Lezione 8

Sommario della Lezione

Sommario della Lezione

- Altri Esempi di Applicazione della Tecnica Divide et Impera

Calcolo delle inversioni

Dato un array di numeri `a=a[0]a[1]...a[n-1]`, diremo che una coppia di indici `(i,j)` è un'**inversione**, se `i<j` e `a[i]>a[j]`.

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7, allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,
allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.
Anche (1,8) è un'inversione in quanto 1<8 ma a[1]=5>a[8]=3.

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,
allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.
Anche (1,8) è un'inversione in quanto 1<8 ma a[1]=5>a[8]=3.
Anche (4,9) è un'inversione in quanto 4<9 ma a[4]=10>a[9]=7.

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,
allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.
Anche (1,8) è un'inversione in quanto 1<8 ma a[1]=5>a[8]=3.
Anche (4,9) è un'inversione in quanto 4<9 ma a[4]=10>a[9]=7.

Problema:

Input: sequenza a=a[0]a[1]...a[n-1] di numeri

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,
allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.
Anche (1,8) è un'inversione in quanto 1<8 ma a[1]=5>a[8]=3.
Anche (4,9) è un'inversione in quanto 4<9 ma a[4]=10>a[9]=7.

Problema:

Input: sequenza a=a[0]a[1]...a[n-1] di numeri
Output: numero di inversioni in a

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,
allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.
Anche (1,8) è un'inversione in quanto 1<8 ma a[1]=5>a[8]=3.
Anche (4,9) è un'inversione in quanto 4<9 ma a[4]=10>a[9]=7.

Problema:

Input: sequenza a=a[0]a[1]...a[n-1] di numeri
Output: numero di inversioni in a

Una prima soluzione sarebbe quella di controllare tutte le possibili coppie di indici (i,j) con i<j, e verificare se a[i]>a[j].

Calcolo delle inversioni

Dato un array di numeri a=a[0]a[1]...a[n-1], diremo che una coppia di indici (i,j) è un'**inversione**, se i<j e a[i]>a[j].

Ad esempio, sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7,
allora (1,2) è un'inversione in quanto 1<2 ma a[1]=5>a[2]=4.
Anche (1,8) è un'inversione in quanto 1<8 ma a[1]=5>a[8]=3.
Anche (4,9) è un'inversione in quanto 4<9 ma a[4]=10>a[9]=7.

Problema:

Input: sequenza a=a[0]a[1]...a[n-1] di numeri
Output: numero di inversioni in a

Una prima soluzione sarebbe quella di controllare tutte le possibili coppie di indici (i,j) con i<j, e verificare se a[i]>a[j]. Ovviamente, ciò darebbe origine ad un algoritmo di complessità $\Theta(n^2)$.

Usiamo Divide et Impera

Usiamo Divide et Impera

1. Se la sequenza a ha un solo elemento, restituisci 0 (essa non ha inversioni)

Usiamo Divide et Impera

1. Se la sequenza `a` ha un solo elemento, restituisci 0 (essa non ha inversioni)
2. Altrimenti, dividi la sequenza in due sottosequenze `l` e `r`

Usiamo Divide et Impera

1. Se la sequenza `a` ha un solo elemento, restituisci 0 (essa non ha inversioni)
2. Altrimenti, dividi la sequenza in due sottosequenze `l` e `r`
3. Calcola (ricorsivamente) il numero di inversioni in `l`

Usiamo Divide et Impera

1. Se la sequenza `a` ha un solo elemento, restituisci 0 (essa non ha inversioni)
2. Altrimenti, dividi la sequenza in due sottosequenze `l` e `r`
3. Calcola (ricorsivamente) il numero di inversioni in `l`
4. Calcola (ricorsivamente) il numero di inversioni in `r`

Usiamo Divide et Impera

1. Se la sequenza `a` ha un solo elemento, restituisci 0 (essa non ha inversioni)
2. Altrimenti, dividi la sequenza in due sottosequenze `l` e `r`
3. Calcola (ricorsivamente) il numero di inversioni in `l`
4. Calcola (ricorsivamente) il numero di inversioni in `r`
5. Calcola il numero di inversioni `(i,j)`, con `a[i]` elemento di `l` e `a[j]` elemento di `r`

Usiamo Divide et Impera

1. Se la sequenza `a` ha un solo elemento, restituisci 0 (essa non ha inversioni)
2. Altrimenti, dividi la sequenza in due sottosequenze `l` e `r`
3. Calcola (ricorsivamente) il numero di inversioni in `l`
4. Calcola (ricorsivamente) il numero di inversioni in `r`
5. Calcola il numero di inversioni `(i,j)`, con `a[i]` elemento di `l` e `a[j]` elemento di `r`
6. Restituisci la somma dei valori calcolati al passo 3. 4. e 5.

