Programmazione avanzata ed elementi di Ingegneria del Software

Insertion Sort

Insertion Sort

Consideriamo un vettore di interi e dividiamolo in

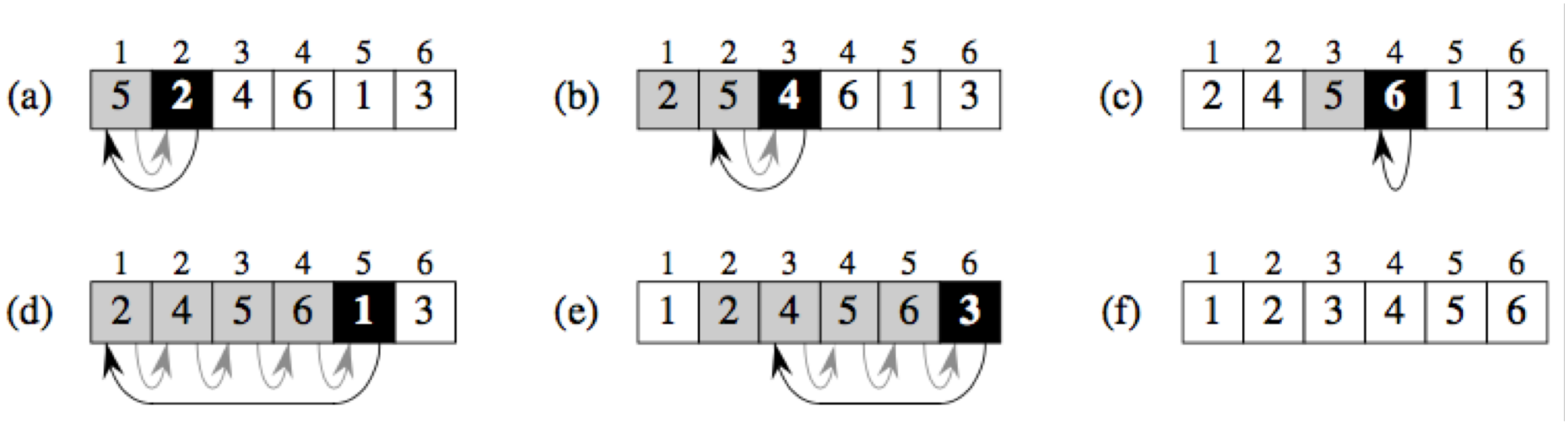
- parte **ordinata** (all'inizio un solo elemento) e
- parte **non ordinata**

6 5 3 1 8 7 2 4

Ad ogni iterazione, prendo un elemento dalla parte non ordinata e lo inserisco nella parte ordinata

- meno efficiente di algoritmi più avanzati
- semplice da implementare
- efficiente per vettori quasi ordinati.

Insertion Sort



Ad ogni iterazione, il rettangolo nero contiene il valore estratto,
che viene confrontato con la porzione di vettore alla sua sinistra, già ordinata.

Il valore estratto viene spostato in modo da mantenere ordinata la porzione di vettore a sinistra, shiftando di conseguenza gli elementi.

Insertion Sort

```
void InsertionSort(int* v, int n) {
    int el, j;
    for (int i = 1; i < n; i++) {
        el = v[i];
        // la porzione di vettore v[0] ... v[i-1]
        // è già ordinata.
        // vi inserisco el, shiftando gli altri valori
        j = i - 1;

        while (j >= 0 && v[j] > el) {
            // NB Short Circuit Evaluation
            v[j + 1] = v[j]; // SHIFT
            j--;
        }
        v[j + 1] = el;
    }
}
```

Insertion Sort

```
void InsertionSort(int* v, int n) {
    int el, j;
    for (int i = 1; i < n; i++) {
        el = v[i];
        j = i - 1;

        while (j >= 0 && v[j] > el) {

            v[j + 1] = v[j]; // SHIFT
            j--;
        }
        v[j + 1] = el;
    }
}
```

