Introduzione all'Algoritmica per i Licei (C++).

4 – Algoritmi di ordinamento elementari.

versione 8 febbraio 2015

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

Il problema dell'ordinamento

Data una sequenza (ad es. array o vector) di elementi, ordinarla rispetto ad una relazione d'ordine definita sugli elementi (di solito su un particolare campo o componente di tali elementi);
ad es. ordinare un insieme di oggetti di una classe Studente rispetto al numero di matricola (costituente la chiave identificativa dell'oggetto), oppure rispetto al cognome, ecc.

Gli algoritmi che risolvono tale problema, cioè gli algoritmi di ordinamento, sono classificabili secondo diversi criteri, quali la semplicità di realizzazione, l'efficienza, oppure il fatto che godano o no di certe proprietà interessanti.

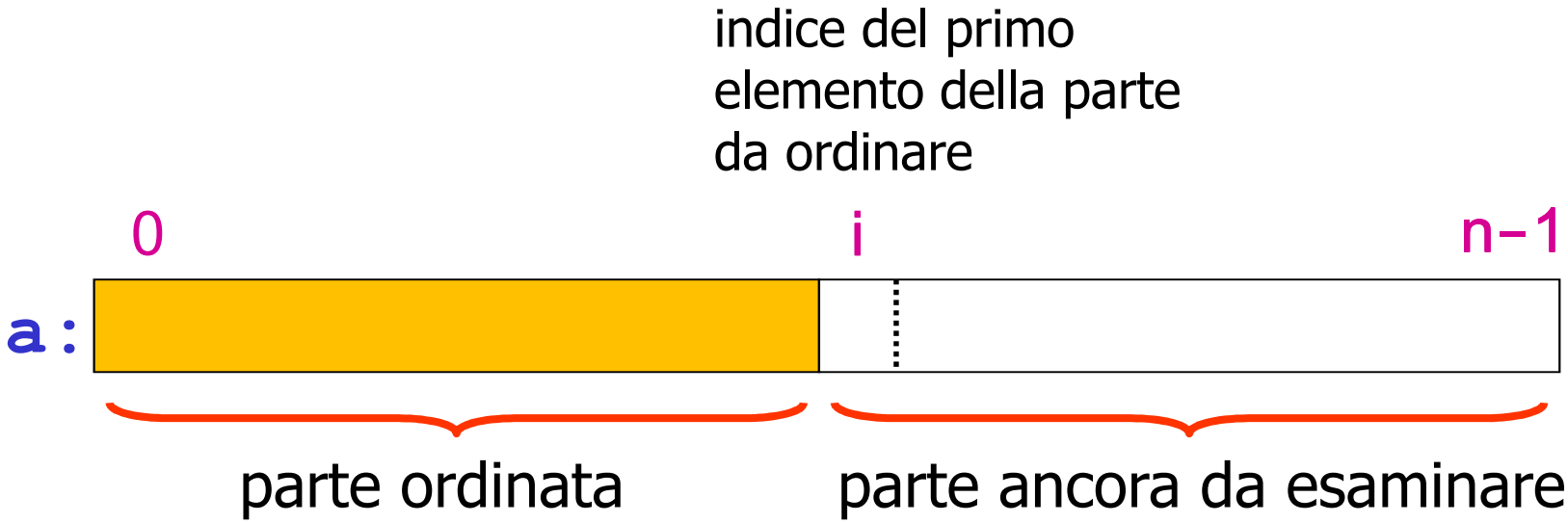
Algoritmo di ordinamento per selezione
o selection sort
o ordinamento per estrazione successiva del minimo

Descrizione informale.

Si cerca il minimo dell'array e lo si mette al primo posto, mettendo il primo al posto del minimo; poi si cerca il minimo nella parte di array dal secondo elemento alla fine, e lo si mette al secondo posto, e così via.

Qual è la situazione al passo generico?

