RICHIAMI SULL'INDUZIONE

-- Principio di induzione semplice:

P(n) Esempio P(n) = "L'algoritmo A che agisce su una lista di n interi è corretto"

Affinchè P(n) sia vera, devo dimostrare
1)Esiste n0>=0 : P(n0) è vera
2) Dato P(n-1) vero ==> P(n) vero

-- Principio di induzione forte

P(n) Esempio P(n) = "L'algoritmo A che agisce su una lista di n interi è corretto"

Affinchè P(n) sia vera, devo dimostrare
1)Esiste n0>=0 : P(n0) è vera
2) Dato P(m) vero per qualsiasi m<n ==> P(n) vero

APPROCCIO DIVIDE ET IMPERA

Posso sfruttare il principio di induzione forte costruendo soluzioni in questo modo

Passo DIVIDE)
Scompongo il mio problema originale P(n) in tanti sottoproblemi dello stesso
tipo: P(m1), P(m2) , ..., P(mk)   m1+m2 +...+ mk=n

Passo IMPERA)
Trovare le soluzioni dei sottoproblemi con chiamate ricorsive:
-- S1=chimataRicorsiva(P(m1)),
S2=chimataRicorsiva(P(m2)), ...,Sk=chimataRicorsiva(P(mk)),
-- "Combino" le soluzioni S1,S2, ...,S2 per trovare la soluzione S di P(n)

UN CASO CLASSICO DI APPROCCIO DIVIDE ET IMPERA E' IL MERGE-SORT

V = 5 6 1 30 9 11 -3 2 Problema è P(n)= "Ordinare un array di n elementi"

INTUIZIONE:

A ordinato = 1 4 6
B ordinato = 3 8 10

C= che sia composto dagli elementi di A e B messi in maniera

ordinati

C= 1 3 4 6 8 10 (La costruzione di C a partire da due array ordinati, è più facile rispetto alla costruzione di C a a partire da due array NON ordinati)

Allora se riprendo V

V = 5 6 1 30 9 11 -3 2

V = 5 6 1 30 | 9 11 -3 2 dico che A= ordina([5 6 1 30])
B= ordina([9 11 -3 2])

V ordinato = COMBINAZIONE DI A e B

SIGNIFICA CHE per ottenere A e B ho di nuovo dei problemi di ordinamento ma di dimensione più piccola, cioè m1= n/2 e m2=n/2, quindi divido il problema originale in P(m1) e (Pm2)

-- Date le soluzioni A e B dei problemi P(m1) e P(m2) devo combinare le soluzioni per
ottenere P(n)

Da un punto di vista del codice:

```
//V array di interi, n è la lunghezza del array
void mergeSort(int V[], int n) {
    int centro=n/2;
    //PROBLEMA P(m1), cioè gli elementi di V da 0 a centro-1
    mergeSort(V,centro); //ASSUMO che ordino A che corrisponde agli elementi
                  // di V da 0 a centro-1
    mergeSort(&V[centro],n-centro); //ASSUMO che ordino B che corrisponde agli elementi
                  // di V da centro+1 a n
    //COMBINO le due soluzioni
    merge(V,centro,n);

}
```

IL problema chiave è quello di costruire la soluzione di P(n), date le soluzioni di P(m1) e P(m2), cioè in questo caso di fornire l'algoritmo per merge

COME COSTRUIRE MERGE:
A ordinato = 1 4 6
B ordinato = 3 8 10
C, vettore appoggio [0 0 0 0 0 0]
i=0, j=0, k=0

A[i]=1 A[j]=3

if A[i]< A[j]

```
     c[k]=A[i]    C=[1 0 0 0 0 0]
     i=i+1
     k=k+1
else
    c[k]=A[j]
    j=j+1
    k=k+1
i=1, j=0, k=1
A ordinato = 1 4 6
B ordinato = 3 8 10

A[i]=4 A[j]=3

if A[i]< A[j]
     c[k]=A[i]
     i=i+1
     k=k+1
else
    c[k]=A[j] C=[1 3 0 0 0 0]
    j=j+1
    k=k+1
i=1, j=1, k=2


A ordinato = 1 4 6
B ordinato = 3 8 10

A[i]=4 A[j]=8

if A[i]< A[j]
     c[k]=A[i] C=[1 3 4 0 0 0]
     i=i+1
     k=k+1
else
    c[k]=A[j]
    j=j+1
    k=k+1
i=2, j=1, k=3

A ordinato = 1 4 6
B ordinato = 3 8 10

A[i]=6 A[j]=8

if A[i]< A[j]
     c[k]=A[i] C=[1 3 4 6 0 0]
     i=i+1
     k=k+1
else
    c[k]=A[j]
    j=j+1
    k=k+1
```

i=3, j=1, k=4

A ordinato = 1 4 6
B ordinato = 3 8 10
DA QUI SI VEDE CHE QUESTO CONFRONTO LO RIPETO FINO A CHE UNO DEI DUE
ARRAY NON TERMINA

QUINDI CONSIDERO l'ARRAY NON TERMINATO (IN QUESTO CASO B) e LO COPIO IN C

```
for i=j to LEN(B)-1
     C[k]=B[i];
     k=k+1
```

C=[1 3 4 6 8 10]

A questo punto ricopio C in V

V=[1 3 4 6 8 10]

Vedi codice mergeSort.c