        j -=h;
      }
     A[j] = x;
    }
    h = h/3;
  }
}
```

sequenza di Knuth

Caratteristiche dello Shell sort

- in loco: oltre al vettore A si usa solo la variabile x
- non stabile: uno scambio tra elementi «lontani» può far sì che un’occorrenza di una chiave duplicata si sposti a sinistra di un’occorrenza precedente della stessa chiave «scavalcandola»:

2_1	2_2	2_3	2_4	2_5	0

2_1	2_2	2_3	2_4	2_5	0

2_1	0	2_3	2_4	2_5	2_2

0	2_1	2_3	2_4	2_5	2_2

Analisi di complessità ad alto livello

- con la sequenza di Shell 1 2 4 8 16 … T(N) = $O(N^2)$
- con la sequenza di Hibbard 1 3 7 15 31 … T(N) = $O(N^{3/2})$
- con la sequenza di Pratt 1 2 3 4 6 8 9 12 … T(N) = $O(N\log^2 N)$
- con la sequenza di Knuth: 1 4 13 40 121 … T(N) = $O(N^{3/2})$
- con la sequenza di Sedgewick 1, 5, 19, 41, 109, 209, … T(N) = $O(N^{4/3})$