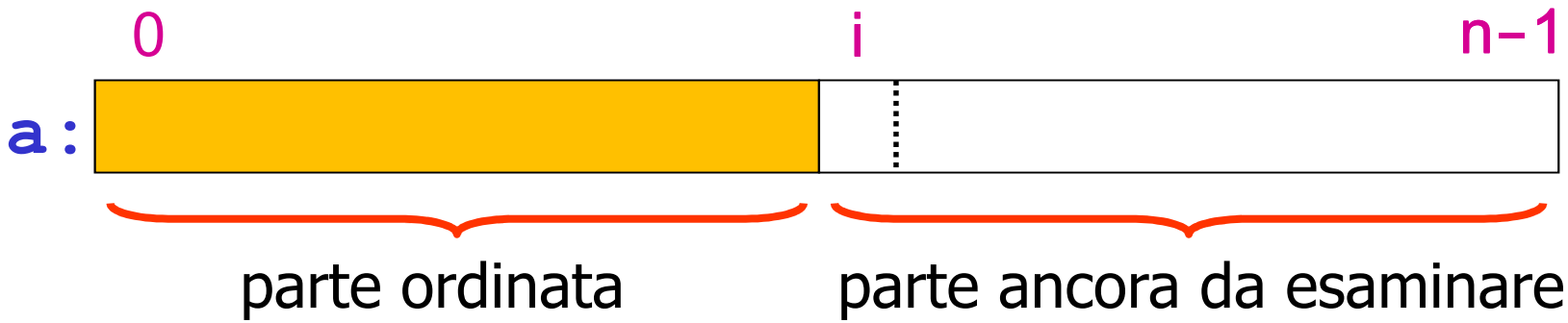
array a, di lunghezza n



Ogni elemento della parte ordinata è minore (o uguale) di tutti gli elementi della parte ancora da esaminare.

Ciascun elemento della parte ordinata è quindi già al suo posto definitivo nell'array.

Cosa bisogna fare nel corpo del ciclo principale?



Cercare il minimo nella parte a[i .. n-1] e scambiarlo con a[i]:

```
iMin = indice dell'elemento minimo in a[i .. n-1];
scambia a[i] con a[iMin];
```

Per quali valori di `i` devono essere eseguite tali istruzioni?

Da i = 0 (al primo passo si cerca il minimo di tutto l'array) fino a i = ... ?

Non c'è bisogno di controllare l'ultimo elemento, di indice n-1, poiché esso risulta automaticamente al posto giusto: essendosi estratto ogni volta il minimo degli elementi rimasti da esaminare, l'ultimo elemento rimasto è il massimo. Quindi:

```
for(int i = 0; i < n-1; i++) {
   iMin = indice dell'elemento minimo in a[i .. n-1];
     scambia a[i] con a[iMin];
}
```

Lo schema della procedura

```
template<typename E> void ssort(E a[], int n) {
  for(int i = 0; i < n-1; i++) {
    int iMin = indice dell'elemento minimo in a[i .. n-1];



    scambia a[i] con a[iMin];
  }
}
```

Troviamo l'indice del minimo in a[i .. n-1]

```
template<typename E> void ssort(E a[], int n) {
  for(int i = 0; i < n-1; i++) {
    int iMin = i;
    for(int j = i+1; j < n; j++) {
      if(a[j] < a[iMin]) iMin = j;
    }
    scambia a[i] con a[iMin];
  }
}
```

Usiamo una funzione ausiliaria "scambia" ("swap")

```
template<typename E> void ssort(E a[], int n) {
  for(int i = 0; i < n-1; i++) {
    int iMin = i;
    for(int j = i+1; j < n; j++) {
      if(a[j] < a[iMin]) iMin = j;
    }
    swap(a, i, iMin);
  }
}
```

Definiamo la funzione ausiliaria

```
template<typename E> void swap(E a[],int i,int j) {
  E temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
```

Tempo di calcolo.

Quanti passi fa l'algoritmo per ordinare un array di n elementi?

- cerca il minimo dell'array, percorrendolo tutto: n passi
- cerca il minimo dal secondo in poi: n-1 passi
- cerca il minimo dal terzo in poi: n-2 passi
- …

Quanti passi fa in tutto?

$$n + (n-1) + (n-2) + (n-3) + \ldots + 2$$

Cioè: $$[n + (n-1) + (n-2) + (n-3) + \ldots + 2 + 1] - 1$$

è la somma dei numeri interi da 1 a n (diminuita di uno).

Ma è la formula scoperta da Gauss bambino!

$n(n+1)/2$ che è uguale a $(n^2 + n)/2$

Il numero di passi, e quindi il tempo di calcolo, cresce come il quadrato del numero degli elementi.

Crescita quadratica del tempo

Se n raddoppia diventando 2n, il quadrato diventa $(2n)^2 = 4n^2$, cioè quadruplica.

Se n si triplica diventando 3n, il quadrato diventa $(3n)^2 = 9n^2$, cioè si moltiplica per 9. Ecc.