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6, 8-7,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6, 8-7,10-2,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6, 8-7,10-2, 10-3, 10-6,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6, 8-7,10-2, 10-3, 10-6, 10-7,

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6, 8-7,10-2, 10-3, 10-6, 10-7, 10-9.

Esempio

Sia a=a[0]a[1]...a[9]= 1 5 4 8 10 2 6 9 3 7

1. Dividiamo a in l= 1 5 4 8 10 e r=2 6 9 3 7.
2. Calcoliamo le inversioni in l=1 5 4 8 10:
   sono solo la coppia di elementi 5-4
3. Calcoliamo le inversioni in in r=2 6 9 3 7:
   sono le coppie di elementi 6-3, 9-3, 9-7.
4. Calcoliamo le inversioni tra elementi di l= 1 5 4 8 10 ed elementi di r=2 6 9 3 7, esse sono:
   4-2, 4-3, 5-2, 5-3, 8-2, 8-3, 8-6, 8-7,10-2, 10-3, 10-6, 10-7, 10-9.
5. In totale, avremmo quindi 1+3+13=17 inversioni in a.

Come calcolare il numero di inversioni del passo 5.,

Come calcolare il numero di inversioni del passo 5., cioè:
calcolare il numero di inversioni `(i,j)`, con `a[i]` elemento di `l` e `a[j]` elemento di `r`.

Come calcolare il numero di inversioni del passo 5., cioè:
calcolare il numero di inversioni `(i,j)`, con `a[i]` elemento di `l` e `a[j]` elemento di `r`.

Primo tentativo: verificare **per ogni** coppia di elementi, il primo in `l` ed il secondo in `r`, se essa è o meno un'inversione.

Come calcolare il numero di inversioni del passo 5., cioè:
calcolare il numero di inversioni `(i,j)`, con `a[i]` elemento di `l` e `a[j]` elemento di `r`.

Primo tentativo: verificare **per ogni** coppia di elementi, il primo in `l` ed il secondo in `r`, se essa è o meno un'inversione.

Non va bene in quanto porterebbe ad un algoritmo di complessità $\Theta(n^2)$

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**.

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]`

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ?

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)
3. se `l[i] > r[j]` allora `r[j]` è invertito con ogni elemento a destra di `l[i]`

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)
3. se `l[i] > r[j]` allora `r[j]` è invertito con ogni elemento a destra di `l[i]` (perchè ? Perchè a destra di `l[i]` ci sono elementi **ancora** più grandi)

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)
3. se `l[i] > r[j]` allora `r[j]` è invertito con ogni elemento a destra di `l[i]` (perchè ? Perchè a destra di `l[i]` ci sono elementi **ancora** più grandi)
4. inserisci il minimo tra `l[i]` e `r[j]` in una sequenza ordinata `c`

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)
3. se `l[i] > r[j]` allora `r[j]` è invertito con ogni elemento a destra di `l[i]` (perchè ? Perchè a destra di `l[i]` ci sono elementi **ancora** più grandi)
4. inserisci il minimo tra `l[i]` e `r[j]` in una sequenza ordinata `c` ed itera (come in MergeSort.

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)
3. se `l[i] > r[j]` allora `r[j]` è invertito con ogni elemento a destra di `l[i]` (perchè ? Perchè a destra di `l[i]` ci sono elementi **ancora** più grandi)
4. inserisci il minimo tra `l[i]` e `r[j]` in una sequenza ordinata `c` ed itera (come in MergeSort. Perchè ?

Però...

