Fondamenti di Informatica L-A

Esercitazione 7

La Ricorsione

Paolo Torroni

Università degli Studi di Bologna
Laurea in Ingegneria Elettronica e delle Telecomunicazioni

Anno Accademico 2007/2008

△ 1. Versione ricorsiva del *Naïve Sort* ▽

- La versione ricorsiva del **naïve sort** (visto nel corso dell'Esercitazione 6), per un vettore di N elementi, è il seguente:

```
/* naiveSort( V, N ) */
```

- se il vettore contiene **almeno due elementi** ($N \geq 2$):
  - **cerca il massimo** elemento nel vettore;
  - **scambia** l'elemento trovato con l'ultimo elemento del vettore;
  - esegui l'**ordinamento** dei primi $N - 1$ elementi:

    `naiveSort( V, N-1 ).`

- Si chiede di:
  1. Implementare in C questo algoritmo per vettori di interi.
  2. Implementare (e usare) una funzione `countN()` per contare il **numero di confronti effettuati tra due elementi del vettore** da `naiveSort()`.

△ 2. Un Altro Algoritmo di Ordinamento (*Merge Sort*) ▽

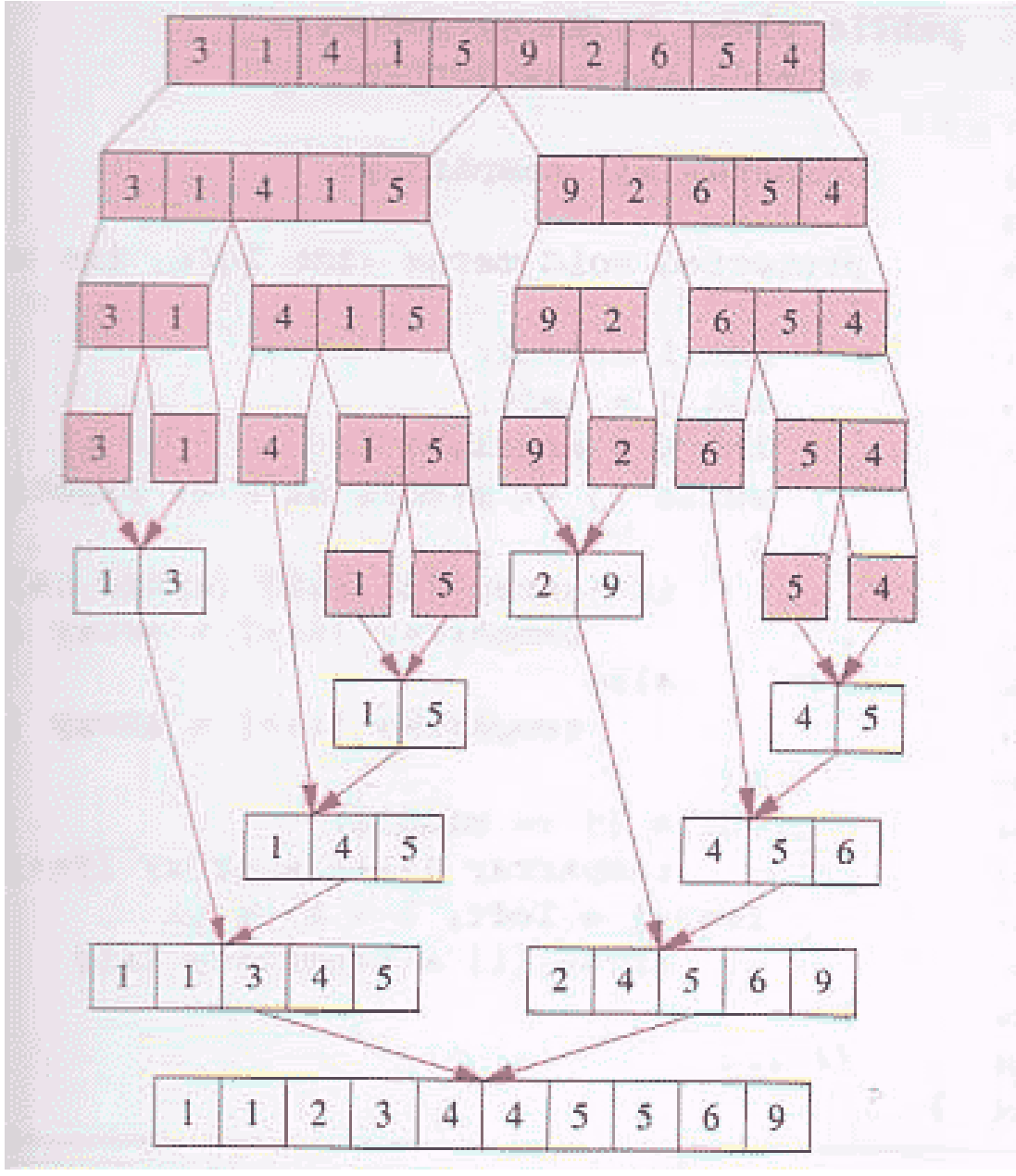
- Vediamo un altro algoritmo di ordinamento.

```
/* mergeSort( V, N ) */
```

- se il vettore contiene almeno due elementi:
  - **dividi** il vettore `V` in due vettori, ciascuno di dimensione `N/2`;
  - **copia** le due metà in due nuovi vettori, `V1` e `V2`;
  - **ordina** `V1` e `V2` separatamente (tramite chiamata ricorsiva);
  - **riunisci** `V1` e `V2` in modo ordinato nel vettore `V` originario.

- Si veda il disegno nella pagina seguente, che raffigura l'esecuzione del merge sort in un vettore di 10 interi di esempio.

△ 2. Un Altro Algoritmo di Ordinamento (*Merge Sort*) ▽



△ 2. Un Altro Algoritmo di Ordinamento (*Merge Sort*) ▽

- Si chiede di implementare in C:
  1. una funzione `vectorCopy()` , il cui prototipo sia:

```
int vectorCopy( int *V1, int *V, int N1 );
```

     che copi in `V1` i primi `N1` elementi di `V`;
  2. una funzione `merge()`, il cui prototipo sia:

```
int merge( int *V, int *V1, int N1, int *V2, int N2 );
```

     che costruisca un vettore `V` di interi ordinato a partire da due vettori `V1` e `V2` di interi ordinati, di dimensione rispettivamente `N1` e `N2`;
  3. l'algoritmo `mergeSort()` per ordinare un vettore di interi;
  4. una funzione `countM()` per contare il **numero di confronti tra due elementi del vettore** effettuati da `mergeSort()`.

△ 3. Valutazione empirica degli algoritmi ▽

- Si progetti una serie di esperimenti per valutare l'efficienza dei due algoritmi in modo empirico:
  1. Si definiscano dei vettori di interi dal contenuto (più o meno) casuale (per esempio: vettori di 10, 20, 50 elementi, sia non ordinati sia già ordinati in senso crescente/decrescente);
  2. Si eseguano le procedure di ordinamento implementate nei due punti precedenti, contando il numero di confronti eseguiti per ordinare ciascun vettore;
  3. Si visualizzi per ciascun esperimento:
     - dimensione del vettore (numero di elementi),
     - se già ordinato (se sì: se in senso crescente/decrescente),
     - algoritmo di ordinamento usato,
     - numero di confronti effettuati.
- Che conclusioni si possono trarre dai risultati sperimentali?