Quindi se il numero di elementi dell'array raddoppia, il tempo necessario per ordinare l'array quadruplica;

se il numero di elementi si triplica, il tempo necessario per ordinare l'array si moltiplica per 9;

se il numero di elementi si decuplica, il tempo necessario si moltiplica per 100; ecc.

Inoltre il numero di passi, per un array di una data lunghezza, è sempre lo stesso, indipendente dalla disposizione degli elementi.

Ad esempio è lo stesso anche se l'array è già ordinato, o quasi ordinato.

Algoritmo di ordinamento per inserimento o insertion sort (isort)

Descrizione informale.

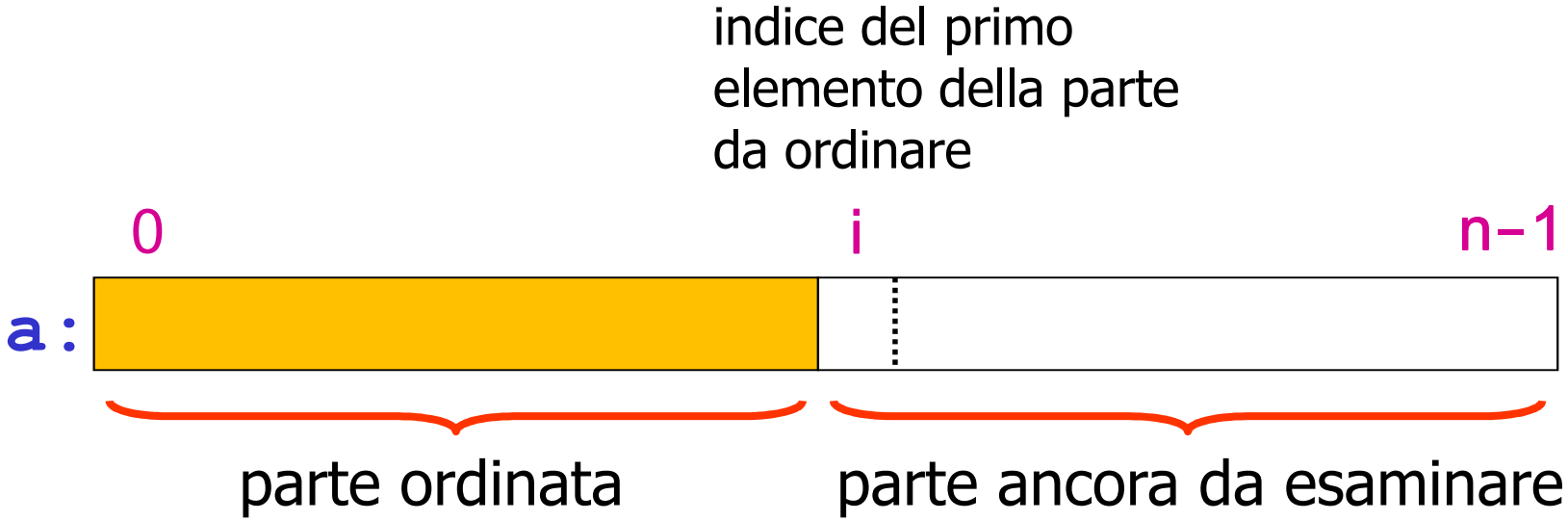
Si prende via via l'elemento successivo dell'array e lo si inserisce, rispettando l'ordine, nella parte già ordinata, come quando, ricevute il proprio mazzetto di carte, se ne prende in mano una dopo l'altra.

Attenzione: "ordinamento per inserimento" è il nome di un *algoritmo di ordinamento*, cioè di un algoritmo che prende come argomento una sequenza esistente e la ordina;

NON è una procedura che inserisce in modo ordinato degli elementi in una sequenza vuota, pur basandosi sullo stesso principio di essa.

Qual è la situazione al passo generico?

