ALGORITMI DI ORDINAMENTO

a cura di

Simona Colucci e Giacomo PISCITELLI
Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari

Bari febbraio 2004

INDICE

1. Il problema dell'ordinamento

Il problema che ci si propone di risolvere è quello di ordinare gli elementi di un insieme secondo una prefissata relazione d'ordine, che dipende ovviamente dalla natura dei dati da ordinare.
I metodi di ordinamento si applicano ad insiemi di dati elementari (singoli numeri o lettere) e anche a dati rappresentati in forma complessa. In questo secondo caso in generale si fa riferimento ad un insieme di record, dei quali si seleziona un campo che viene usato come **campo chiave** per l'ordinamento. Perciò il risultato dell'ordinamento sarà una ridisposizione dei record secondo l'ordinamento introdotto nel campo considerato.

I metodi di ordinamento, a seconda del loro modo di operare, possono essere classificati in quattro categorie:

1. *Inserzione*: Si considerano gli elementi uno alla volta e ciascun nuovo elemento viene collocato nella posizione che gli compete all'interno dell'insieme di elementi considerati in precedenza.
2. *Selezione*: Si scandisce l'insieme in modo da individuare l'elemento più piccolo, che viene separato dal resto dell'insieme. Dell'insieme rimasto viene ancora trovato il più piccolo elemento e lo si colloca di seguito a quello trovato in precedenza. Questo procedimento si itera fino a quando non si sono considerati tutti gli elementi.
3. *Scambio*: Si scandisce tutto l'insieme, se due elementi vengono trovati fuori posto si scambiano tra loro. Questo procedimento viene ripetuto fino a quando non si hanno più cambiamenti.
4. *Fusione*: Si sfrutta un'operazione più semplice dell'ordinamento, chiamata fusione, che consiste nel formare un insieme ordinato a partire da due insiemi già ordinati.

Nel seguito come insieme di dati da ordinare considereremo un vettore di numeri interi ALFA di dimensione N. ALFA(i) indicherà l'i-esimo elemento del vettore.
La relazione d'ordine che si prende in considerazione è l'usuale ordinamento numerico (basato sui concetti di minore e maggiore). Supporremo inoltre di realizzare un *ordinamento crescente*.

2. Ordinamento per selezione (Selection Sort) – Il metodo di selezione diretta.

Il metodo fa uso di due indici di posizione :

- i: è l'indice dell'elemento del vettore che delimita l'inizio dell'insieme dei valori presi in considerazione ad ogni singolo passo.
- j: è un cursore che ad ogni passo percorre l'insieme dei valori da scandire.

All'inizio l'indice i viene posizionato sul primo elemento del vettore (ALFA); j invece comincia a scandire dal secondo elemento.
Il primo elemento viene confrontato con i rimanenti (scanditi da j), scambiando di posto il primo elemento e quello di posizione j se quest'ultimo ha valore inferiore.
Al secondo passo l'indice i viene portato alla seconda posizione, perché il nuovo insieme dei valori da scandire non comprende il primo, che è sicuramente il minimo.
L'indice j percorre l'insieme dei valori da scandire, a partire dall'elemento successivo a i; il processo riprende in maniera analoga al passo precedente e viene iterato finchè l'indice i non giunge all'ultima posizione.

ESEMPIO:

25	8	8	8	8
37	37	25	25	25
41	41	41	27	27
27	27	->27	41	37
->8	-> 25	37	->37	41

Nell'esempio le colonne rappresentano i passi dell'algoritmo, gli elementi evidenziati in azzuro il sottoinsieme del vettore da scandire per l'ordinamento ad ogni passo e la freccia l'elemento minimo di ognuno dei sottoinsiemi.

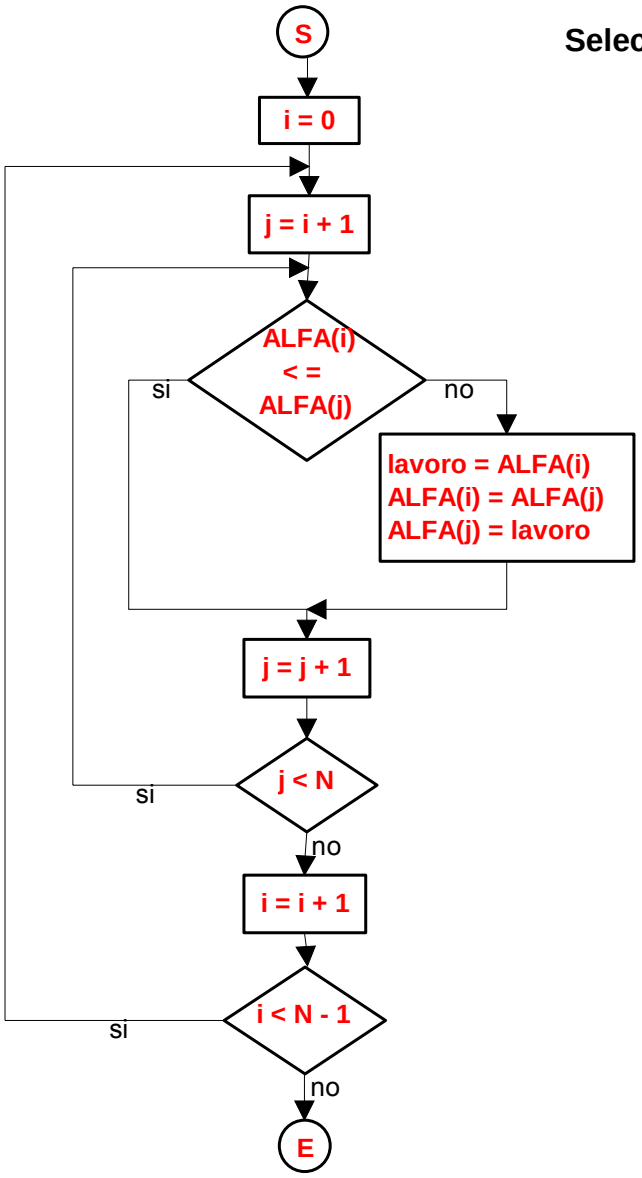
Il metodo di selezione diretta presenta un numero di confronti proporzionale ad N^2.
Infatti al primo passaggio si effettuano N-1 confronti, al secondo N-2 e così via, fino ad arrivare all'ultimo confronto; si ha pertanto:

