Appunti

Lez 1

Algoritmo: procedura, che descrive tramite una sequenza di passi elementari, come risolvere un problema.

Analizziamo le risorse necessarie per l'esecuzione:

- tempo
- spazio
- correttezza
- fault tolerant
- scalabilità
- mantenibilità

Algoritmi

- InsertionSort - incrementale ~ n^2
- MergeSort - divide et impera ~ $nlogn$

InsertionSort

Si usa una variabile [key] di supporto invece di un array nuovo. Array A[1….n] con A.length → lunghezza.

```
InsertionSort(A)
  n = A.length
  for j=2 to n
    key = A[j]
    i = j-1
    while (i>0) and (A[i]>key)
      A[i+1] = A[i]
      i = i-1
    A[i+1] = key
```

Correttezza

[ciclo for] = A[1….., j-1] ordinato

[ciclo while] = A[1…..i] e A[i+2…j] ordinato, A[i+2…..j] > key

Invariante: proprietà che deve valere prima che inizi il ciclo, deve essere mantenuta durante il ciclo, quindi vale anche quando esco dal ciclo.

Lez 2

Quanto costa l'esecuzione, in questo caso InsertionSort?

modello di calcolo

operazioni elementari → costo costante

dimensione del problema

- ordinamento → #elementi
- alg. su grafi → #archi, #nodi
- prodotto di interi → #bit

```
InsertionSort(A) //c0
  n = A.length  //c1
  for j=2 to n  //c2*n
    key = A[j]  //c3*(n-1)
    i = j-1  //c4*(n-1)
    while (i>0) and (A[i]>key)  //c5*Sum(j=2,n)(tj+1)
      A[i+1] = A[i]  //c6*Sum(j=2,n)tj
      i = i-1  //c7*Sum(j=2,n)tj
    A[i+1] = key  //c8(n-1)
```

$$T^I(n) = c_0 + c_1 + c_2 n + (c_3 + c_4 + c_8)(n-1) + c_5 \sum_{j=2}^{n}(tj+1) + (c_6 + c_7)\sum_{j=2}^{n} tj$$

casi di interesse

- caso migliore → $T^i{}_{min}$
- caso peggiore → $T^i{}_{max}$
- caso medio → $T^i{}_{med}$

Caso migliore

A è già ordinato $tj = 0j \ \ \forall j$

$T^I(n) = c_0 + c_1 + c_2 n + (c_3 + c_4 + c_8)(n-1) + c_5 \sum_{j=2}^{n}(tj+1) + (c_6 + c_7)\sum_{j=2}^{n} tj$

$T^I(n) = c_0 + c_1 + c_2 n + (c_3 + c_4 + c_5 + c_8)(n-1)$ Perchè tj=0

$T^I(n) = an + b \sim n$

Caso peggiore

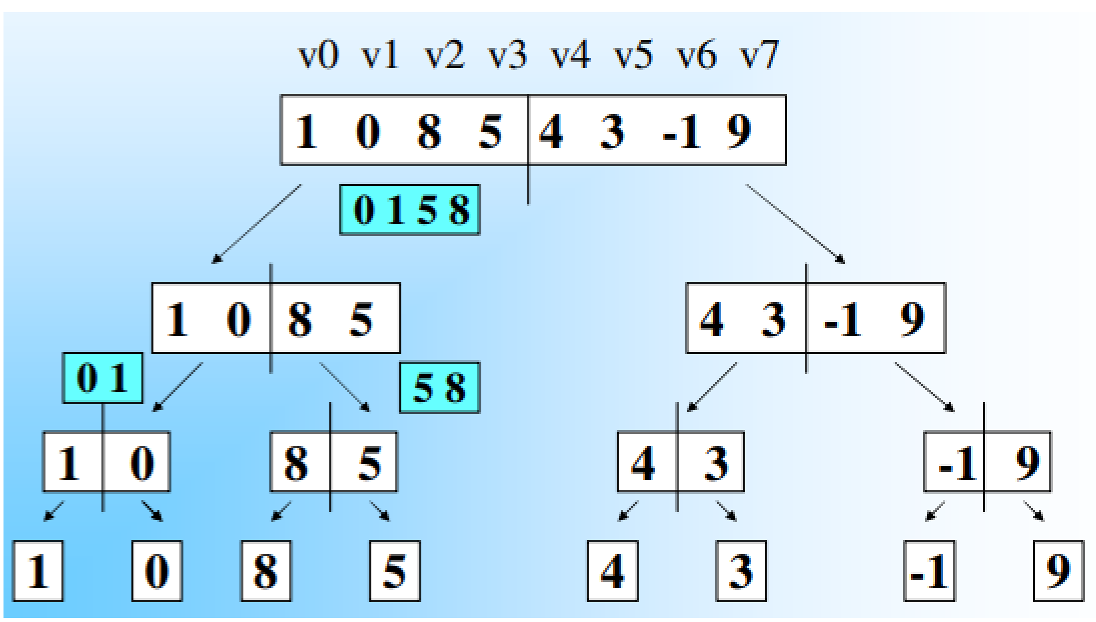
$\sim n^2$

Caso medio

$\sim n^2$

MergeSort

- divide l'array in due parti
- ordina i sottoarray
- fonde i sottoarray ordinati



```
MergeSort(A,p,r)
  if p<r
    q = (p+r)/2
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    Merge(A,p,q,r)

Merge(A,p,q,r)
  n1 = q-p+1
  n2 = r-q
  crea L[1...n1+1]
  crea R[1...n2+1]
  for i = 1 to n1
    L[i] = A[p+i-1]
  for j = 1 to n2
    R[j] = A[q+j]
  L[n1+1] = R[n2+1] = ∞
  i=j=1
  for k = p to r
    if L[i] <= R[j]
      A[k] = L[i]
      i++
    else
      A[k] = R[j]
      j++
```

Induzione su r-p = l

- l=0 → Merge non fa niente
- (l>0) si divide in due l'array

Induzione = per dimostrare che P(n) vale ∀ n

- si dimostra P(0)
- ∀ n P(n) → P(n+1)