Il problema diventerebbe più semplice se `l` ed `r` fossero **ordinate**. Infatti:

1. Esamina `l` ed `r` da sinistra a destra confrontando `l[i] con r[j]`, per `i` e `j` crescenti
2. se `l[i] < r[j]` allora `l[i]` non è invertito con nessun elemento a destra di `r[j]` (perchè ? Perchè a destra di `r[j]` ci sono elementi **ancora** più grandi)
3. se `l[i] > r[j]` allora `r[j]` è invertito con ogni elemento a destra di `l[i]` (perchè ? Perchè a destra di `l[i]` ci sono elementi **ancora** più grandi)
4. inserisci il minimo tra `l[i]` e `r[j]` in una sequenza ordinata `c` ed itera (come in MergeSort. Perchè ? Perchè occorre **preservare** l'ordinamento, se vogliamo usare questa tecnica)

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

```
l=1 4 5 8 10
```

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9`.

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`,

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato 4 inversioni.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

Adesso confrontiamo `4` con `3`.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

Adesso confrontiamo `4` con `3`. Poichè `4>3` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `3`,

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

Adesso confrontiamo `4` con `3`. Poichè `4>3` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `3`, ovvero abbiamo trovato altre `4` inversioni.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

Adesso confrontiamo `4` con `3`. Poichè `4>3` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `3`, ovvero abbiamo trovato altre `4` inversioni. Metteremo `3` in `c` e proseguiamo.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

Adesso confrontiamo `4` con `3`. Poichè `4>3` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `3`, ovvero abbiamo trovato altre `4` inversioni. Metteremo `3` in `c` e proseguiamo.

Adesso confrontiamo `4` con `6`.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando 1 con 2 scopriamo che 1 non è invertito con alcun altro elemento di `r`. Metteremo 1 in `c` e proseguiamo.

Confrontiamo ora 4 con 2. Poichè 4>2 questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con 2, ovvero abbiamo trovato 4 inversioni. Metteremo 2 in `c` e proseguiamo.

Adesso confrontiamo 4 con 3. Poichè 4>3 questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con 3, ovvero abbiamo trovato altre 4 inversioni. Metteremo 3 in `c` e proseguiamo.

Adesso confrontiamo 4 con 6. Poichè 4<6 questo ci dice che 4 non è invertito con alcun altro elemento di `r` a destra di 6.

ritorniamo all'esempio...

Le versioni ordinate di `l` è `r` sono

`l=1 4 5 8 10` `r=2 3 6 7 9.`

Confrontando `1` con `2` scopriamo che `1` non è invertito con alcun altro elemento di `r`. Metteremo `1` in `c` e proseguiamo.

Confrontiamo ora `4` con `2`. Poichè `4>2` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `2`, ovvero abbiamo trovato `4` inversioni. Metteremo `2` in `c` e proseguiamo.

Adesso confrontiamo `4` con `3`. Poichè `4>3` questo ci dice che sicuramente che tutti i **rimanenti** elementi di `l` sono invertiti con `3`, ovvero abbiamo trovato altre `4` inversioni. Metteremo `3` in `c` e proseguiamo.

Adesso confrontiamo `4` con `6`. Poichè `4<6` questo ci dice che `4` non è invertito con alcun altro elemento di `r` a destra di `6`. Metteremo `4` in `c` e proseguiamo...

Possibile pseudocodice

```
ContaInversioni(a,i,j)
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
RETURN 0
 }
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
RETURN 0
 } ELSE {
   m=(i+j)/2
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
RETURN 0
 } ELSE {
   m=(i+j)/2
   c1= ContaInversioni(a,i,m)
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
RETURN 0
 } ELSE {
   m=(i+j)/2
   c1= ContaInversioni(a,i,m)
   c2= ContaInversioni(a,m+1,j)
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
RETURN 0
 } ELSE {
   m=(i+j)/2
   c1= ContaInversioni(a,i,m)
   c2= ContaInversioni(a,m+1,j)
   c3= ContaInversioniMerge(a,i,m,j)
```

