

FONDAMENTI DI INFORMATICA

OTTAVA LEZIONE

Prof. Alfredo Accattatis

(accattatis@ing.uniroma2.it)

Tutor: Prof. Vittorio Calafiore

(vcalafio@gmail.com)

I numeri reali

- **Razionali**: rapporto tra due numeri interi (numero di cifre finito dopo la virgola oppure sequenza infinita periodica).
- **Irrazionali** (non possono essere espressi come rapporto di interi, costituiti da sequenze infinite di numeri senza apparente regolarità). Esempio: radice quadrata di 2.
- **Trascendenti**: sono ancora numeri *irrazionali* ma con la caratteristica aggiuntiva di NON essere la soluzione di equazioni polinomiali (v. testo). Esempio: pi-greco.

Tipo Numero Reale

- Tipo doppia precisione (o **double**) : IEEE Standard 754 : ogni valore memorizzato come double occupa 64 bit.

Bits	Usage
63	Sign (0 = positive, 1 = negative)
62 to 52	Exponent, biased by 1023
51 to 0	Fraction f of the number $1.f$

- Tipo singola precisione (o **single**) : IEEE Standard 754: ogni valore memorizzato come single occupa 32 bit.

Bits	Usage
31	Sign (0 = positive, 1 = negative)
30 to 23	Exponent, biased by 127
22 to 0	Fraction f of the number $1.f$

Numeri reali rappresentati da un single sono rappresentati con **meno precisione** rispetto ai numeri rappresentati da un tipo double.

Rappresentazione di numeri reali

(Matlab: *double*)

- Con un numero finito di cifre è possibile rappresentare solo un **numero reale** che *approssima* con un certo *errore* il numero reale dato (razionale e irrazionale)
- Vengono usate due notazioni
 - Notazione in virgola fissa
 - Dedica parte delle cifre alla parte intera e le altre alla parte frazionaria (la posizione della virgola è fissata): $\pm \text{XXX}.\text{YY}$
 - Notazione in virgola mobile
 - Dedica alcune cifre a rappresentare un esponente della base che indica l'ordine di grandezza del numero rappresentato
 - Esempio notazione scientifica $\pm \text{X.XX} \cdot 10^{\pm \text{YY}}$

Perché la rappresentazione in virgola mobile

- Limitazioni della rappresentazione in virgola fissa
 - Fissato il numero di cifre e la posizione della virgola:
 1. Non rappresenta bene numeri frazionari molto grandi
 2. Non rappresenta bene numeri frazionari molto piccoli

FISSATO IL NUMERO DI CIFRE PER RAPPRESENTARE PARTE INTERA E FRAZIONARIA

- Come rappresentare in virgola fissa:
 - 5000000000000000.00003
 - 432 000 000 000
 - 0.00000000000000000008
- La rappresentazione in virgola mobile estende l'intervallo di numeri rappresentati a parità di cifre, rispetto alla notazione in *virgola fissa*
- *notazione scientifica*
 - Si esprime 432 000 000 000 come 4.32×10^{11}
 - Le 11 posizioni dopo il 4 vengono espresse dall'esponente
- Principio della rappresentazione in virgola mobile (detta anche *floating point*)
 - Si fa scorrere la virgola decimale fino ad una posizione conveniente, tramite l'esponente

Rappresentazione in virgola mobile

- E' utile perché
 - Permette di rappresentare in maniera compatta numeri molto grandi, ma anche molto piccoli, sia positivi che negativi
- Numeri reali rappresentati tramite una coppia di numeri $\langle m, e \rangle$

$$n = \pm m \cdot b^{\pm e}$$

- m : *mantissa* (detto anche *significante*), normalizzata tra due potenze successive della *base* b

$$b^{i-1} \leq |m| < b^i$$

- e : *esponente* intero (detto anche *caratteristica*)
- Sia m che e hanno un numero fissato di cifre:
 - Intervalli limitati
 - Errori di arrotondamento

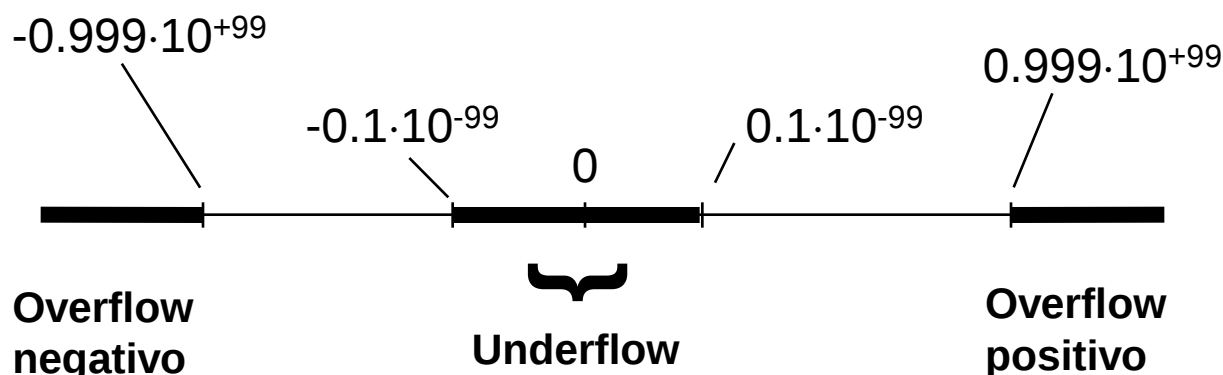
Esempio in base 10

- Numerali a 5 cifre $\pm .XXX \pm EE$
- Mantissa: 3 cifre con segno (normalizzata tra due potenze successive della base b)

$$0.1 \leq |m| < 1$$

- Esponente: 2 cifre con segno

$$-99 \leq e \leq +99$$



- Notare che con lo stesso numero di cifre in notazione a virgola fissa $\pm XXX.YY$
 - L'intervallo scende $[-999.99, +999.99]$
 - Ma si hanno 5 cifre significative invece di 3

Normalizzare la mantissa. Perché?

