Informatica Generale
Andrea Corradini

14 - Algoritmi: ordinamento per inserimento e ricorsione

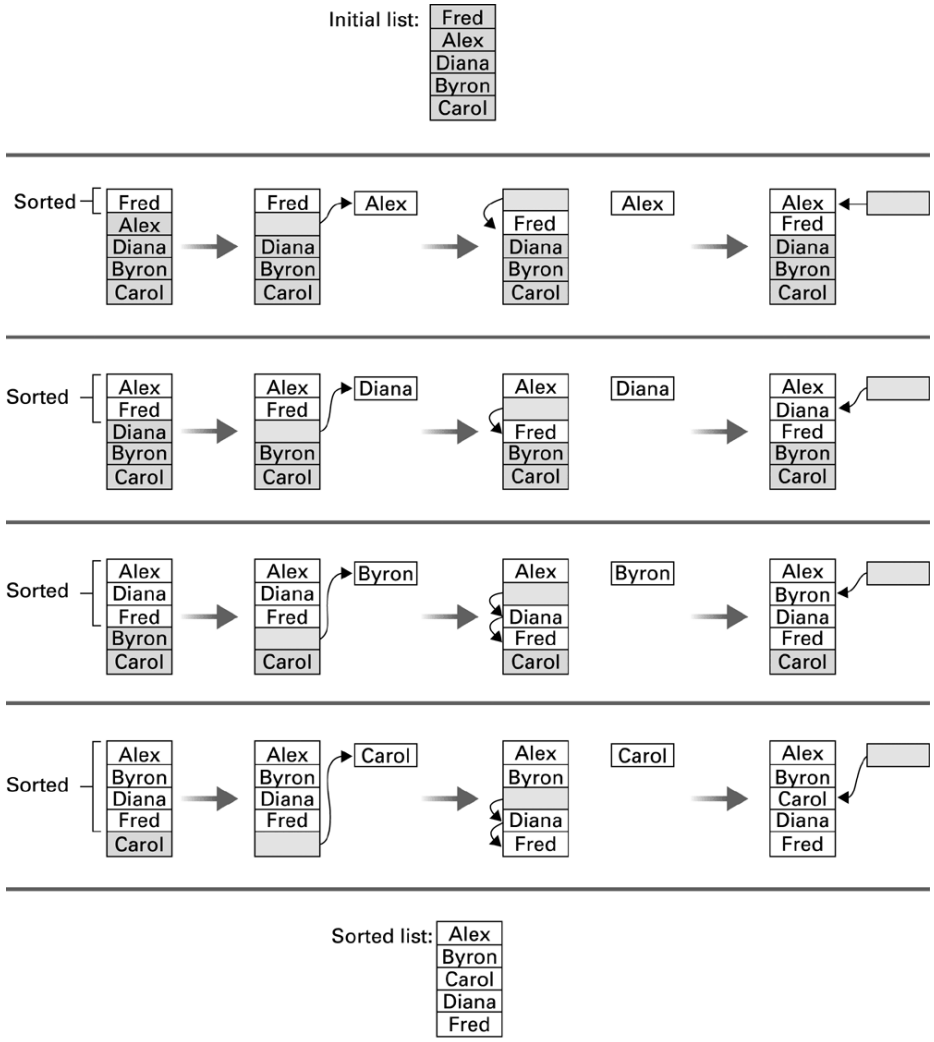
Sommario

- Un algoritmo iterativo: l'ordinamento per inserimento (***insertion sort***)
- La ricorsione: i numeri triangolari
- Il fattoriale
- Come funziona la ricorsione?

I due problemi fondamentali: ricerca e ordinamento

- **Ordinamento:** data una sequenza S di valori confrontabili tra loro, riorganizzare S in modo che contenga gli stessi valori, ma in ordine crescente
  - **Ordinamento per inserimento (insertion sort)**
  - Bubble sort, Quick sort, Merge sort
- **Ricerca:** dato un valore X e una sequenza di valori S, dire se X compare in S.
  - Ricerca lineare
  - Ricerca binaria

Esempio: come ordinare alfabeticamente la lista Fred, Alex, Diana, Byron, and Carol