Possibile pseudocodice

```
ContaInversioni(a,i,j)
IF(i>=j) {
RETURN 0
 } ELSE {
   m=(i+j)/2
   c1= ContaInversioni(a,i,m)
   c2= ContaInversioni(a,m+1,j)
   c3= ContaInversioniMerge(a,i,m,j)
   RETURN c1+c2+c3
}
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
    IF(a[i1]<=a[i2]){
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
     b[k]=a[i2]
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
     b[k]=a[i2]
     i2=i2+1
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
     b[k]=a[i2]
     i2=i2+1
     k=k+1 }
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
     b[k]=a[i2]
     i2=i2+1
     k=k+1 }
FOR(k=1; k<=j-i+1;k=k+1){
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
     b[k]=a[i2]
     i2=i2+1
     k=k+1 }
FOR(k=1; k<=j-i+1;k=k+1){
     a[i+k-1]=b[k]
```

Pseudocodice per ContaInversioniMerge(a,i,m,j)

```
ContaInversioniMerge(a,i,m,j)
c=0 k=1 i1=i i2=m+1
WHILE ((i1<=m) && (i2<=j)) {
     IF(a[i1]<=a[i2]){
       b[k]=a[i1]
       i1=i1+1
     } ELSE {
       c=c+(m+1-i1)
       b[k]=a[i2]
       i2=i2+1 }
     k=k+1 }
WHILE (i1<=m){
     b[k]=a[i1]
     i1=i1+1
     k=k+1 }
WHILE (i2<=j){
     b[k]=a[i2]
     i2=i2+1
     k=k+1 }
FOR(k=1; k<=j-i+1;k=k+1){
     a[i+k-1]=b[k] RETURN c
```

Analisi

Analisi

Per contare le inversioni in `a=a[0]a[1]...a[n-1]` chiameremo quindi l'algoritmo `ContaInversioni(a,0,n-1)`

Analisi

Per contare le inversioni in `a=a[0]a[1]...a[n-1]` chiameremo quindi l'algoritmo `ContaInversioni(a,0,n-1)`

`ContaInversioniMerge(a, 0, m, n-1)` richiede tempo dn, per qualche costante d,

Analisi

Per contare le inversioni in `a=a[0]a[1]...a[n-1]` chiameremo quindi l'algoritmo `ContaInversioni(a,0,n-1)`

`ContaInversioniMerge(a, 0, m, n-1)` richiede tempo dn, per qualche costante d, quindi l'equazione di ricorrenza per la complessità $T(n)$ di `ContaInversioni(a,0,n-1)` sarà :

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ \end{cases}$$

Analisi

Per contare le inversioni in `a=a[0]a[1]...a[n-1]` chiameremo quindi l'algoritmo `ContaInversioni(a,0,n-1)`

`ContaInversioniMerge(a, 0, m, n-1)` richiede tempo dn, per qualche costante d, quindi l'equazione di ricorrenza per la complessità $T(n)$ di `ContaInversioni(a,0,n-1)` sarà :

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ 2T(n/2) + dn & \text{altrimenti.} \end{cases}$$

Analisi

Per contare le inversioni in `a=a[0]a[1]...a[n-1]` chiameremo quindi l'algoritmo `ContaInversioni(a,0,n-1)`

`ContaInversioniMerge(a, 0, m, n-1)` richiede tempo dn, per qualche costante d, quindi l'equazione di ricorrenza per la complessità $T(n)$ di `ContaInversioni(a,0,n-1)` sarà :

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ 2T(n/2) + dn & \text{altrimenti.} \end{cases}$$

Dal risultati precedenti sappiamo che tale equazione di ricorrenza ha soluzione $T(n) = O(n \log n)$

Analisi

Per contare le inversioni in `a=a[0]a[1]...a[n-1]` chiameremo quindi l'algoritmo `ContaInversioni(a,0,n-1)`

`ContaInversioniMerge(a, 0, m, n-1)` richiede tempo dn, per qualche costante d, quindi l'equazione di ricorrenza per la complessità $T(n)$ di `ContaInversioni(a,0,n-1)` sarà :

$$T(n) = \begin{cases} d & \text{se } n = 1 \\ 2T(n/2) + dn & \text{altrimenti.} \end{cases}$$

Dal risultati precedenti sappiamo che tale equazione di ricorrenza ha soluzione $T(n) = O(n \log n)$ migliorando l'algoritmo semplice che aveva complessità $\Theta(n^2)$.

