DOMANDA: Illustrare l'Algoritmo di Ordinamento per Inserimento (AOINS)

RISPOSTA:
Lo scopo di un Algoritmo di Ordinamento è ordinare (in senso crescente) un array di dati. I dati possono essere numeri, caratteri, stringhe di caratteri o qualsiasi altro insieme di dati su cui sia definita una relazione di ordine.

L'AOINS ha come dati di input un array e il suo size e ha come dato di output lo stesso array con i dati ordinati. L'AOINS non fa uso di altri array o strutture dati e per questo motivo si dice che opera "in situ" o "in place".

L'AOINS è basato sull'applicazione dell'approccio incrementale al problema dell'ordinamento. Infatti, a ogni passo l'algoritmo risolve una istanza del problema dell'ordinamento, cioè l'ordinamento di una "porzione" dell'array A; all'aumentare del passo aumenta la dimensione dell'istanza del problema, fino ad arrivare alla dimensione n del problema dato.
Diamo una rapida analisi della meccanica dell'AOINS. Al generico passo `i`, essendo già stata ordinata nei passi precedenti la porzione `0..(i-1)` dell'array, l'AOINS ordinerà la porzione `0..i` dell'array A. E' importante osservare che tale ordinamento non è quello definitivo e che nei successivi passi tale ordinamento può essere modificato dall'inserimento di altri tra i rimanenti elementi dell'array.
L'idea di base è che se i primi `i-1` elementi sono ordinati, allora per ordinare i primi `i` elementi basta inserire l'`i`-simo elemento nella sua "giusta" posizione, cioè nella posizione in cui tale elemento avrà alla sua sinistra gli elementi minori o uguali e alla sua destra gli elementi uguali o maggiori.
Quindi, a ogni passo l'AOINS risolve il problema di "inserire" (di qui il nome dell'algoritmo) un elemento dell'array nella porzione già ordinata alla sua sinistra, in modo che la lunghezza della porzione ordinata aumenti di 1.
Un esempio aiuta a capire la dinamica dell'AOINS. Si consideri un array A di n=8 elementi interi

27	41	36	11	28	13	52	31
i =0	1	2	3	4	5	6	n-1=7

Nell'ottica incrementale, il primo elemento dell'array si trova nella posizione corretta: una porzione di lunghezza 1 è sempre ordinata!
Quindi, il primo passo (`i`=1) dell'AOINS consiste nell'ordinare i primi 2 elementi dell'array, cioè la porzione 0..1 dell'array A.

27	41	36	11	28	13	52	31
0	**i =1**	2	3	4	5	6	n-1=7

Ciò si ottiene considerando l'`i`-simo elemento, cioè l'elemento di indice 1 che è 41, e inserendolo nella "giusta" posizione, in modo che la porzione 0..1 sia ordinata. Poiché 41 è maggiore di 27, la sua posizione corretta è proprio quella in cui già si trova e, in questo caso, nessuno spostamento o scambio deve essere effettuato.
Al secondo passo (`i`=2) si devono ordinare i primi 3 elementi (porzione 0..2) sapendo che la porzione 0..1 è già ordinata. Ciò si ottiene considerando l'`i`-simo elemento, cioè l'elemento di indice 2 che è 36, e inserendolo nella "giusta" posizione, in modo che la porzione 0..2 risulti ordinata.

27	41	36	11	28	13	52	31
0	1	**i =2**	3	4	5	6	n-1=7

In generale, la "giusta" posizione viene determinata con un procedimento iterativo che procede a ritroso, esaminando successivamente le posizioni (`i`-1), (`i`-2),…. ed eventualmente fino alla prima pozione

(indice 0). In tale procedimento l'elemento da inserire, 36 nell'esempio, viene confrontato con gli elementi alla sua sinistra. Se l'elemento alla sua sinistra è maggiore allora quell'elemento è spostato (*shiftato*) di una posizione a destra; altrimenti, l'elemento da inserire viene inserito nella posizione a destra dell'elementi che risulta essere minore. Nel nostro caso, si ha che, poiché 41 è maggiore di 36, allora 41 viene spostato a destra; poi, poiché 27 è minore di 36, allora 36 è inserito nella posizione 1, cioè la posizione immediatamente a destra di 27. Al termine del passo, l'array A ha i seguenti valori

27	36	41	11	28	13	52	31
0	1	**i =2**	3	4	5	6	n-1=7

Si noti che l'istanza 3 del problema, cioè l'ordinamento dei primi 3 elementi dell'array A, è stata risolta.
Al passo successivo, il passo `i`=3, l'elemento da inserire è 11. Poiché 11 è minore di tutti gli elementi alla sua sinistra, il procedimento a ritroso per l'individuazione della "giusta" posizione di 11 produce la traslazione a destra dei primi 3 elementi e l'inserimento di 11 nella prima posizione dell'array, quella di indice 0. Il nuovo valore dell'array A risulta

11	27	36	41	28	13	52	31
0	1	2	**i =3**	4	5	6	n-1=7

Si noti che l'istanza 4 del problema, cioè l'ordinamento dei primi 4 elementi dell'array A, è stata risolta.
Il passo successivo, `i`=4, ha lo scopo di inserire 28 nella giusta posizione in modo che la porzione 0..4 sia ordinata.
E così via, fino all'ultimo passo, `i`=7, in cui si determina la "giusta" posizione dell'elemento 31 in modo che la porzione 0..7, cioè tutto l'array A, risulti ordinata.