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

Per quanto riguarda il numero di spostamenti essi sono al più N-1.
La prestazione del metodo per quanto riguarda il numero di confronti non è influenzata dalla configurazione iniziale dei dati.

Per comprendere il funzionamento dell'algoritmo se ne riporta di seguito il flow-chart:

Selection sort



3. Ordinamento per scambio

3.1 Il metodo dell'affioramento (Bubble Sort).

Il procedimento tipico di qualsiasi algoritmo di scambio è il seguente:
si considerano i primi due elementi dell'insieme; se il primo elemento della coppia è maggiore del secondo i due elementi vengono scambiati di posto tra di loro; si considera ora una nuova coppia formata dal secondo e dal terzo elemento dell'insieme, si confrontano e si scambiano le posizioni se necessario; questo procedimento si ripete, considerando coppie di elementi consecutivi, fino a quando non si è esaurito tutto l'insieme.
Il metodo di affioramento ha la particolarità che gli elementi più "pesanti" tendono ad andare verso il basso e quelli più leggeri verso l'alto.
In particolare ad ogni passo (step) dell'algoritmo l'elemento in assoluto più pesante sarà collocato nella posizione finale, che al passo successivo non verrà più scandita.
Risulta evidente che il numero massimo di passi è N-1 , ma dato che ad ogni passo più elementi potrebbero essere "a posto", i passi potrebbero anche di meno.
La variabile più importante usata nel procedimento è:

- scambi: viene inizializzata a 0 ad ogni passo e serve per controllare se ci sono stati scambi ad un certo passo.

ESEMPIO:

25	25	25	25	8
37	37	27	8	25
41	27	8	27	27
27	8	37	37	37
8	41	41	41	41

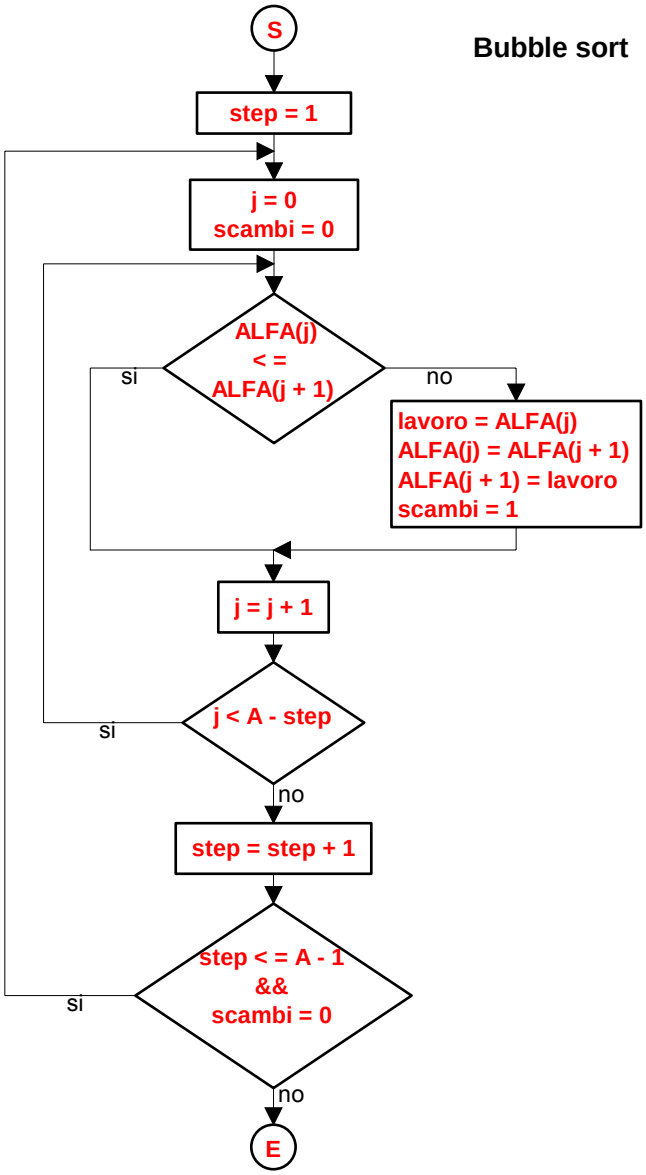
Nell'esempio le colonne rappresentano i passi dell'algoritmo, gli elementi evidenziati in azzuro il sottoinsieme del vettore da scandire per l'ordinamento ad ogni passo.