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente e poi crescente

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente e poi crescente ed $a[h]$ è il minimo;

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente e poi crescente ed $a[h]$ è il minimo; se $h = 0$ allora la sequenza è crescente,

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente e poi crescente ed $a[h]$ è il minimo; se $h = 0$ allora la sequenza è crescente, mentre se $h = n-1$ allora è decrescente).

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente e poi crescente ed $a[h]$ è il minimo; se $h = 0$ allora la sequenza è crescente, mentre se $h = n-1$ allora è decrescente).

Da cui il seguente problema algoritmico:

Input: sequenza unimodale `a=a[0]a[1]...a[n-1]` di numeri

Esercizio

Una sequenza $a = a[0] \ldots a[n-1]$ di n interi *distinti* è detta *unimodale* se esiste un indice h tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$ (cioè la sequenza è dapprima decrescente e poi crescente ed $a[h]$ è il minimo; se $h = 0$ allora la sequenza è crescente, mentre se $h = n-1$ allora è decrescente).

Da cui il seguente problema algoritmico:

Input: sequenza unimodale `a=a[0]a[1]...a[n-1]` di numeri
Output: l'indice `h` tale che $a[0] > \ldots > a[h] < \ldots < a[n-1]$

Soluzione attraverso Ricerca Binaria

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato,

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$ e a destra $a[m+1]$:

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$ e a destra $a[m+1]$:

- se **tutte** le diseguaglianze $a[m-1] > a[m] < a[m+1]$ valgono,

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$ e a destra $a[m+1]$:

- se **tutte** le diseguaglianze $a[m-1] > a[m] < a[m+1]$ valgono, allora l'elemento mediano è proprio $a[m]$ ed abbiamo terminato;

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$ e a destra $a[m+1]$:

- se **tutte** le diseguaglianze $a[m-1] > a[m] < a[m+1]$ valgono, allora l'elemento mediano è proprio $a[m]$ ed abbiamo terminato;
- altrimenti, se $a[m-1] > a[m]$, l'elemento che cerchiamo si trova a destra di $a[m]$,

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$ e a destra $a[m+1]$:

- se **tutte** le diseguaglianze $a[m-1] > a[m] < a[m+1]$ valgono, allora l'elemento mediano è proprio $a[m]$ ed abbiamo terminato;
- altrimenti, se $a[m-1] > a[m]$, l'elemento che cerchiamo si trova a destra di $a[m]$,
- altrimenti, l'elemento che cerchiamo si trova a sinistra $a[m]$.

Soluzione attraverso Ricerca Binaria

Prendiamo l'elemento *mediano* $a[m]$ del sottovettore considerato, e consideriamo l'elemento a sinistra $a[m-1]$ e a destra $a[m+1]$:

- se **tutte** le diseguaglianze $a[m-1] > a[m] < a[m+1]$ valgono, allora l'elemento mediano è proprio $a[m]$ ed abbiamo terminato;
- altrimenti, se $a[m-1] > a[m]$, l'elemento che cerchiamo si trova a destra di $a[m]$,
- altrimenti, l'elemento che cerchiamo si trova a sinistra $a[m]$.

Ci sono un paio di casi base: il sottovettore considerato ha 1 o 2 elementi, nel qual caso scegliamo il minimo.

Possibile pseudocodice

UNIMODALE(a, i, j)

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
       }
    }
6. m = ⌊(i + j)/2⌋
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
6. m = ⌊(i + j)/2⌋
7. IF ((a[m − 1] > a[m])&&(a[m + 1] > a[m])){
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
6. m = ⌊(i + j)/2⌋
7. IF ((a[m − 1] > a[m])&&(a[m + 1] > a[m])){
8. RETURN a[m]
    }
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
6. m = ⌊(i + j)/2⌋
7. IF ((a[m − 1] > a[m])&&(a[m + 1] > a[m])){
8. RETURN a[m]
    }
9. IF (a[m − 1] > a[m]) {
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
6. m = ⌊(i + j)/2⌋
7. IF ((a[m − 1] > a[m])&&(a[m + 1] > a[m])){
8. RETURN a[m]
    }
9. IF (a[m − 1] > a[m]) {
10.    RETURN UNIMODALE(a, m + 1, j)
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
6. m = ⌊(i + j)/2⌋
7. IF ((a[m − 1] > a[m])&&(a[m + 1] > a[m])){
8. RETURN a[m]
    }
9. IF (a[m − 1] > a[m]) {
10.    RETURN UNIMODALE(a, m + 1, j)
11.    } ELSE {
```

Possibile pseudocodice