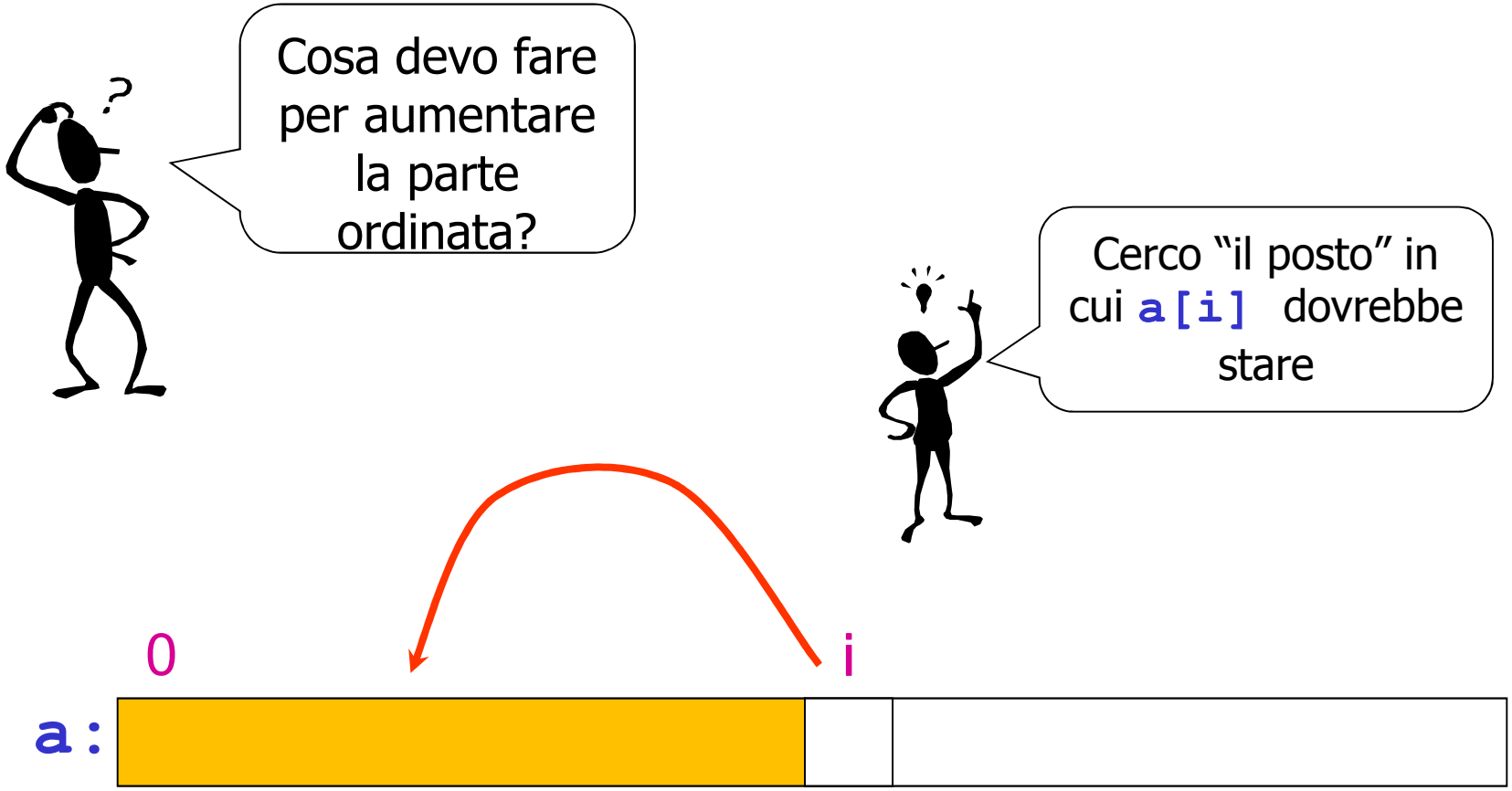
array a, di lunghezza n



A differenza del selection sort, degli elementi ancora da esaminare non sappiamo nulla!

Che cosa bisogna fare al passo generico?

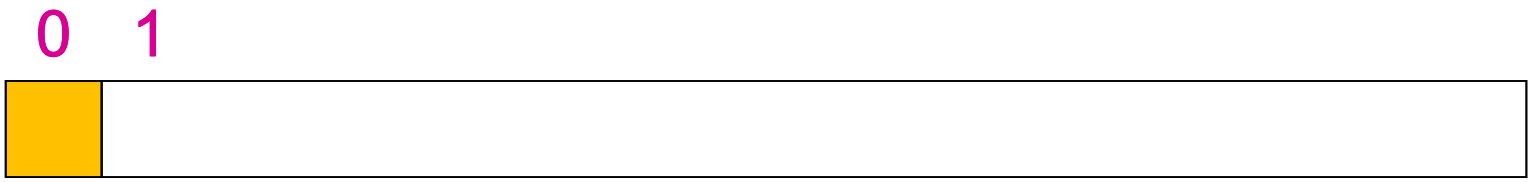
Il passo



Schema dell'algoritmo

```
for(int i = ?; i < ?; i++) {
```
prendi l'elemento **a[i]**
e inseriscilo al posto giusto in a[0 .. i-1],
spostando gli elementi successivi per fargli spazio.