Per il bubble-sort l'analisi del caso peggiore porta a concludere che il metodo è di ordine N^2 . Il caso peggiore si ha quando l'insieme è ordinato in modo decrescente.
In tale eventualità il numero complessivo di confronti è

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

Si può dimostrare che questa valutazione è vera anche nella media dei casi.
Nel caso in cui in qualche passo vada a posto più di un elemento il numero di confronti complessivi da operare sarà inferiore.
Per gli spostamenti si segue un ragionamento analogo (nel caso peggiore sono N(N-1)/2).
La prestazione del metodo è pertanto influenzata dalla configurazione iniziale dei dati.

Per comprendere il funzionamento dell'algoritmo se ne riporta di seguito il flow-chart:



3.2 Quicksort (ordinamento non decrescente)

Considerato un elemento a caso, che chiamiamo perno o separatore, si effettua una partizione dell'insieme collocando nella sua giusta posizione il perno e allocando alla sua sinistra tutti gli elementi più piccoli ed alla sua destra tutti gli elementi più grandi del perno stesso.
Sui due sottoinsiemi così individuati si applica ricorsivamente il concetto di partizione.
Il concetto fondamentale di questo tipo di algoritmo è rappresentato dalla figura 1:

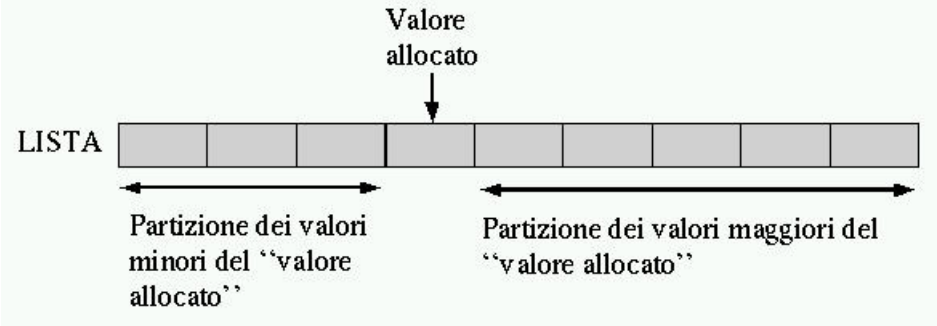


Figura 1: Il concetto base dell'algoritmo del Quicksort: suddivisione del vettore in due gruppi disordinati, separati da un valore piazzato correttamente nel suo posto rispetto all'ordinamento.

Una sola scansione del vettore è sufficiente per collocare definitivamente un elemento (per esempio il primo) nella sua destinazione finale e allo stesso tempo per lasciare tutti gli elementi con un valore inferiore a quello da una parte, anche se disordinati, e tutti quelli con un valore maggiore, dall'altra.
In questo modo, attraverso delle chiamate ricorsive, è possibile elaborare i due segmenti del vettore rimasti da riordinare.
L'algoritmo può essere descritto grossolanamente come:

1. localizzazione della collocazione finale del primo valore, separando in questo modo i valori;
2. ordinamento del segmento precedente all'elemento collocato definitivamente;
3. ordinamento del segmento successivo all'elemento collocato definitivamente.

Variabili

`LISTA`

Il vettore da ordinare in modo crescente.

`A`

L'indice inferiore del segmento di vettore da ordinare.

`Z`

L'indice superiore del segmento di vettore da ordinare.

`CF`

Sta per «collocazione finale» ed è l'indice che cerca e trova la posizione giusta di **`LISTA[A]`** nel vettore.

`I`

È l'indice che insieme a **`CF`** serve a ripartire il vettore.

Indichiamo con **`PART`** la procedura che esegue la scansione del vettore, o di un suo segmento, per determinare la collocazione finale (indice **`CF`**) del primo elemento (del vettore o del segmento in questione).

Sia **`LISTA`** il vettore da ordinare. Il primo elemento da collocare corrisponde inizialmente a **`LISTA[A]`**, e il segmento di vettore su cui intervenire corrisponde a **`LISTA[A:Z]`** (cioè a tutti gli elementi che vanno dall'indice **`A`** all'indice **`Z`**).

Alla fine della prima scansione, l'indice **CF** rappresenta la posizione in cui occorre spostare il primo elemento, cioè **LISTA[A]**. In pratica, **LISTA[A]** e **LISTA[CF]** vengono scambiati.

