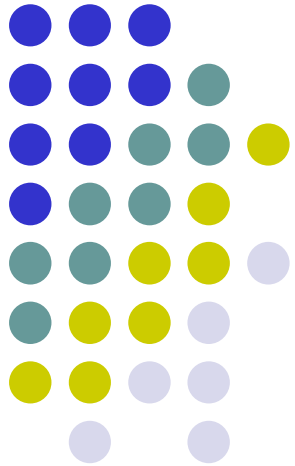
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica

A.A. 2006/2007



Dott.Davide Di Ruscio

Dipartimento di Informatica

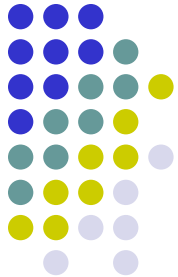
Università degli Studi di L'Aquila

Lezione del 13/02/07

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele

Sommario (II parte)



Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

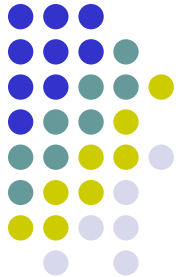
RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

McGraw-Hill

Ricerca di un elemento in un array

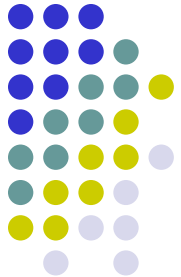


- Vediamo ora due metodi fondamentali per determinare se, dati un array di *n* interi ed un intero *x* forniti in input, l'elemento x è presente nell'array, ossia se esiste una componente dell'array avente lo stesso valore di *x*.

- Ricerca sequenziale: a partire dalla prima componente scandisce in sequenza una per una le componenti dell'array alla ricerca di *x*

- Ricerca binaria:
  - Può essere utilizzata solo se le componenti dell'array sono ordinate, ad esempio in modo crescente
  - E' molto più efficiente della ricerca sequenziale perché, sfruttando l'ordinamento delle componenti, dimezza ad ogni passo lo spazio di ricerca, ossia il numero di componenti da controllare

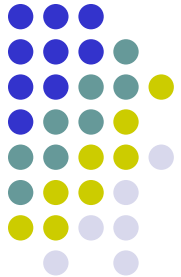
Ordinamento di un array



- Vediamo ora alcuni metodi fondamentali per ordinare in modo crescente gli elementi di un array di *n* interi fornito in input.

- Vedremo ora tre metodi o algoritmi noti:
  - Ordinamento per selezione
  - Ordinamento a bolle
  - Ordinamento per inserimento

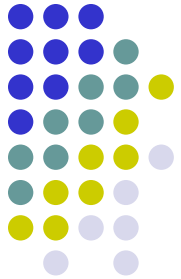
Ordinamento per selezione



- Idea di risoluzione:
  - Cerchiamo l'elemento minimo dell'array
  - Lo scambiamo con l'elemento a[0]
  - Ripetiamo il ragionamento per l'array a[1],…,a[n-1], dato che l'elemento a[0] è già nella sua posizione definitiva.

- Più precisamente:
  - Si eseguono n-1 passi, numerati da 0 ad n-2
  - Al generico passo i, avendo fissato le componenti di indici da 0 ad i-1 si fissa a[i] nel modo seguente:
    - si determina l'indice imin della componente minima a partire da quella di indice i in poi
    - si scambiano a[i] ed a[imin]

- N.B.: una volta fissata a[n-2], anche a[n-1] è nella sua posizione definitiva



Consideriamo il seguente array *a*:
13 11 19 7 14

Per ordinare *a*, possiamo cercare l'elemento minimo e portarlo in posizione a[0]. In questo caso, l'elemento piu' piccolo e' 7, memorizzato in a[3], quindi cambiamo di posto a[3] e a[0]:

7 11 19 13 14
|_____|

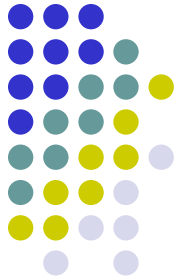
Ora possiamo cercare il minimo tra gli elementi rimanenti (a[1], a[2], a[3], a[4]), e portarlo nella posizione a[1]. In questo caso, il minimo e' 11, che si trova già al posto giusto.