Da quale valore di i si inizia?
Per poter effettuare l'inserimento ci deve essere una porzione iniziale già ordinata. ma la porzione costituita dal solo primo elemento a[0] è banalmente ordinata!
Si inizia quindi dal secondo elemento, a[1].



Schema dell'algoritmo

```
for(int i = ?; i < ?; i++) {
```
prendi l'elemento **a[i]**
e inseriscilo al posto giusto in a[0 .. i-1],
spostando gli elementi successivi per fargli spazio.

Con quale valore di i si termina?

Quando arriviamo all'ultimo elemento, a differenza del ssort non sappiamo nulla di tale elemento, che va quindi inserito al posto giusto come tutti gli altri.

Differenza fra il ssort e l'isort.

Si noti la differenza nella forma del ciclo for nei due algoritmi.

ssort (ordinamento per estrazione successiva del minimo):

```
for(int i = 0; i < n-1; i++)
```

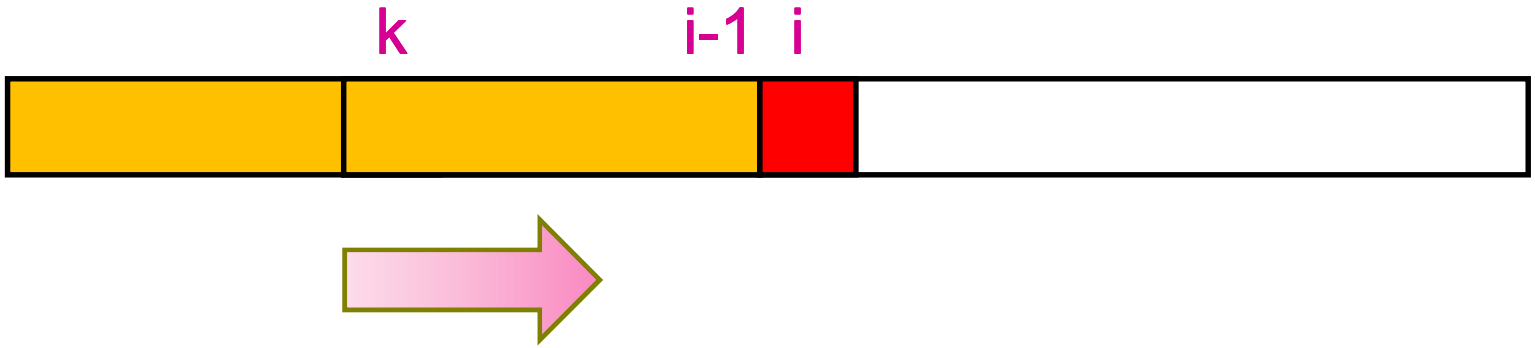
isort (ordinamento per inserimento):

```
for(int i = 1; i < n; i++)
```

Come inserire un elemento al posto giusto

Il passo da fare

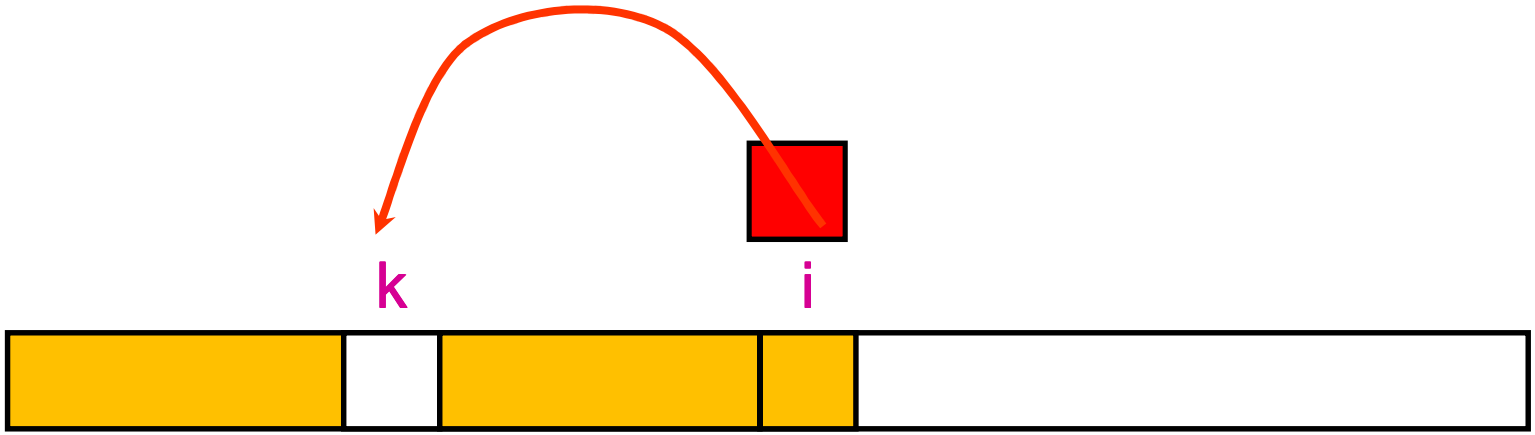
Vogliamo inserire l'elemento a[i] al posto giusto k (che dobbiamo scoprire qual è) in a[0 .. i-1]:



Ordinamento per inserimento

Il passo da fare

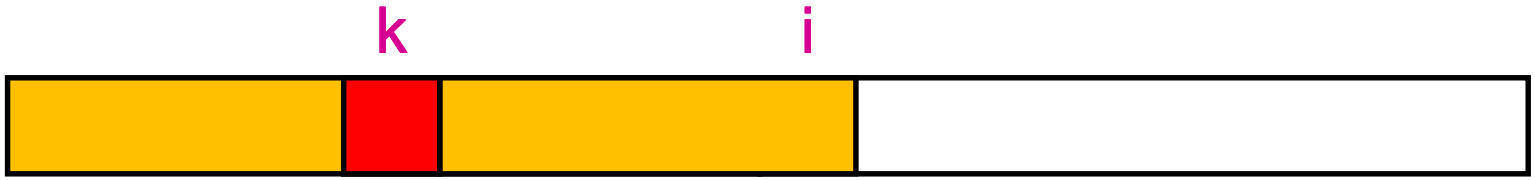
Vogliamo inserire l'elemento a[i] al posto giusto k (che dobbiamo scoprire qual è) in a[0 .. i-1]:



Ordinamento per inserimento

Il passo da fare

Vogliamo inserire l'elemento a[i] al posto giusto k (che dobbiamo scoprire qual è) in a[0 .. i-1]:



Inserimento di a[i] in a[0 .. i-1]: versione 1.

Situazione iniziale: il segmento a[0 .. i-1] è ordinato

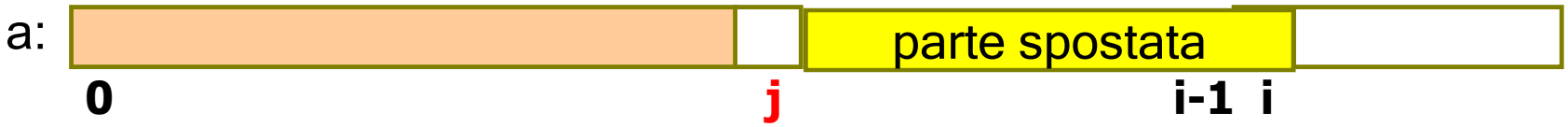
el = a[i] è l'elemento da inserire

A partire da a[i-1] compreso, sposto a destra di un posto tutti gli elementi di a[0..i-1]:

```
for(int j = i; j > 0; j--) {

  a[j] = a[j-1];
}
```

Inserimento di a[i] in a[0 .. i-1]: versione 1.



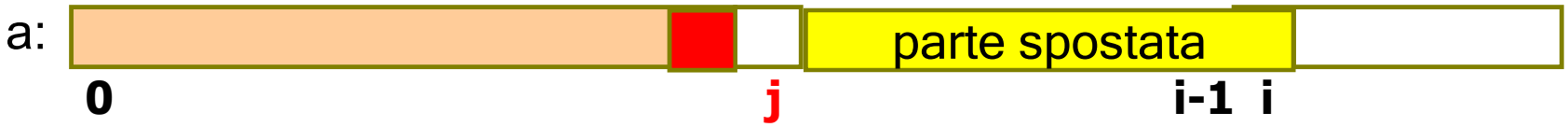
el = a[i] è l'elemento da inserire

A partire da a[i-1] compreso, sposto a destra di un posto tutti gli elementi di a[0..i-1]:

```
for(int j = i; j > 0; j--) {

  a[j] = a[j-1];
}
```

Inserimento di a[i] in a[0 .. i-1]: versione 1.