Durante la scansione che serve a determinare la collocazione finale del primo elemento, **PART** deve occuparsi di spostare gli elementi prima o dopo quella posizione, in funzione del loro valore, in modo che alla fine quelli inferiori o uguali a quello dell'elemento da collocare si trovino nella parte inferiore, e gli altri dall'altra. In pratica, alla fine della prima scansione, gli elementi contenuti in **LISTA[A:(CF-1)]** devono contenere valori inferiori o uguali a **LISTA[CF]**, mentre quelli contenuti in **LISTA[(CF+1):Z]** devono contenere valori superiori.

Indichiamo con **QSORT** la procedura che esegue il compito complessivo di ordinare il vettore. Il suo lavoro consisterebbe nel chiamare **PART** per riordinare gli elementi che vanno dal primo all'ultimo del vettore **LISTA** restituendo l'indice della collocazione finale, e quindi di richiamare se stessa in modo da riordinare la prima parte e poi la seconda.

Assumendo che **PART** e le chiamate ricorsive di **QSORT** svolgano il loro compito correttamente, si potrebbe fare un'analisi informale dicendo che se l'indice **Z non** è maggiore di **A**, allora c'è un elemento (o nessuno) all'interno di **LISTA[A:Z]** e inoltre, **LISTA[A:Z]** è già nel suo stato finale. Se **Z** è maggiore di **A**, allora (per assunzione) **PART** ripartisce correttamente **LISTA[A:Z]**. L'ordinamento separato dei due segmenti (per assunzione eseguito correttamente dalle chiamate ricorsive) completa l'ordinamento di **LISTA[A:Z]**. Le figure 2 e 3 mostrano due fasi della scansione effettuata da **PART** all'interno del vettore o del segmento che gli viene fornito.

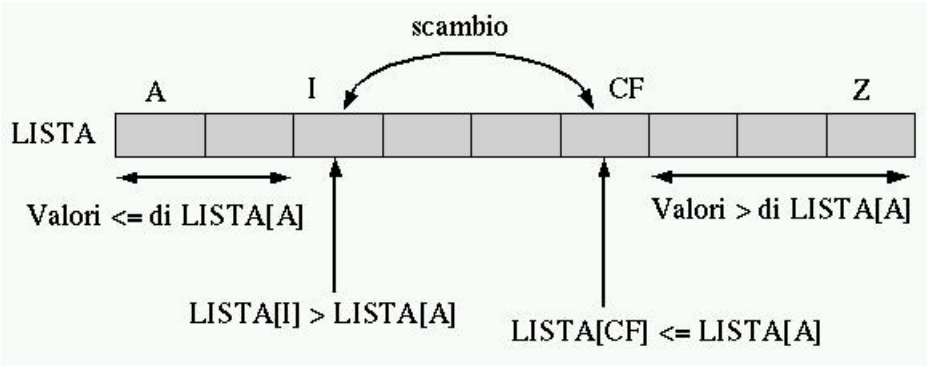


Figura 2: La scansione del vettore da parte di **PART** avviene portando in avanti l'indice **I** e portando indietro l'indice **CF**. Quando l'indice **I** localizza un elemento che contiene un valore maggiore di **LISTA[A]**, e l'indice **CF** localizza un elemento che contiene un valore inferiore o uguale a **LISTA[A]**, gli elementi cui questi indici fanno riferimento vengono scambiati, quindi il processo di avvicinamento tra **I** e **CF** continua.