7 11 19 13 14

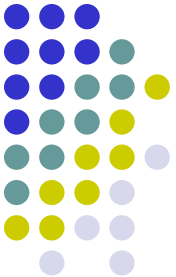
Ripetendo il procedimento ...
7 11 13 19 14
|__|

7 11 13 14 19
|__|

7 11 13 14 19



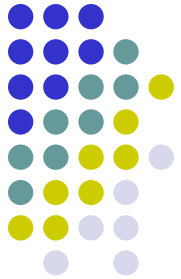
- Pseudocodice:
  - Per *i* che varia da 0 a *n-1* incrementando *i* di 1 ad ogni iterazione
    - Leggi da input *a[i]*
  - Per *i* che varia da 0 a *n-2* incrementando *i* di 1 ad ogni iterazione
    - Poni *imin* uguale ad *i*
    - Per *j* che varia da *i+1* a *n-1* incrementando *j* di 1 ad ogni iterazione
      - Se *a[j]* è minore di a*[imin]* poni *imin=j*
    - Scambia *a[i]* e *a[imin]*
  - Per *i* che varia da *0* a *n-1* incrementando *i* di 1 ad ogni iterazione
    - Stampa *a[i]*



```
main()
{
    const int n=100;
    int a[n],i,j,imin,aux;

    for (i=0; i<n; i=i+1)
          scanf("%d",&a[i]);
    for (i=0; i<=n-2; i=i+1)
    {
          imin=i;
          for (j=i+1; j<n; j++)
             if (a[j]<a[imin]) imin=j;
          aux = a[i];
          a[i] = a[imin];
          a[imin] = aux;
    }
     for (i=0; i<n; i=i+1)
          printf("%d ",a[i]);
}
```

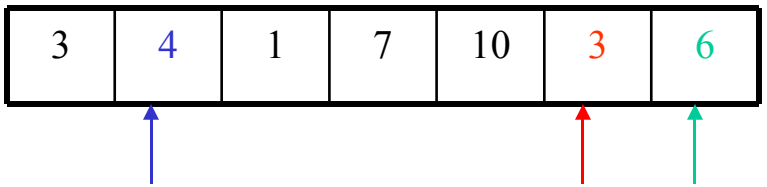
Ordinamento a bolle



- L'algoritmo si basa sul principio che in un array ordinato presi comunque due elementi adiacenti abbiamo che il primo è minore o uguale del secondo
- L'algoritmo confronta ripetutamente coppie adiacenti che vengono scambiate se non rispettano l'ordinamento
- Idea di risoluzione:
  - Si eseguono n-1 passi, numerati da 0 ad n-2
  - Al generico passo i, avendo fissato le componenti di indici da 0 ad i-1 si fissa a[i] nel modo seguente:
    - Si scambiano le componenti contigue di a che non rispettano l'ordinamento, a partire dal quelle di indici (n-2,n-1) fino a quelle di indici (i,i+1), ossia dall'ultima del vettore fino alla posizione i
- N.B.: al generico passo i la componente minima a partire dall'indice i viene spostata in posizione i; in altre parole, si segue la stessa idea di fondo dell'ordinamento per selezione
- All'inizio della iterazione *i* le prime (i-1) posizioni dell'array contengono i primi (*i*-1) posizioni dell'array ordinati
- Non sempre sono necessarie *n-1* iterazioni. Se durante una generica iterazione non avviene nessuno scambio, allora il vettore è stato ordinato e l'algoritmo può quindi terminare

Esempio

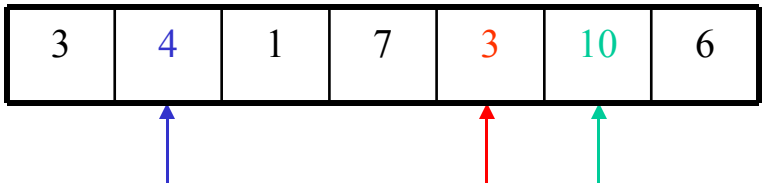
- Passo 1



i = 1
j = 6
j - 1

a[j-1] è minore di a[j], quindi non accade nulla.