11	13	27	28	36	41	52	31
0	1	2	3	4	5	6	**i=n-1=7**

La struttura dell'algoritmo è costituita da un ciclo `for` per i passi e da un ciclo `while` innestato per il procedimento a ritroso di individuazione della "giusta" posizione.
Questa è l'implementazione in C dell'AOINS, per un array di tipo `char`:

```
void ord_inser(char array[],int n)
 {
  int i,j;
  char el_da_ins;
  for (i=1;i<n;i++)
  {
     el_da_ins = array[i];
     j = i-1;
     while(j >= 0  &&  el_da_ins < array[j])
         {
           array[j+1] = array[j];
           j--;
         }
     array[j+1] = el_da_ins;
  }
 }
```

Un breve commento al codice. Il `for` su `i` da 1 a n-1 gestisce gli n-1 passi dell'AOINS. Lo scopo del passo `i` è inserire nella giusta posizione l'elemento di indice `i`, che è salvato nella variabile **`el_da_ins`**, in modo che la porzione 0..`i` risulti ordinata.

Il processo iterativo a ritroso per l'individuazione della giusta posizione e il successivo inserimento di **el_da_ins** è realizzato mediante il ciclo `while`, il cui predicato di continuazione è **j >= 0 && el_da_ins < array[j]**.
Si noti che l'uscita dal ciclo `while` significa che è stata trovata la giusta posizione, quella di indice **j+1**, e che gli elementi maggiori di **el_da_ins** sono stati spostati a destra di tale posizione. L'ultima operazione del ciclo `for` consiste proprio nell'inserimento di **el_da_ins** nella posizione di indice **j+1**.

Analizziamo la complessità dell'AOINS. La complessità di spazio è n, perché l'algoritmo opera completamente "in place".
Passiamo alla complessità di tempo. Negli algoritmi di ordinamento si considerano come operazioni dominanti l'operazione di confronto tra due elementi dell'array e l'operazione di scambio tra 2 elementi dell'array, o meglio, nel nostro caso, di spostamento a destra di un elemento dell'array.
L'operazione di confronto tra due elementi dell'array è quella che appare nel predicato del `while`.
Il ciclo `while` viene attivato n-1 volte, poiché il `while` è interno al ciclo `for`. Il numero di iterazioni di ogni attivazione del `while` non è costante, perché dipende sia dalla lunghezza della porzione dove si deve trovare la giusta posizione dell'elemento da inserire, sia dal valore degli elementi di tale porzione.
Dobbiamo quindi fare un'analisi di complessità di caso peggiore. Il caso peggiore, cioè il massimo numero possibile di confronti, si ha quando si esce dal ciclo `while` sempre perché `j < 0` , ovvero quando la giusta posizione è sempre la prima posizione (`j = 0`) dell'array. Per inciso, questo accade quando l'array in input è ordinato nel senso decrescente.
Quindi, al primo passo, `i`=1, il massimo numero possibile di confronti nel `while` è 1; al secondo passo, `i`=2, il massimo numero possibile di confronti nel `while` è 2; e così via, fino all'ultimo passo, `i`=n-1, in cui il massimo numero possibile di confronti è n-1.
Per ottenere il numero totale di confronti bisogna sommare i numeri dei confronti di ogni passo, cioè (1 + 2 +…+ (n-2) + (n-1) . Ma tale numero è la somma dei primi (n-1) numeri naturali, che per la formula di Gauss è $\frac{n\cdot(n-1)}{2}$. Si tratta di una complessità di caso peggiore.
Per quanto concerne il numero di scambi, o meglio il numero di shift a destra, basta osservare che si ha uno spostamento a destra ogni volta che è vero il predicato del `while`. Pertanto, nel caso peggiore, il numero di shift a destra è uguale al numero dei confronti.
In conclusione, la complessità di tempo dell'AOINS è

$T(n)=\frac{n\cdot(n-1)}{2}=\frac{1}{2}(n^2-n)=O(n^2)$ confronti, al più

$T(n)=\frac{n\cdot(n-1)}{2}=\frac{1}{2}(n^2-n)=O(n^2)$ scambi (shift), al più.

Infine, è importante notare che il caso migliore si ha quando l'array A in input è già ordinato. In questa situazione l'AOINS effettua solo (n-1) confronti (uno per ogni iterazione del `for`, in quanto il blocco del `while` non viene mai eseguito) e 0 scambi/shift. Gergalmente, si dice che l'AOINS è in grado di "accorgersi" con (n-1) confronti del fatto che un array sia già ordinato.