L'algoritmo di *Insertion sort* in pseudocodice

- **Input**: una lista di valori, confrontabili tra loro
- **Output**: la stessa lista, ordinata in modo crescente
- **Assunzione**: non si crea una nuova lista, ma si modifica quella di input

```
procedure Sort (Lista)
   N ← 2;
while (il valore di N non supera la lunghezza di Lista) do
(  Seleziona l'elemento N-esimo di Lista come pivot;//perno
   Sposta il pivot in una posizione temporanea
      lasciando uno spazio vuoto in Lista;
   while (c'è un nome sopra lo spazio vuoto e quel
         nome è maggiore del pivot) do
   (  sposta nello spazio vuoto il nome nella posizione
      precedente lasciando un vuoto al suo posto
   )
   Sposta il pivot nello spazio vuoto in Lista
   N ← N+1;
)
```

L'algoritmo di *Insertion sort* in pseudocodice: versione finale

- Assumiamo che la **Lista** sia $S = s_1, s_2, ..., s_L$
- La lunghezza della lista è **L**

```
procedure Sort (Lista)
   N ← 2;
while (N <= L) do
(    pivot ← s_N;  // perno
     k ← N;   // indice della posizione vuota
     while  (k > 0 AND s_{k-1} > pivot) do
     (  s_k ← s_{k-1}; // copio nella posizione vuota
        k ← k-1; // sposto la posizione vuota
      )
      s_k ← pivot;
      N ← N+1;
)
```

La ricorsione: un semplice esempio

La **ricorsione** è un metodo di risoluzione di problemi molto potente: sfrutta l'idea di suddividere un problema da risolvere in sottoproblemi simili a quello originale, ma più semplici.

Esempio: Vogliamo calcolare l'area di una forma triangolare di dimensione **n** (vedi sotto). Ogni quadrato **[]** ha area unitaria (vale **1**). Il valore dell'area corrispondente viene a volte chiamato ***numero triangolare n-esimo***.
Per esempio, per **n=4** il triangolo è fatto come segue:

```
[]
[][]
[][][]
[][][][]
```

Osservando questo schema possiamo affermare che **il quarto numero triangolare è 10**.

Definizione ricorsiva di numeri triangolari

Scriviamo una procedura **`NumTriangolare(N)`** che restituisca come risultato il numero triangolare N-esimo. Ragioniamo per passi.
1) Se l'ampiezza del triangolo è 1, il triangolo consiste di un unico quadrato **[]** e la sua area è uguale a 1.
2) Per il caso più generale, consideriamo la figura sulla destra:
Supponiamo di conoscere l'area del triangolo più piccolo, formato dalle prime tre righe: il *terzo numero triangolare*.
In questo caso possiamo calcolare facilmente l'area del triangolo grande: il *quarto numero triangolare*:

```
[]
[][]
[][][]
[][][][]
```

```
risultato ← N + numero_triangolare_di_N-1;
```

Ma allora possiamo sfruttare la procedura, scrivendo: .

```
risultato ← N + NumTriangolare(n-1);
```

Calcolo dei numeri triangolari

```
procedure NumTriangolare(N)
   if (N = 1)
   then risultato ← 1;
   else risultato ← N + NumTriangolare(N-1);
   (Restituisci il valore di risultato)
```

Definizione di algoritmi ricorsivi

Un **algoritmo ricorsivo** per la risoluzione di un problema deve essere definito nel modo seguente:

- prima si definisce come risolvere dei problemi analoghi a quello di partenza, ma che hanno "dimensione piccola" e possono essere risolti in maniera estremamente semplice (detti **casi base**);
- poi si definisce come ottenere la soluzione del problema di partenza combinando la soluzione di uno o più problemi analoghi, ma di "dimensione inferiore".
- **Attenzione:** bisogna garantire che si raggiunga sempre un caso base, per assicurare la terminazione.

Analisi della procedura NumTriangolare

- La procedura **NumTriangolare** termina sempre? Cosa succede se la invoco con parametro **N=0** o con un numero negativo?

- La versione che segue è "più robusta":

```
procedure NumTriangolare(N)
   if (N <= 0)
   then risultato ← 0;    // caso anomalo
   else if (N = 1)        // caso base
      then risultato ← 1;
      else risultato ← N + NumTriangolare(N-1);
   return risultato; // Restituisci il valore di
     risultato)
```

Definizione ricorsiva di “fattoriale”

- La ricorsione non era necessaria per calcolare l'N-esimo numero triangolare: si poteva usare la nota formula di Gauss:

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

- Una funzione simile, ma per la quale non esiste una formula per calcolarla direttamente: la funzione fattoriale.