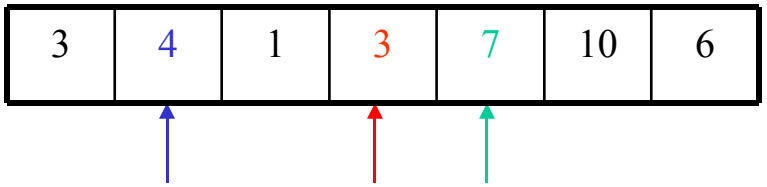
- Passo 2



i = 1
j = 5
j - 1

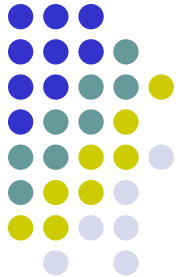
a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.

- Passo 3

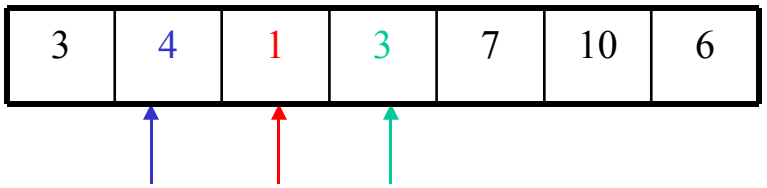


i = 1
j = 4
j - 1

a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.



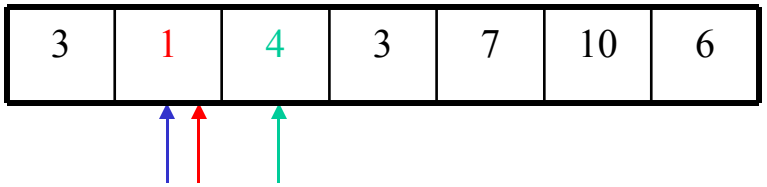
- Passo 4



i = 1
j = 3
j - 1

a[j-1] è minore di a[j], quindi non accade nulla.

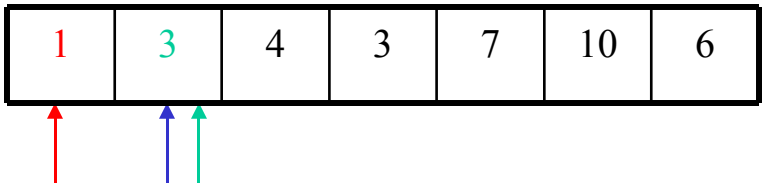
- Passo 5



i = 1
j = 2
j - 1

a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.

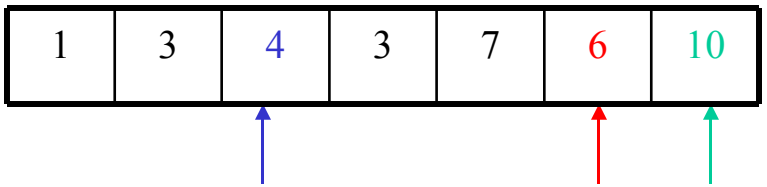
- Passo 6



i = 1
j = 1
j - 1

a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.

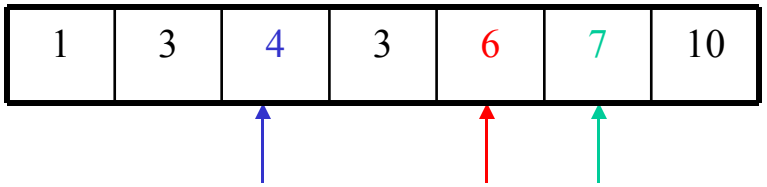
- Passo 7



i = 2
j = 6
j - 1

a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.

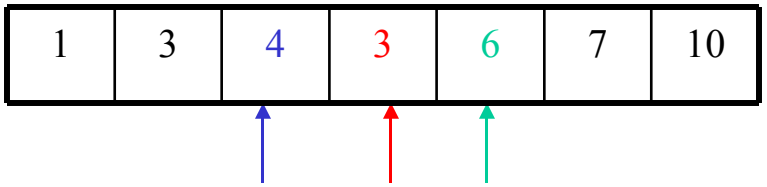
- Passo 8



i = 2
j = 5
j - 1

a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.

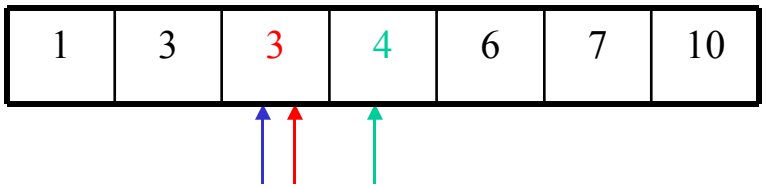
- Passo 9



i = 2
j = 4
j - 1

a[j-1] è minore di a[j], quindi non accade nulla.