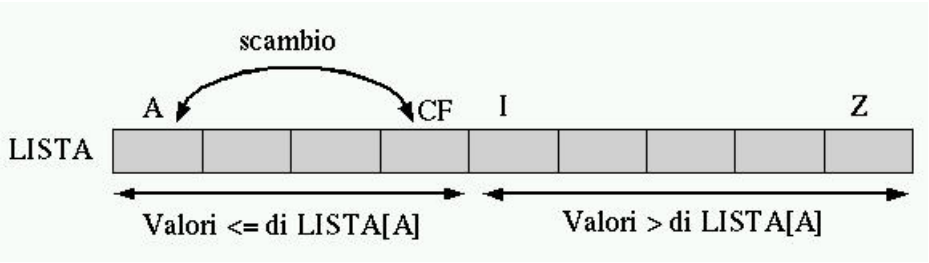
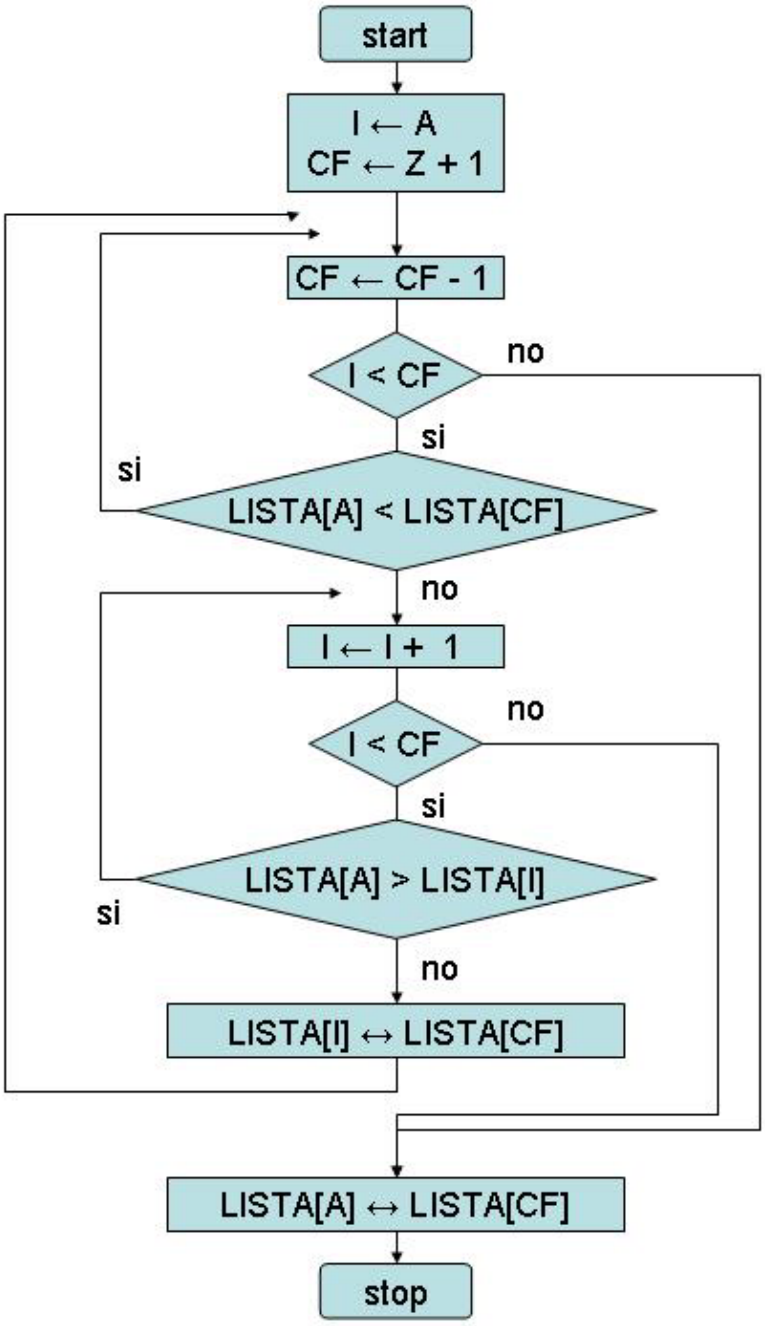


Figura 3: Quando la scansione è giunta al termine, quello che resta da fare è scambiare l'elemento **LISTA[A]** con **LISTA[CF]**.

In pratica, l'indice **I**, iniziando dal valore **A+1**, viene spostato verso destra fino a che viene trovato un elemento maggiore di **LISTA[A]**, quindi è l'indice **CF** a essere spostato verso sinistra, iniziando dalla stessa posizione di **Z**, fino a che viene incontrato un elemento minore o uguale a **LISTA[A]**. Questi elementi vengono scambiati e lo spostamento di **I** e **CF** riprende. Ciò prosegue fino a che **I** e **CF** si incontrano, e in quel momento **LISTA[A:Z]** è stata ripartita, e **CF** rappresenta la collocazione finale per l'elemento **LISTA[A]**.

Di seguito si riporta il flow-chart dell'algoritmo di partizione:



Una volta definito l'algoritmo di partizione l'algoritmo di quicksort può essere schematicamente definito come:

1. se A= Z stop
2. esegui l'algoritmo di partizione su LISTA delimitato da A e Z determinando la posizione CF dell'elemento separatore
3. Se A < CF esegui quicksort su LISTA delimitato da A, CF-1
4. Se CF < Z esegui quicksort su LISTA delimitato da CF+1, Z

L'algoritmo di quicksort viene utilizzato quando la disposizione degli elementi è il più possibile casuale; in questo caso il numero medio di confronti risulta in ordine di grandezza inferiore al numero di confronti dei metodi visti in precedenza.
Data la complessità dell'analisi del quicksort, ci riferiremo solo al caso migliore: il caso in cui la disposizione di dati è tale che ad ogni passo il perno viene collocato alla metà esatta dell'insieme che si considera (es. 85 32 10 71 63 52 21 101 124 152 112 132 149 96 48).
Supponiamo che la dimensione dell'insieme sia:

$N= 2^h-1$

Il numero dei confronti per ogni passaggio risulta proporzionale ad N, perché N sono gli elementi globalmente interessati, mentre il numero di passaggi è h-1. Quindi il numero di confronti è proporzionale a N*(h-1).
Tale ordine risulta valido anche nella media dei casi. Il caso peggiore si ha quando il vettore è ordinato, caso in cui il numero di passaggi di partizione è N e il numero di confronti risulta proporzionale ad N^2.
La prestazione del metodo per il confronto è influenzata dalla configurazione iniziale dei dati; il numero di spostamenti è certamente inferiore al numero di confronti.

ESEMPIO
Vediamo il comportamento dell'algoritmo riferito all'insieme (34 93 64 25 18 29 76 81), di 8 elementi delimitato da A = 1 e Z = 8.

quicksort su LISTA delimitato da 1 e 8.