el = a[i] è l'elemento da inserire

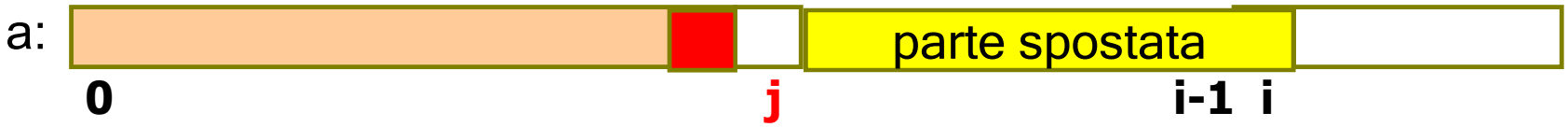
A partire da a[i-1] compreso, sposto a destra di un posto tutti gli elementi di a[0..i-1]:

```
for(int j = i; j > 0; j--) {
```

se però el è ≥ dell'elemento da spostare, mi fermo

```
   a[j] = a[j-1];
}
```

Inserimento di a[i] in a[0 .. i-1]: versione 1.



el = a[i] è l'elemento da inserire

A partire da a[i-1] compreso, sposto a destra di un posto tutti gli elementi di a[0..i-1]:

```
for(int j = i; j > 0; j--) {
   if(el >= a[j-1]) break; // el va inserito in a[j]
   a[j] = a[j-1];
}
```

Inserimento di a[i] in a[0 .. i-1]: versione 1.

Se esco dal for regolarmente, ho spostato tutti gli elementi, e `el` va inserito in `a[0]`; ma in tal caso `j` è uguale a 0, quindi `j` è in ogni caso l'indice a cui inserire el:

```
for(int j = i; j > 0; j--) {
   if(el >= a[j-1]) break; // el va inserito in a[j]
   a[j] = a[j-1];
}
a[j] = el;
```

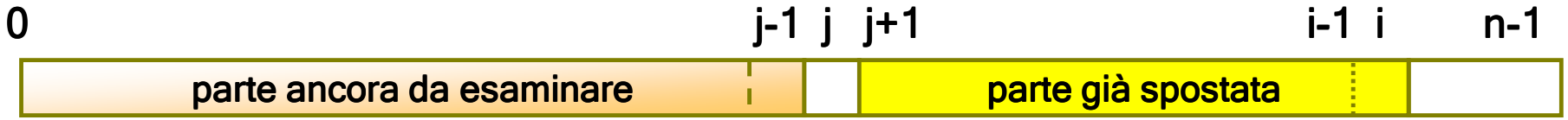
La procedura completa (versione 1)

```
template<typename E> void isort(E a[], int n) {
  for(int i = 1; i < n; i++) {
    E el = a[i];
    int j;
    for(j = i; j>0; j--) {
      if(el >= a[j-1]) break;
      a[j] = a[j-1];
    }
    a[j] = el;
  }
}
```

Versione considerata da molti non elegante e difficile da leggere: bisogna ricordare che il break fa uscire solo dal for interno.

Inserimento di x in a[0 .. i-1]: versione 2.

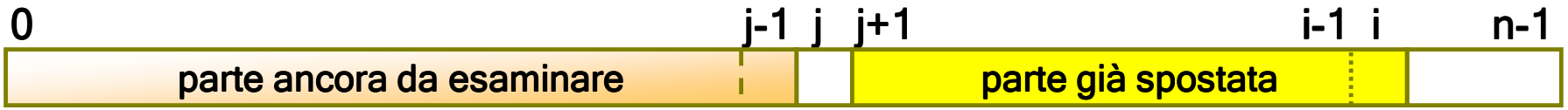
Situazione al passo generico.



L'elemento da inserire el è minore di tutti gli elementi "gialli" già spostati, e a[j] è il posto libero (che si è creato spostando a destra di uno tutti gli elementi successivi)

Inserisci x in a[0 .. i-1]: versione 2.

Situazione al passo generico.

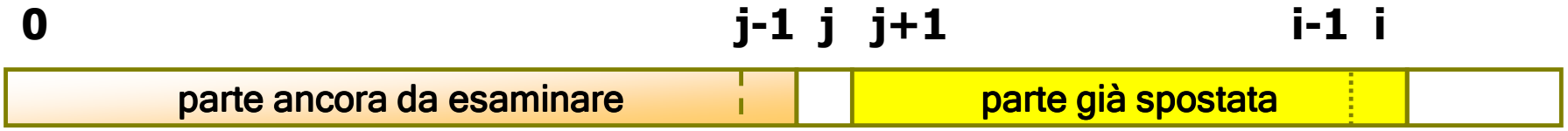


L'elemento da inserire el è minore di tutti gli elementi "gialli" già spostati, e a[j] è il posto libero (che si è creato spostando a destra di uno tutti gli elementi successivi)

Corpo del ciclo e condizione di uscita dal ciclo.

