ALGORITMI DI BASE (4° Parte):
ORDINAMENTO "BUBBLE-SORT" e INSERIMENTO IN ORDINE

Ordinamento di Dati tramite l'Algoritmo di Ordinamento chiamato "BUBBLE-SORT"

Sviluppo Top-Down dell'algoritmo Bubble-Sort …

```
do
{
   << scorri la lista: ogni volta che trovi un elemento maggiore del successivo, scambiali >>
}
while  << hai effettuato degli scambi >>
```

Affiniamo l'operazione interna al ciclo do-while …

```
do
{
   // scorri la lista e, se un elemento è maggiore del successivo, scambiali …
   << inizialmente, ipotizza di non aver fatto scambi >>
   for (int K = 0; K <= N - 2; K++)          // scorri la lista dal primo al PENULTIMO elemento …
        if  ( << l'elemento in pos. K  è  MAGGIORE  del successivo >> )
        {
             << scambia  l'elemento in pos. K  con il successivo >>
             << ricordati che hai effettuato uno scambio >>
        }
}
while  << hai effettuato degli scambi >>
```

Ora il codice completo …

```
do
{
   // scorri la lista e, se un elemento è maggiore del successivo, scambiali …
   HaFattoScambi = false;                    // inizialmente, ipotizza di non aver fatto scambi

   for (int K = 0; K <= N - 2; K++)          // scorri la lista dal primo al PENULTIMO elemento
        if  ( Lista[K] > Lista[K+1] )        // se l'elemento in pos. K  è  MAGGIORE  del successivo
        {
             // scambia  l'elemento in pos. K  con il successivo …
             Temp = Lista[K];                // salva, l'elemento in pos. K, in una variabile di appoggio
             Lista[K] = Lista[K + 1];        // copia il successivo nell'elemento in pos. K
             Lista[K + 1] = Temp;            // recupera l'elemento che era in pos. K e copialo nel successivo

             HaFattoScambi = true;           // ricordati che hai effettuato uno scambio
        }
}
while  ( HaFattoScambi )                     // ripeti se hai effettuato almeno uno scambio
```

Inserimento di un Nuovo Dato in una Lista Ordinata mantenendo l'Ordinamento della Lista (Inserimento in Ordine)

Sviluppo Top-Down dell'Inserimento in Ordine …

<< determina la posizione di inserimento PosIns del NuovoDato nella Lista >>
<< sposta in avanti tutti gli elementi della Lista dalla posizione PosIns in poi >>
<< inserisci il NuovoDato memorizzandolo nella posizione PosIns >>

Affiniamo le operazioni individuate …

```
// determina la posizione di inserimento PosIns del NuovoDato nella Lista …
int K = 0;                          // fai iniziare la scansione dal primo elemento della Lista
while ( ( K <= N-1 )  &&            // ripeti mentre "ci sono ancora elementi da esaminare" E …
        (Lista[K] < NuovoDato) )    // …  " l'elemento in esame è MINORE del nuovo dato"
   K++;                             // passa all'elemento successivo
PosIns = K;                         // in ogni caso, al termine del ciclo, K indica la posizione di inserimento

// sposta in avanti tutti gli elementi della Lista dalla posizione PosIns in poi …
for ( K = N-1;  K >= PosIns;  K - -)   // scansione parziale della lista "all'indietro" (dall'ultimo a PosIns)
   Lista[K+1] = Lista[K];              // copia l'elemento in esame Lista[K] nella posiz. successiva Lista[K+1]

// inserisci il NuovoDato memorizzandolo nella posizione PosIns …
Lista[PosIns] = NuovoDato;          // creato lo "spazio", poni il nuovo dato nella posizione richiesta
N++;                                // avendo aggiunto un elemento, incrementa la variabile N
```

E' possibile ottimizzare il codice cercando la posizione di inserimento con una scansione all'indietro e spostando gli elementi già durante tale scansione …

```
// determina la posizione di inserimento PosIns e, al tempo stesso, sposta gli elementi in avanti …
int K = N-1;                        // fai iniziare la scansione dall'ultimo elemento della Lista
while ( ( K >= 0 )  &&              // ripeti mentre "ci sono ancora elementi da esaminare" E …
        ( Lista[K] > NuovoDato ) )  // …  " l'elemento in esame è MAGGIORE del nuovo dato"
{
   Lista[K+1] = Lista[K];           // copia l'elemento in esame Lista[K] nella posiz. successiva Lista[K+1]
   K - -;                           // passa all'elemento successivo (scansione all'indietro)
}
PosIns = K;                         // in ogni caso, al termine del ciclo, K indica la posizione di inserimento

// inserisci il NuovoDato memorizzandolo nella posizione PosIns …
Lista[PosIns] = NuovoDato;          // creato lo "spazio", si pone il nuovo dato nella posizione richiesta
N++;                                // avendo aggiunto un elemento, si incrementa la variabile N
```

Nota Bene: l'operatore *&& (AND condizionale)*, a differenza dell'operatore *& (AND logico)*, valuta la seconda condizione solo se la prima è vera. Per questo motivo, quando K, decrementandosi progressivamente, raggiunge il valore "-1", la prima condizione *(K >= 0)* risulta falsa e, quindi, la seconda condizione *(Lista[K] > NuovoDato)* non viene valutata. Questo evita l'errore generato dal fatto che la posizione "-1" non esiste nella lista.