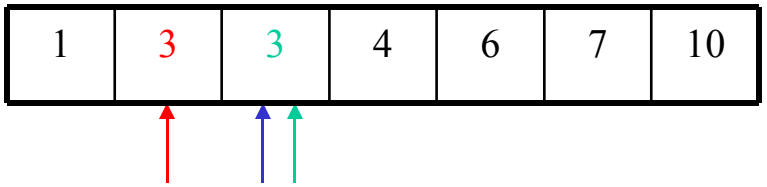
- Passo 7



i = 2
j = 3
j - 1

a[j-1] è maggiore di a[j], quindi vengono scambiati di posto.

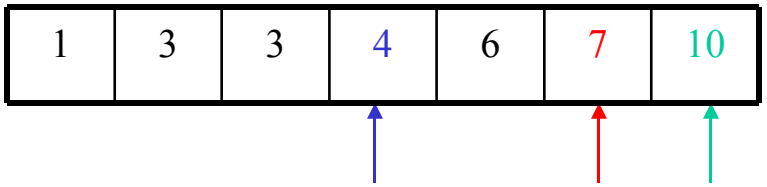
- Passo 8



i = 2
j = 2
j - 1

a[j-1] è uguale a[j], quindi non accade nulla.

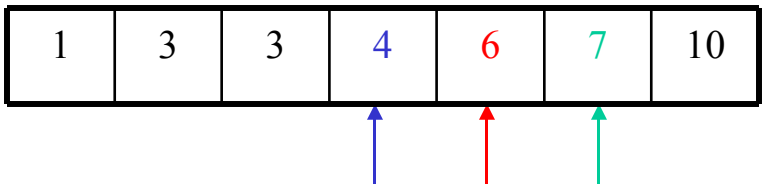
- Passo 9



i = 3
j = 6
j - 1

a[j-1] è minore di a[j], quindi non accade nulla.

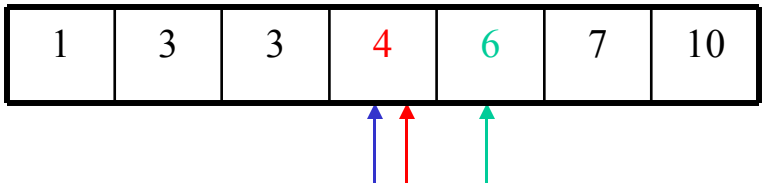
- Passo 10



i = 3
j = 5
j - 1

a[j-1] è minore a[j], quindi non accade nulla.

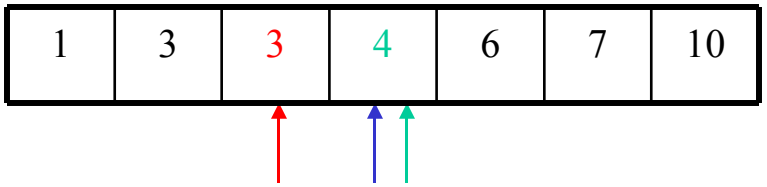
- Passo 11



i = 3
j = 4
j - 1

a[j-1] è minore a[j], quindi non accade nulla.

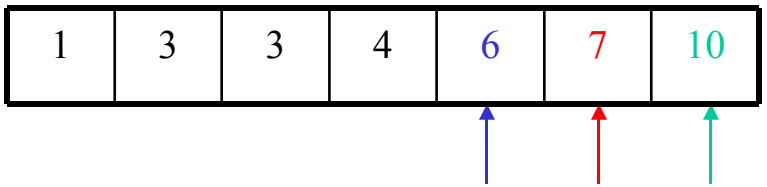
- Passo 12



i = 3
j = 3
j - 1

a[j-1] è minore di a[j], quindi non accade nulla.

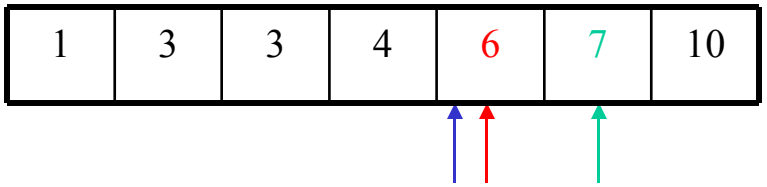
- Passo 10



i = 4
j = 6
j - 1

a[j-1] è minore a[j], quindi non accade nulla.