- se `j == 0` (cioè il posto libero è il primo) oppure `el` ≧ `a[j-1]` allora `el` deve essere inserito nel posto libero.
- altrimenti (cioè `j > 0` e `el < a[j-1]`) bisogna spostare a sinistra il posto vuoto, e quindi spostare `a[j-1]` a destra.

Inserisci x in a[0 .. i-1]: versione 2.

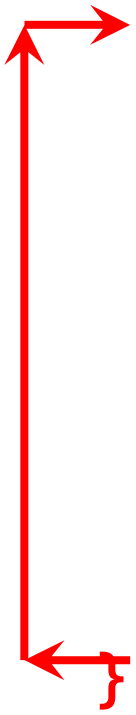


Si esce dal ciclo per j = 0 oppure el >= a[j-1];
quindi, anche ricordando de Morgan,
si continua per j > 0 and el < a[j-1].
In conclusione:

```
while(j > 0 && el < a[j-1]) {
  a[j] = a[j-1];
  j--;
}
a[j] = el;
```

La procedura completa

```
template<typename E> void isort(E a[], int n) {
  for(int i = 1; i < n; i++) {
    E el = a[i];
    int j = i;
    while(j > 0 && el < a[j-1]) {
      a[j] = a[j-1];
      j--;
    }
    a[j] = el;
  }
```



Come si sarà notato, le due "versioni" dell'inserimento sono del tutto equivalenti, la differenza è puramente sintattica.

Tuttavia il ragionamento con cui ci si arriva è nelle due versioni lievemente diverso: si scelga quella che risulta più naturale.

La versione con il while e senza il break è di solito considerata più elegante.

Tempo di calcolo

L'algoritmo è costituito da un ciclo for di n passi, senza uscite forzate. Tuttavia all'interno di tale ciclo viene eseguito un ciclo interno di inserimento, il cui tempo di calcolo non è costante, ma in generale sarà dipendente dal valore di a[i].

In media, tale ciclo interno troverà il posto giusto all'incirca a metà della parte già ordinata, e farà quindi un numero di passi circa uguale a i/2 (divisione intera),
cioè 1/2 la prima volta, 2/2 la seconda, 3/2 la terza, ...;
approssimativamente, il numero totale di passi sarà allora:

$1/2\ (1 + 2 + 3 + \ldots + n) = 1/2\ (n(n+1)/2) = (n^2 + n)/4$

Il tempo di calcolo cresce in modo quadratico rispetto alla lunghezza dell'array da ordinare, cioè è circa proporzionale al quadrato di tale lunghezza, come per il ssort.

Tempo di calcolo (continua)

caso peggiore: l'array di partenza è ordinato inversamente; allora ogni esecuzione dell'algoritmo *inserisci* effettua k passi, con k successivamente uguale a 1, 2,..,n (verificare simulando a mano l'esecuzione);
il numero totale di passi è quindi:

$1+2+3+ \ldots + n = n(n+1)/2 = (n^2 + n)/2$

quadratico

caso migliore: l'array di partenza è già ordinato o ha pochi elementi "fuori posto"; allora ogni inserimento, inserendo un elemento che è maggiore o uguale dei precedenti, fa un solo passo (eccetto eventualmente i pochi elementi fuori posto).
Il numero totale di passi è quindi

1+1+1+ ... (n volte) (+ i passi per i pochi "fuori posto")

cioè circa proporzionale a n (o lineare in n).

Uso dell'algoritmo isort.

L'algoritmo di "ordinamento per inserimento" è quindi da scegliersi nei casi in cui, per qualche motivo, si sa a priori che la sequenza da ordinare è già quasi ordinata, cioè contiene un numero piccolo di elementi fuori posto.

Esercizi

1) Scrivi una versione dell'ordinamento per selezione la quale operi attraverso l'estrazione successiva del massimo invece che del minimo.

2) Scrivi un programma che misura i tempi di calcolo per i vari algoritmi di ordinamento per lunghezze crescenti di array.

3) Calcola la lunghezza dell'array per il quale, sul tuo computer, l'algoritmo di ordinamento per selezione ci metterebbe circa 20 minuti, e controlla se la tua previsione è corretta (facendo una passeggiata nel frattempo).