```
UNIMODALE(a, i, j)
1. IF (i == j){
2.    RETURN a[i]
3.    } ELSE {
4.       IF (j == i + 1) {
5.       RETURN min(a[i], a[j])
        }
    }
6. m = ⌊(i + j)/2⌋
7. IF ((a[m − 1] > a[m])&&(a[m + 1] > a[m])){
8. RETURN a[m]
    }
9. IF (a[m − 1] > a[m]) {
10.    RETURN UNIMODALE(a, m + 1, j)
11.    } ELSE {
12    RETURN UNIMODALE(a, i, m − 1)
13    }
```

La equazione di ricorrenza che descrive la complessità $T(n)$ dell'algoritmo $\texttt{UNIMODALE}(a, 0, n-1)$ è sempre la solita

$$T(n) = \begin{cases} c_0 & \text{se } n \leq 1 \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

per costanti c_0 e c opportune.

La equazione di ricorrenza che descrive la complessità $T(n)$ dell'algoritmo $\texttt{UNIMODALE}(a, 0, n-1)$ è sempre la solita

$$T(n) = \begin{cases} c_0 & \text{se } n \leq 1 \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

per costanti c_0 e c opportune. La soluzione dell'equazione di ricorrenza è $T(n) = O(\log n)$.

Un altro esercizio

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1] \dots a[n-1]$,

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1] \ldots a[n-1]$, un **massimo relativo** di a è un elemento di a che è *maggiore o uguale* dei suoi adiacenti

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1]\ldots a[n-1]$, un **massimo relativo** di a è un elemento di a che è *maggiore o uguale* dei suoi adiacenti (o di un **solo** adiacente, **se** stiamo considerando $a[0]$ o $a[n-1]$).

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1]\ldots a[n-1]$, un **massimo relativo** di a è un elemento di a che è *maggiore o uguale* dei suoi adiacenti (o di un **solo** adiacente, **se** stiamo considerando $a[0]$ o $a[n-1]$).

Esempio:
la sequenza $a = [5, 10, 20, 15]$ ha un solo massimo relativo, che è 20

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1]\ldots a[n-1]$, un **massimo relativo** di a è un elemento di a che è *maggiore o uguale* dei suoi adiacenti (o di un **solo** adiacente, **se** stiamo considerando $a[0]$ o $a[n-1]$).

Esempio:
la sequenza $a = [5, 10, 20, 15]$ ha un solo massimo relativo, che è 20

la sequenza $a = [10, 20, 15, 2, 23, 90, 67, 80]$ ha tre massimi relativi: 20, 90 e 80.

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1]\ldots a[n-1]$, un **massimo relativo** di a è un elemento di a che è *maggiore o uguale* dei suoi adiacenti (o di un **solo** adiacente, **se** stiamo considerando $a[0]$ o $a[n-1]$).

Esempio:
la sequenza $a = [5, 10, 20, 15]$ ha un solo massimo relativo, che è 20

la sequenza $a = [10, 20, 15, 2, 23, 90, 67, 80]$ ha tre massimi relativi: 20, 90 e 80.

Vogliamo risolvere il seguente problema algoritmico.
Input: array $a = a[0]a[1]\ldots a[n-1]$

Un altro esercizio

Dato una sequenza di numeri $a = a[0]a[1]\ldots a[n-1]$, un **massimo relativo** di a è un elemento di a che è *maggiore o uguale* dei suoi adiacenti (o di un **solo** adiacente, **se** stiamo considerando $a[0]$ o $a[n-1]$).

Esempio:
la sequenza $a = [5, 10, 20, 15]$ ha un solo massimo relativo, che è 20

la sequenza $a = [10, 20, 15, 2, 23, 90, 67, 80]$ ha tre massimi relativi: 20, 90 e 80.

Vogliamo risolvere il seguente problema algoritmico.
Input: array $a = a[0]a[1]\ldots a[n-1]$
Output: un (qualsivoglia) massimo relativo di a.

Soluzione semplice

Soluzione semplice

Scorri *a* da sinistra a destra,

Soluzione semplice

Scorri *a* da sinistra a destra, e ci fermiamo appena si trova un elemento di *a* che è *maggiore o uguale* dei suoi adiacenti (che sicuramente esiste).

Soluzione semplice

Scorri a da sinistra a destra, e ci fermiamo appena si trova un elemento di a che è *maggiore o uguale* dei suoi adiacenti (che sicuramente esiste).

L'algoritmo avrebbe complessità $\Theta(n)$ nel caso peggiore.

Soluzione più efficiente basata su Ricerca Binaria

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di a con i suoi adiacenti.

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di *a* con i suoi adiacenti. Se tale elemento di mezzo **non è** inferiore a nessuno dei suoi adiacenti,

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di *a* con i suoi adiacenti. Se tale elemento di mezzo **non è** inferiore a nessuno dei suoi adiacenti, allora esso è un massimo relativo e lo ritorniamo.

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di *a* con i suoi adiacenti. Se tale elemento di mezzo **non è** inferiore a nessuno dei suoi adiacenti, allora esso è un massimo relativo e lo ritorniamo.

Se l'elemento di mezzo è inferiore al suo adiacente "sinistro",

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di *a* con i suoi adiacenti. Se tale elemento di mezzo **non è** inferiore a nessuno dei suoi adiacenti, allora esso è un massimo relativo e lo ritorniamo.

Se l'elemento di mezzo è inferiore al suo adiacente "sinistro", allora esiste sicuramente un massimo relativo nella metà di sinistra di *a*.

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di *a* con i suoi adiacenti. Se tale elemento di mezzo **non è** inferiore a nessuno dei suoi adiacenti, allora esso è un massimo relativo e lo ritorniamo.

Se l'elemento di mezzo è inferiore al suo adiacente "sinistro", allora esiste sicuramente un massimo relativo nella metà di sinistra di *a*.

Se l'elemento di mezzo è inferiore al suo adiacente "destro",

Soluzione più efficiente basata su Ricerca Binaria

Confrontiamo l'elemento di mezzo di *a* con i suoi adiacenti. Se tale elemento di mezzo **non è** inferiore a nessuno dei suoi adiacenti, allora esso è un massimo relativo e lo ritorniamo.

Se l'elemento di mezzo è inferiore al suo adiacente "sinistro", allora esiste sicuramente un massimo relativo nella metà di sinistra di *a*.

Se l'elemento di mezzo è inferiore al suo adiacente "destro", allora esiste sicuramente un massimo relativo nella metà di destra di *a*.

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      }
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      } ELSE {
6.        IF (c > 0 && a[c-1] > a[c]) {
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      } ELSE {
6.       IF (c > 0 && a[c-1] > a[c]) {
7.             return TrovaMaxRel(a,i, c-1,n)
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      } ELSE {
6.        IF (c > 0 && a[c-1] > a[c]) {
7.              return TrovaMaxRel(a,i, c-1,n)
8.    } ELSE {
9.              return TrovaMaxRel(a,c+1, j,n)
   }
```