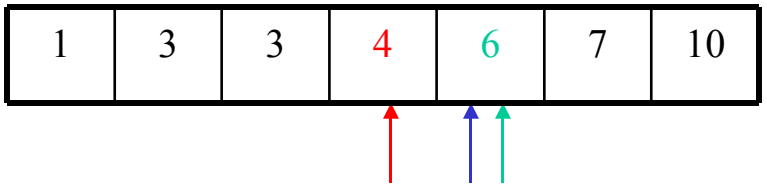
- Passo 11



i = 4
j = 5
j - 1

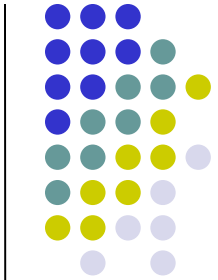
a[j-1] è minore a[j], quindi non accade nulla.

- Passo 12

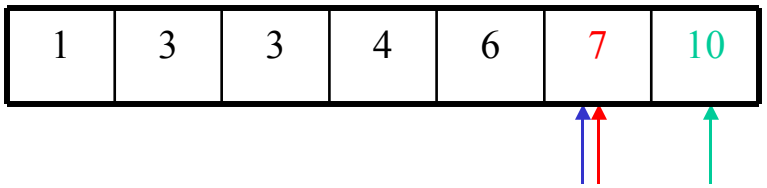


i = 4
j = 4
j - 1

a[j-1] è minore di a[j], quindi non accade nulla.



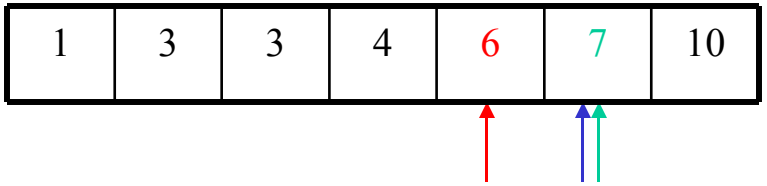
- Passo 10



i = 5
j = 6
j - 1

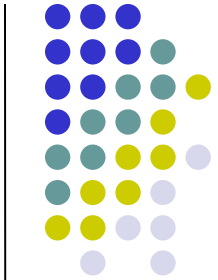
a[j-1] è minore a[j], quindi non accade nulla.

- Passo 11



i = 5
j = 5
j - 1

a[j-1] è minore a[j], quindi non accade nulla.



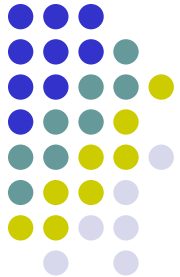
- Pseudocodice:
  - Per *i* che varia da 0 a *n-1* incrementando *i* di 1 ad ogni iterazione
    - Leggi *a[i]*
  - Per *i* che varia da 0 a *n-2* incrementando *i* di 1 ad ogni iterazione
    - Per *j* che varia da *n-2* ad *i* decrementando *j* di 1 ad ogni iterazione
      - Se *a[j]* è maggiore di a*[j+1]*
        - Scambia *a[j]* e *a[j+1]*
  - Per *i* che varia da 0 a *n-1* incrementando *i* di 1 ad ogni iterazione
    - Stampa *a[i]*

```
#include<stdio.h>

main()
{
    const int n=30;
    int a[n],i,j,aux;

    for (i=0; i<n; i=i+1)
          scanf("%d",&a[i]);
    for (i=0; i<n-1; i=i+1)
          for (j=n-2; j>=i; j=j-1)
                if (a[j]>a[j+1])
                {
                          aux=a[j];
                          a[j]=a[j+1];
                          a[j+1]=aux
                }
    for (i=0; i<n; i=i+1)
          printf("%d ",a[i]);
}
```