1. siccome A=Z è falso si va in sequenza;
2. si esegue partizione su LISTA delimitato da A=1 e Z=8 trovando la posizione CF = 4 per il separatore 34.
3. poiché A < CF è vero si esegue quicksort su LISTA delimitato da A=1 e Z= 3
   - 3.1 siccome A=Z è falso si va in sequenza;
   - 3.2 si esegue partizione su LISTA delimitato da A=1 e Z=3 trovando la posizione CF = 3 per il separatore 29.
   - 3.3 poiché A < CF è vero si esegue quicksort su LISTA delimitato da A=1 e Z= 2
     - 3.3.1 siccome A=Z è falso si va in sequenza;
     - 3.3.2 si esegue partizione su LISTA delimitato da A=1 e Z=2 trovando la posizione CF=2 per il separatore 25.
     - 3.3.3 poiché A < CF è vero si esegue quicksort su LISTA delimitato da A=1 e Z= 1
       - 3.3.3.1 siccome A=Z è vero si rientra
     - 3.3.4 Poiché CF< Z è falso, si rientra

3.4 Poiché CF< Z è falso, si rientra

4. poiché CF < Z è vero si esegue quicksort su LISTA delimitato da A=5 e Z= 8

4.1 siccome A=Z è falso si va in sequenza;

4.2 si esegue partizione su LISTA delimitato da A=5 e Z=8 trovando la posizione CF = 5 per il separatore 64.

4.3 poiché A < CF è falso si va in sequenza

4.4 poiché CF < Z è vero si esegue quicksort su LISTA delimitato da A=6 e Z= 8

4.4.1 siccome A=Z è falso si va in sequenza;

4.4.2 si esegue partizione su LISTA delimitato da A=6 e Z=8 trovando la posizione CF=8 per il separatore 93.

4.4.3 poiché A < CF è vero si esegue quicksort su LISTA delimitato da A=6 e Z= 7

4.4.3.1 siccome A=Z è falso si va in sequenza

4.4.3.2 si esegue partizione su LISTA delimitato da A=6 e Z=7 trovando la posizione CF=7 per il separatore 81.

4.4.3.3 poiché A < CF è vero si esegue quicksort su LISTA delimitato da A=6 e Z= 6

4.4.3.3.1 siccome A=Z è vero si rientra

4.4.3.4 poiché CF< Z è falso, si rientra

4.4.4 Poiché CF< Z è falso, si rientra

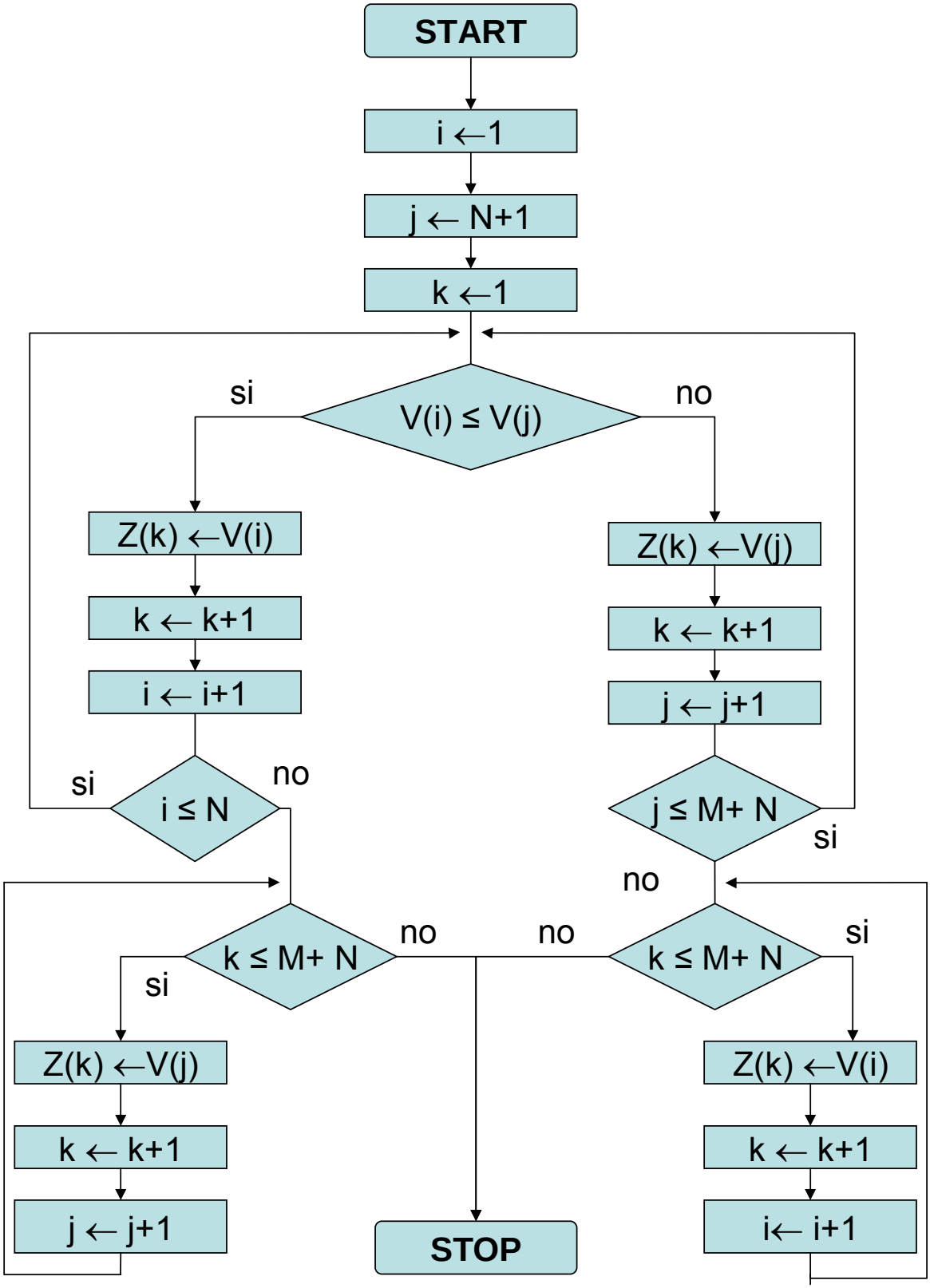
4.Ordinamento per fusione - Algoritmo Merge Sort