Numero esecuzioni

n

n-1

n-1

$\sum_{i=1}^{n-1} t_i$

$\sum_{i=1}^{n-1}(t_i - 1)$

$\sum_{i=1}^{n-1}(t_i - 1)$

n-1

Insertion Sort

```
void InsertionSort(int* v, int n) {
    int el, j;
    for (int i = 1; i < n; i++) {
        el = v[i];
        j = i - 1;

        while (j >= 0 && v[j] > el) {

            v[j + 1] = v[j]; // SHIFT
            j--;
        }
        v[j + 1] = el;
    }
}
```

Numero esecuzioni

n

n-1

n-1

$\sum_{i=1}^{n-1} t_i$

$\sum_{i=1}^{n-1}(t_i-1)$

$\sum_{i=1}^{n-1}(t_i-1)$

n-1

Il tempo di esecuzione dipende dal *valore* dell'input, non solo dalla dimensione.

$$f(n) = n + 2(n-1) + \sum_{i=1}^{n-1} t_i + 2\sum_{i=1}^{n-1}(t_i-1) + (n-1) = 2n - 1 + 3\sum_{i=1}^{n-1} t_i$$

Insertion Sort

```
void InsertionSort(int* v, int n) {
    int el, j;
    for (int i = 1; i < n; i++) {
        el = v[i];
        j = i - 1;

        while (j >= 0 && v[j] > el) {

            v[j + 1] = v[j]; // SHIFT
            j--;
        }
        v[j + 1] = el;
    }
}
```

Caso migliore: vettore già ordinato (crescente)

`el>=v[j] per ogni i quando j=i-1`

`test ciclo while sempre FALSE`

$t_i = 1$

$$f(n) = 2n - 1 + 3\sum_{i=1}^{n-1} t_i = 5n - 4$$

$O(n)$

Insertion Sort

```
void InsertionSort(int* v, int n) {
    int el, j;
    for (int i = 1; i < n; i++) {
        el = v[i];
        j = i - 1;

        while (j >= 0 && v[j] > el) {

            v[j + 1] = v[j]; // SHIFT
            j--;
        }
        v[j + 1] = el;
    }
}
```

Caso peggiore: vettore ordinato in senso inverso (decrescente)

el<v[j]

ciclo while si interrompe quando j<0

$t_i = i+1$

$$f(n) = 2n - 1 + 3\sum_{i=1}^{n-1} t_i = \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

$O(n^2)$

Considerazioni su complessità

Caso peggiore:
Limite superiore per qualsiasi input (abbiamo la garanzia che non verrà superato).
Il caso peggiore si verifica spesso

Considerazioni su complessità

Caso medio:
Molto difficile da calcolare (a volte impossibile).
Il caso medio spesso è brutto quanto il caso peggiore.

Es. quando l'algoritmo deve determinare dove inserire l'elemento a[i] nel sottoarray a[0… i-1], in media metà degli elementi del sottoarray saranno più grandi di a[i]. Ci aspettiamo $t_i \cong \frac{i}{2}$ e quindi $O(n^2)$

Esercizi

- Illustrate l'operazione di InsertionSort su un array dato
- Calcolare la complessità f(n) utilizzando come input un array dato. Verificare la correttezza di f(n) contando le istruzioni durante l'esecuzione.
- Ipotizzare che lo spostamento di un valore dell'array (dunque `el=v[i], v[j+1]=v[j]`, ecc) abbia costo doppio rispetto alle altre istruzioni elementari. L'ipotesi ha senso perché gli elementi da spostare potrebbero essere strutture. Calcolare f(n).

Esercizi

- Modificare InsertionSort in modo che disponga gli elementi in ordine non crescente
- Modificare InsertionSort in modo che ammetta una certa tolleranza nell'ordinamento. Calcolare f(n). L'algoritmo modificato è più rapido di quello originale? Che problemi da?

- Implementare insertion sort in modo ricorsivo e calcolare complessità computazionale.

Esercizi

Algoritmo SelectionSort:

- si trovi l'elemento più piccolo di **a**, e si scambi con **a**[0]
- si trovi l'elementi più piccolo di **a**[1… n-1], e si scambi con **a**[1]
- …
- si trovi l'elementi più piccolo di **a**[j… n-1], e si scambi con **a**[j]

Implementare l'algoritmo.
Trovare complessità computazionale.
Svolgere gli esercizi proposti per InsertionSort