Ordinamento per inserimento

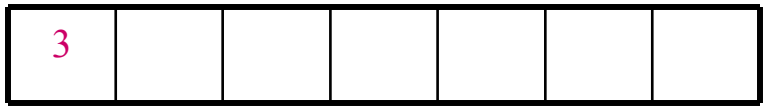


- Idea di risoluzione
  - Vengono effettuati *n* passi, numerati da *1* a *n-1*.
  - Al generico passo i, avendo già ordinato il sottovettore dei primi i elementi del vettore, ossia dall'indice 0 all'indice i-1, si inserisce a[i] in tale sottovettore nella posizione che rispetta l'ordinamento, eventualmente traslando gli elementi successivi, in modo da formare un sottovettore di i+1 componenti ordinato
  - N.B.: chiaramente alla prima iterazione il sottoarray composto dal solo a[0] è banalmente ordinato.
- Questo ordinamento è particolarmente efficace quando vuole ordinare il vettore a in fase di lettura dei suoi elementi
- Non è molto diverso dal modo in cui un essere umano, spesso, ordina un mazzo di carte
- L'algoritmo tende a spostare man mano gli elementi maggiori verso destra

Esempio

- Passo 1

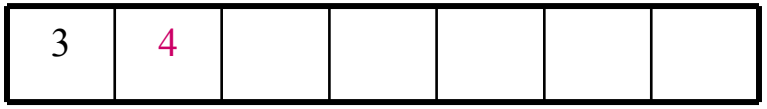
Acquisisco l'elemento 3 che viene messo nella posizione j+1



i = 0
j = -1

- Passo 2

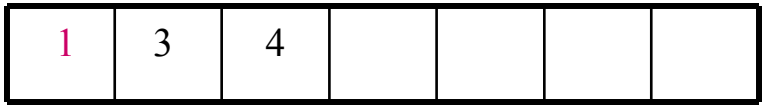
Acquisisco l'elemento 4 che viene messo nella posizione j+1



i = 1
j = 0

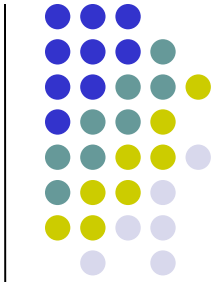
- Passo 3

Acquisisco l'elemento 1



i = 2
j = -1

j è minore di zero, quindi si esce dal ciclo e si posiziona 1 in posizione 0



- Passo 4

Acquisisco l'elemento 7

1	3	4	7			

i = 3
j = 2

a[j] è minore di 7, quindi il nuovo elemento viene inserito nella posizione j+1.

- Passo 5

Acquisisco l'elemento 10

1	3	4	7	10		

i = 4
j = 3

a[j] è minore di 10, quindi il nuovo elemento viene inserito nella posizione j+1

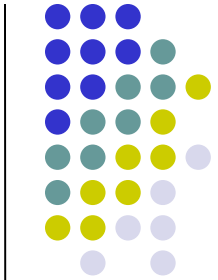
- Passo 6

Acquisisco l'elemento 3

1	3	3	4	7	10	

i = 5
j = 1

a[j] è minor uguale di 3, quindi 3 viene inserito in j+1.



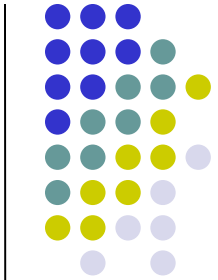
- Passo 7

Acquisisco l'elemento 6

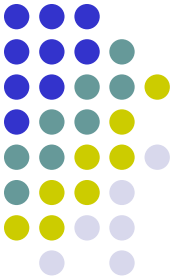
1	3	3	4	6	7	10

i = 6
j = 3

a[j] è minore uguale di 3, quindi 3 viene inserito in j+1.



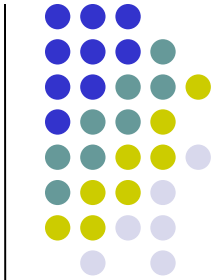
- Pseudocodice:
  - Per *i* che varia da 0 a *n-1* incrementando *i* di 1 ad ogni iterazione
    - Leggi *a[i]*
  - Per *i* che varia da 1 a n-*1* incrementando *i* di 1 ad ogni iterazione
    - Poni *aux =a[i]*
    - Poni *j=i-1*
    - Mentre *j* è maggiore o uguale a 0 e *a[j]* è maggiore di *aux*
      - Poni *a[j+1]=a[j]*
      - Decrementa *j* di 1
    - Poni *a[j+1]=aux*
  - Per *i* che varia da 0 a *n-1* incrementando *i* di 1 ad ogni iterazione
    - Stampa *a[i]*



