Algoritmi Selection Sort e Insertion Sort "in-place"

Fabrizio d'Amore - Sapienza DIAG

aprile 2016

Selection Sort in-place

```
void selectionSort(int a[], int n) {
  int i, j, k, tmp;
  for(i = 0; i < n-1; i++) {
    tmp = a[i]; min = tmp; k = i;
    for(j = i+1; j < n; j++)
      if(a[j] < min) {
        min = a[j];
        k = j;
      }
    a[i] = min; a[k] = tmp;
}
```

Selection Sort in-place

```
void selectionSort(int a[], int n) {
  int i, j, k, tmp;
  for(i = 0; i < n-1; i++) {
    tmp = a[i]; min = tmp; k = i;
    for(j = i+1; j < n; j++)
      if(a[j] < min) {
        min = a[j];
        k = j;
      }
    a[i] = min; a[k] = tmp;
}
```

op dominante [n(n-1)/2 volte]

proprietà Selection Sort

- numero (asintotico) di passi dipendente dalla dimensione n dell'input, ma non dalla permutazione degli n elementi
  - costo di caso peggiore, medio e migliore: $\Theta(n^2)$
- non molto usato
- in-place (n.ro costante di variabili aggiuntive: 4)
- caso peggiore: $\Theta(n^2)$ confronti, $\Theta(n)$ scambi su array, $\Theta(n^2)$ assegnazioni (in cache)

Insertion Sort in-place

```
void insertionSort(int a[], int n) {
  int i, j, tmp;
  for(i = 1; i < n; i++) {
    tmp = a[i];
    for(j = i-1; j >= 0 && a[j] > tmp; j--)
      a[j+1] = a[j];
    a[j+1] = tmp;
  }
}
```

Insertion Sort in-place

```
void insertionSort(int a[], int n) {
  int i, j, tmp;
  for(i = 1; i < n; i++) {
    tmp = a[i];
    for(j = i-1; j >= 0 && a[j] > tmp; j--)
      a[j+1] = a[j];
    a[j+1] = tmp;
  }
}
```

op dominante [n(n-1)/2 volte]

proprietà Insertion Sort

- numero (asintotico) di passi dipendente dalla dimensione n dell'input e dalla permutazione degli n elementi
  - costo di caso peggiore (input ordinato al contrario): $\Theta(n^2)$
  - costo di caso migliore (input ordinato): $\Theta(n)$
- molto usato su input "piccoli"
- sfrutta eventuali ordinamenti già presenti nell'input
- in-place (n.ro costante di variabili aggiuntive: 3)
- caso peggiore: $\Theta(n^2)$ confronti, $\Theta(n^2)$ scambi su array