$$n! = n * (n - 1) * \ldots * 3 * 2 * 1$$

- Per convenzione **0! = 1**. Inoltre, il fattoriale non è definito per i numeri negativi. Usiamo la seguente definizione induttiva di fattoriale:

```
0! = 1                  // caso base
n! = n * (n-1)!         // se  n > 0
```

Procedura ricorsiva per fattoriale

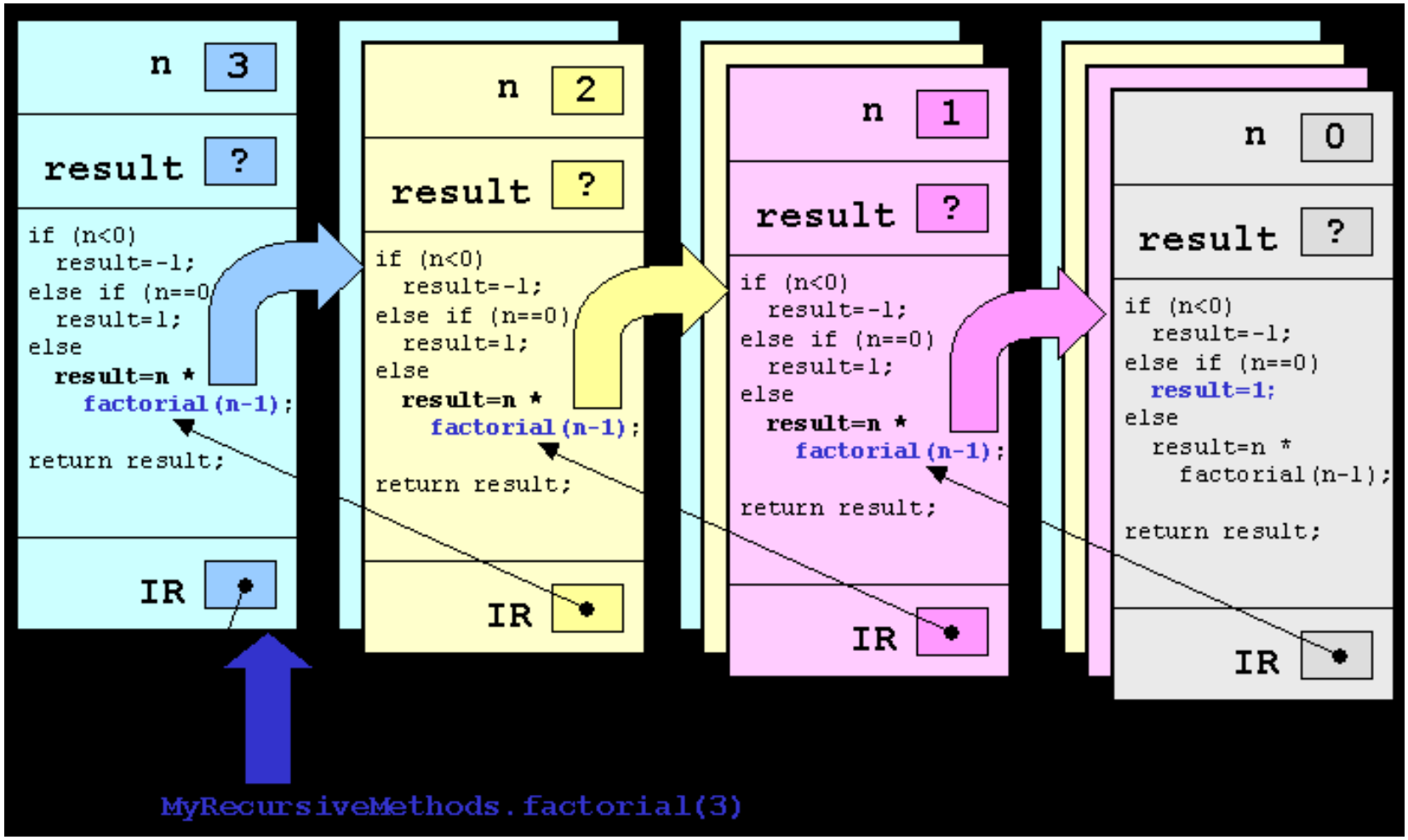
```
procedure factorial(n)
   if (n < 0)
   then result ← -1;    // caso anomalo
   else if (n = 0)         // caso base
      then result ← 1;
      else result ← n * factorial(n-1);
   return result;
```

- Cosa succede se invochiamo la procedura con parametro 3?

```
      fact ← factorial(3);
```

Vediamolo...

Come funziona la ricorsione?



Come funziona la ricorsione?