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      } ELSE {
6.        IF (c > 0 && a[c-1] > a[c]) {
7.              return TrovaMaxRel(a,i, c-1,n)
8.    } ELSE {
9.              return TrovaMaxRel(a,c+1, j,n)
   }
```

Sia $T(n)$ il numero di operazioni effettuate dall'algoritmo `TrovaMaxRel(a,0,n-1,n)`.

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      } ELSE {
6.        IF (c > 0 && a[c-1] > a[c]) {
7.              return TrovaMaxRel(a,i, c-1,n)
8.    } ELSE {
9.              return TrovaMaxRel(a,c+1, j,n)
   }
```

Sia $T(n)$ il numero di operazioni effettuate dall'algoritmo `TrovaMaxRel(a,0,n-1,n)`. É ovvio che

$$T(n) \leq \begin{cases} c_0 & \text{se } n = 1 \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

per opportune costanti c_0 e c,

Pseudocodice

```
1. TrovaMaxRel(a,i,j,n)
2.     c=(i+j)/2
3.     IF((c==0 ||a[c-1]≤a[c])&&(a[c+1]≤a[c]||c == n-1)){
4.           return c
5.      } ELSE {
6.        IF (c > 0 && a[c-1] > a[c]) {
7.              return TrovaMaxRel(a,i, c-1,n)
8.    } ELSE {
9.              return TrovaMaxRel(a,c+1, j,n)
   }
```

Sia $T(n)$ il numero di operazioni effettuate dall'algoritmo `TrovaMaxRel(a,0,n-1,n)`. É ovvio che

$$T(n) \leq \begin{cases} c_0 & \text{se } n = 1 \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

per opportune costanti c_0 e c, da cui ne discende che $T(n) = O(\log n)$.