Sfrutta il metodo risolutivo di un problema più semplice di quello dell'ordinamento: quello della **fusione**, in cui si parte da due insiemi ordinati, che vengono fusi per generare un unico insieme ordinato.
Si acquisisce un elemento dal primo insieme e uno dal secondo, vengono confrontati tra loro e si copia nel vettore di destinazione l'elemento più piccolo; si acquisisce un nuovo elemento dall'insieme dove è stata tolto l'elemento e lo si riconfronta con quello rimasto.
Il procedimento ha termine quando sono esaminati tutti gli elementi.
L'algoritmo assume che i due insiemi di partenza, di dimensione N ed M, siano contenuti in un vettore V di dimensione N+M. Il risultato viene posto in un vettore Z, di dimensione N+M.
L'algoritmo fa uso di tre variabili:

- *i* è un cursore che delimita in V il primo insieme ordinato di partenza $1 \leq i \leq N$
- *j* è un cursore che delimita in V il secondo insieme ordinato di partenza $N+1 \leq j \leq N+M$
- *k* è un cursore che scorre su tutto il vettore Z $1 \leq k \leq N+M$

4.1 L'algoritmo di fusione

La fusione è un processo più semplice dell'ordinamento: si ha nel caso peggiore un numero di confronti uguale ad N+M . Una delle configurazioni di caso peggiore si ha quando gli elementi vengono presi alternativamente da due insiemi (es. Primo vettore: 503 703 765; Secondo Vettore : 87 512 757)
Si riporta di seguito il flow-chart dell'algoritmo di fusione.


START
i ←1
j ← N+1
k ←1
si
V(i) ≤ V(j)
no
Z(k) ←V(i)
k ← k+1
i ← i+1
Z(k) ←V(j)
k ← k+1
j ← j+1
si
i ≤ N
no
j ≤ M+ N
si
no
k ≤ M+ N
no
no
k ≤ M+ N
si
si
Z(k) ←V(j)
k ← k+1
j ← j+1
Z(k) ←V(i)
k ← k+1
i← i+1
STOP

4.2 Il merge-sort binario

Il **mergesort binario** ha come base l'algoritmo di fusione appena descritto.
Esso suddivide gli elementi di V in coppie costituite da elementi consecutivi (primo-secondo, terzo-quarto,..) e ordina tra di loro gli elementi di una stessa coppia; si ottengono così delle coppie ordinate; a questo punto si ordinano tra di loro gli elementi della prima e della seconda coppia, della terza e della quarta, …. ottenendo alla fine delle quadruple ordinate; si ripete fino ad ottenere un unico insieme ordinato.
L'algoritmo suppone di partire da un insieme disordinato, di dividerlo in due sezioni e di fondere le due sezioni avendovi prima applicato ricorsivamente l'algoritmo.
Per semplicità supponiamo che i due insiemi da ordinare siano allocati nel vettore V in due aree contigue delimitate da u,i la prima e i+1,t la seconda.
La variabile i è in sostanza usata per dividere in due l'area di memoria delimitata da u e t.
Il risultato della fusione viene sempre collocato nel vettore Z. Alla fine della fusione supponiamo che in V si ricopi l'insieme ordinato.
L'algoritmo può allora essere formulato nel modo seguente:

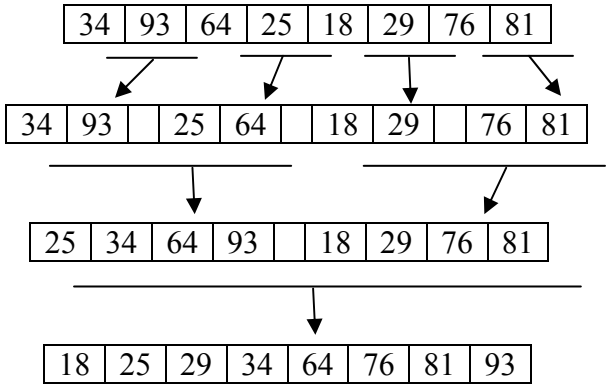
1. se u = t allora stop
2. i = $\lfloor (u+t)/2 \rfloor$
3. esegui merge sort su V delimitato da u ed i
4. esegui merge sort su V delimitato da i+1 ed t
5. esegui fusione su V delimitato da u , i e da i+1, t

L'algoritmo di merge-sort è particolarmente vantaggioso perché il numero di confronti è lo stesso in tutti i casi, anche quello peggiore.
Il metodo opera partizioni dell'insieme di partenza che sono ad ogni passo della stessa dimensione.
Supponiamo che la dimensione dell'insieme sia:

$N= 2^h$

Il numero dei confronti per ogni passaggio risulta N, perché N sono gli elementi globalmente interessati, mentre il numero di passaggi è h = $\log_2$ N. Quindi il numero di confronti è proporzionale a N*(h) = N * $\log_2$ N.

Esempio : (34 93 64 25 18 29 76 81) N=8



Applichiamo l'algoritmo su questo vettore (inizialmente u=1 e t =8).

Sort-merge su V delimitato da u=1 e t=8;

1. siccome u=t è falso si va in sequenza;
2. i ← 4 ;
3. esegui sort-merge su V delimitato da u=1 e t = 4;
   - 3.1 siccome u=t è falso si va in sequenza;
   - 3.2 i ← 2 ;
   - 3.3 esegui sort-merge su V delimitato da u=1 e t = 2;
     - 3.3.1 siccome u=t è falso si va in sequenza;
     - 3.3.2 i ← 1 ;
     - 3.3.3 esegui sort-merge su V delimitato da u=1 e t = 1;
       - 3.3.3.1 siccome u=t è vero si rientra;
     - 3.3.4 esegui sort-merge su V delimitato da u=2 e t = 2;
       - 3.3.4.1 siccome u = t è vero si rientra;
     - 3.3.5 esegui fusione su V delimitato da u=1 i=1; i+1=2 t=2
   - 3.4 esegui sort-merge su V delimitato da u=3 e t = 4;
     - 3.4.1 siccome u=t è falso si va in sequenza;
     - 3.4.2 i ← 3 ;
     - 3.4.3 esegui sort-merge su V delimitato da u=3 e t = 3;
       - 3.4.3.1 siccome u=t è vero si rientra;
     - 3.4.4 esegui sort-merge su V delimitato da u=4 e t = 4;
       - 3.4.4.1 siccome u = t è vero si rientra;
     - 3.4.5 esegui fusione su V delimitato da u=1 i=2; i+1=3 t=4
   - 3.5 esegui fusione su V delimitato da u=1 i=2; i+1=3 t=4

4. esegui sort-merge su V delimitato da u=5 e t = 8;
   - 4.1 siccome u=t è falso si va in sequenza;
   - 4.2 i ← 6 ;
   - 4.3 esegui sort-merge su V delimitato da u=5 e t = 6;
     - 4.3.1 siccome u=t è falso si va in sequenza;
     - 4.3.2 i ← 5 ;
     - 4.3.3 esegui sort-merge su V delimitato da u=5 e t = 5;
       - 4.3.3.1 siccome u=t è vero si rientra;
     - 4.3.4 esegui sort-merge su V delimitato da u=6 e t = 6;
       - 4.3.4.1 siccome u = t è vero si rientra;
     - 4.3.5 esegui fusione su V delimitato da u=5 i=5; i+1=6 t=6
   - 4.4 esegui sort-merge su V delimitato da u=7 e t = 8;
     - 4.4.1 siccome u=t è falso si va in sequenza;
     - 4.4.2 i ← 7 ;
     - 4.4.3 esegui sort-merge su V delimitato da u=7 e t = 7;
       - 4.4.3.1 siccome u=t è vero si rientra;
     - 4.4.4 esegui sort-merge su V delimitato da u=8 e t = 8;
       - 4.4.4.1 siccome u = t è vero si rientra;
     - 4.4.5 esegui fusione su V delimitato da u=7 i=7; i+1=8 t=8
   - 4.5 esegui fusione su V delimitato da u=5 i=6; i+1=7 t=8
5. esegui fusione su V delimitato da u=1 i=4; i+1=5 t=8

Al primo passaggio abbiamo N/2 coppie di dimensione N/N/2 =2 su cui applicare l'algoritmo di fusione (per il quale nel caso peggiore si hanno un numero di confronti pari alla grandezza della coppia). Il numero di confronti al primo passo è pertanto (N/2) * 2= N.

Al secondo passaggio abbiamo N/4 vettori di dimensione N/N/4 =4 su cui applicare l'algoritmo di fusione (per il quale nel caso peggiore si hanno un numero di confronti pari alla grandezza della coppia). Il numero di confronti al secondo passo è pertanto (N/4) * 4 = N.

Procedendo è facile comprendere che il numero dei confronti risulta N ad ogni passaggio.