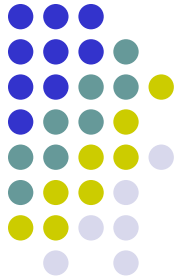
```
main()
{
    const int n=20;
    int a[n],i,j,aux;

    for (i=0; i<n; i=i+1)
          scanf("%d",&a[i]);
    for (i=1; i<n; i++)
    {
          aux = a[i];
          j = i-1;
          while (j>=0 && a[j]>aux)
          {
                    a[j+1] = a[j];
                    j--;
          }
          a[j+1] = aux;
    }
    for (i=0; i<n; i=i+1)
          printf("%d ",a[i]);
}
```

Scambio righe



- L’utente inserisce una matrice composta da numeri interi
- Il programma scambia le righe pari con quelle dispari
- Le dimensioni della matrice inserita dall’utente sono fissate all’interno del programma

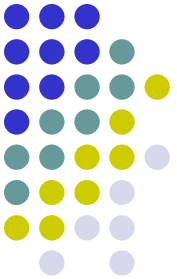


Pseudocodice

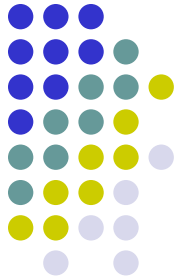
- for ([contatore riga])
  - for ([contatore colonna])
    - [leggi numero e mettilo nella cella (riga, colonna)]

- for ([contatore riga; va da zero al numero i righe della matrice, meno uno; incremento di 2])
  - for ([contatore colonna])
    - [scambia la cella (riga, colonna) con quella (riga + 1, colonna)]
- for ([contatore riga])
  - for ([contatore colonna])
    - [visualizza numero nella cella (riga, colonna)]

```c
#include <stdio.h>
void main()
{
    const unsigned int DIMRIGA = 4, DIMCOL = 4;
    unsigned int riga, colonna;
    int matrice [DIMRIGA][DIMCOL], temporanea;
    for (riga = 0; riga < DIMRIGA; riga++)
        for (colonna = 0; colonna < DIMCOL; colonna++)
        {
                printf ("numero (%u,%u): ", riga, colonna);
                scanf ("%d", &matrice[riga][colonna]);
        }
    for (riga = 0; riga < DIMRIGA - 1; riga = riga + 2)
        for (colonna = 0; colonna < DIMCOL; colonna++)
        {
                temporanea = matrice[riga][colonna];
                matrice[riga][colonna]=matrice[riga + 1][colonna];
                matrice[riga + 1][colonna] = temporanea;
        }
    for (riga = 0; riga < DIMRIGA; riga++)
    {
        for (colonna = 0; colonna < DIMCOL; colonna++)
                printf ("%d ", matrice[riga][colonna]);
        printf ("\n");
    }
}
```



Calcolo somma di matrici



- Nell'insieme delle matrici quadrate di ordine *n* è possibile definire la somma

- Se

$$A = (a_{ij}),\ B=(b_{ij}) \quad i,j = 1,\ldots,n$$

  sono due matrici quadrate di ordine *n*, si chiama matrice somma la matrice

$$C=(c_{ij}) \text{ con } c_{ij}=a_{ij}+b_{ij} \qquad i,j =1,\ldots,n$$

- La matrice somma, ovvero C = A+B, si può calcolare con il seguente programma:

```c
#include <stdio.h>
void main()
{
    const unsigned int DIM = 3;
    unsigned int riga, colonna;
    int a[DIM][DIM], b[DIM][DIM], c[DIM][DIM];
    for (riga = 0; riga < DIM; riga++)
          for (colonna = 0; colonna < DIM; colonna++)
          {
                    printf("Inserisci elemento a[%d][%d]",riga,colonna);
                    scanf("%d",&a[riga][colonna]);
          }
    for (riga = 0; riga < DIM; riga++)
          for (colonna = 0; colonna < DIM; colonna++)
          {
                    printf("Inserisci elemento b[%d][%d]",riga,colonna);
                    scanf("%d",&b[riga][colonna]);
          }
    for (riga = 0; riga < DIM; riga++)
          for (colonna = 0; colonna < DIM; colonna++)
              c[riga][colonna]= a[riga][colonna] + b[riga][colonna];

    for (riga = 0; riga < DIM; riga++)
    {
          for (colonna = 0; colonna < DIM; colonna++)
              printf("%d ",c[riga][colonna]);
          printf("\n");
    }
```