Altro esercizio

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]

Altro esercizio

Input: array `a=a[1]a[2]...a[n]`, con `a[1]<a[n]`
Output: un intero `i` per cui `a[i]<a[i+1]` (esiste sempre!).

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j].

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m]

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m] e a[m]...a[j],

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m] e a[m]...a[j], dove m=(i + j)/2 è l' indice mediano fra i e j, in base alle seguenti osservazioni:

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m] e a[m]...a[j], dove m=(i + j)/2 è l' indice mediano fra i e j, in base alle seguenti osservazioni:

Se a[i]<a[m],

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m] e a[m]...a[j], dove m=(i + j)/2 è l' indice mediano fra i e j, in base alle seguenti osservazioni:

Se a[i]<a[m], allora è possibile considerare il **solo** sottoproblema a[i]...a[m], in cui il primo estremo è minore dell'ultimo.

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m] e a[m]...a[j], dove m=(i + j)/2 è l' indice mediano fra i e j, in base alle seguenti osservazioni:

Se a[i]<a[m], allora è possibile considerare il **solo** sottoproblema a[i]...a[m], in cui il primo estremo è minore dell'ultimo.
Se a[m]$\leq$a[i]<a[j]

Altro esercizio

Input: array a=a[1]a[2]...a[n], con a[1]<a[n]
Output: un intero i per cui a[i]<a[i+1] (esiste sempre!).

Risolviamo (più in generale) il problema di identificare due valori consecutivi crescenti in un vettore a[i]...a[j] con a[i]<a[j]. Il problema originale corrisponde al caso a=a[1]a[2]...a[n]

Il problema relativo ad a[i]...a[j] puo essere ridotto ad **uno solo** dei sottoproblemi a[i]...a[m] e a[m]...a[j], dove m=(i + j)/2 è l' indice mediano fra i e j, in base alle seguenti osservazioni:

Se a[i]<a[m], allora è possibile considerare il **solo** sottoproblema a[i]...a[m], in cui il primo estremo è minore dell'ultimo.
Se a[m]$\leq$a[i]<a[j] allora è possibile considerare il **solo** sottoproblema a[m]...a[j] in cui il primo estremo è minore dell'ultimo.

Pseudocodice

```
1. Trova(a,i,j)
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
7.         return Trova(a,i,m)
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
7.          return Trova(a,i,m)
8.           } ELSE {
7.          return Trova(a,m,j)
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
7.          return Trova(a,i,m)
8.           } ELSE {
7.          return Trova(a,m,j)
    }
   }
```

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
7.          return Trova(a,i,m)
8.           } ELSE {
7.          return Trova(a,m,j)
    }
   }
```

Sia $T(n)$ il numero di operazioni effettuate dall'algoritmo `Trova(a,1,n)`

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
7.         return Trova(a,i,m)
8.          } ELSE {
7.         return Trova(a,m,j)
   }
  }
```

Sia $T(n)$ il numero di operazioni effettuate dall'algoritmo `Trova(a,1,n)`
É ovvio che

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 2 \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

per opportune costanti c_0 e c,

Pseudocodice

```
1. Trova(a,i,j)
2.   IF(i+1==j){
3.   return i
4.      } ELSE {
5.       m=(i+j)/2
6.       IF(a[i]<a[m]){
7.         return Trova(a,i,m)
8.          } ELSE {
7.         return Trova(a,m,j)
   }
  }
```

Sia $T(n)$ il numero di operazioni effettuate dall'algoritmo `Trova(a,1,n)`
É ovvio che

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 2 \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

per opportune costanti c_0 e c, da cui ne discende che $T(n) = O(\log n)$.