- Nell'esempio della slide precedente tra 0.1 e 1.
- Per ottenere UNIFORMITA' di rappresentazione
- Infatti dato il numero 1440 esso si potrebbe rappresentare come:
 - 1440×10^0
 - 1.440×10^3
 - 0.144×10^4
 - 14400×10^{-1}
- Imponendo la normalizzazione esso sarà sempre rappresentato in maniera UNIVOCA = 0.144×10^4
- Corrispondenza BIUNIVOCA tra entità e sua RAPPRESENTAZIONE

Rappresentazione in virgola mobile dei numeri binari

- Lo stesso approccio della virgola mobile può essere seguito per rappresentare i numeri binari come $\pm m \cdot b^{\pm e}$

$$\pm 1.xxxxxxxxxx_2 \cdot 2^{\pm yyyy_2}$$

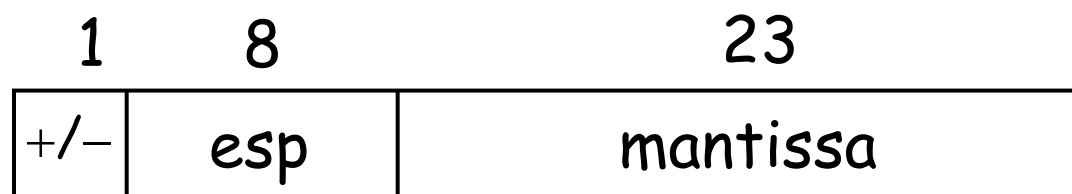
- Da notare che, di solito, la base b è implicita e quindi
 - È sufficiente memorizzare *segno*, *mantissa* (o significante) ed *esponente* (o caratteristica)
- Si usa un certo numero di bit (almeno 32)
 - Si riserva spazio per segno, mantissa ed esponente

Standard per la Rappresentazione

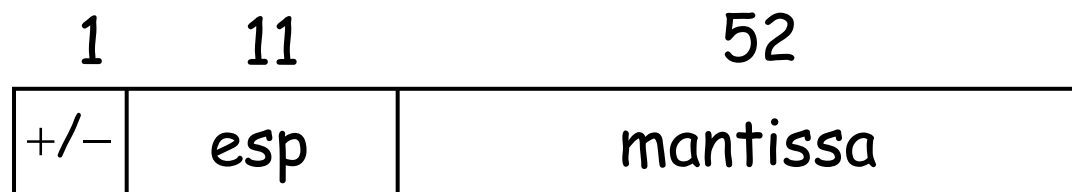
- Importanza di definire uno standard per la rappresentazione dei numeri in virgola mobile
 - Per definire la semantica delle istruzioni in virgola mobile
- IEEE Computer Society (Institute of Electrical and Electronics Engineers) definisce lo “IEEE standard for binary floating arithmetic” (riferito come **IEEE 754**) nel 1985
 - Specifica il formato, le operazioni, le conversioni tra i diversi formati floating point e quelle tra i diversi sistemi di numerazione, il trattamento delle eccezioni
- Nel 1989 IEEE 754 diventa uno standard internazionale (IEC 559)
 - IEC: International Electrotechnical Commission
 - International Standards and Conformity Assessment for all electrical, electronic and related technologies

Standard IEEE 754

- Formato *non proprietario*, ossia non dipendente dall'architettura del calcolatore
- Precisione *semplice* a 32 bit:



- Precisione *doppia* a 64 bit:



- Notazioni in modulo e segno
- Alcune configurazioni dell'esponente sono riservate
- Segno (1 bit): 0 per positivo, 1 per negativo

IEEE 754 a 32 bit: esponente

- Esponente (8 bit)
 - Rappresentato in eccesso 127 (*polarizzazione* o *bias*)
 - L'intervallo di rappresentazione è $[-127, 128]$
 - Le due configurazioni estreme sono riservate, quindi
$$-126 \leq e \leq 127$$
 - Se gli 8 bit dell'esponente contengono $10100011 = 163_{10}$
 - L'esponente vale $163 - 127 = 36$
 - Se gli 8 bit dell'esponente contengono $00100111 = 39_{10}$
 - L'esponente vale $39 - 127 = -88$
- Perché la polarizzazione?
 - Il numero più grande che può essere rappresentato è $11...11$
 - Il numero più piccolo che può essere rappresentato è $00...00$
 - Quindi, quando si confrontano due interi polarizzati, per determinare il minore basta considerarli come interi senza segno
 - **Più facile progettare la ALU**

Numeri normalizzati

- Un numerale si intende in questa rappresentazione quando $e \neq 00000000$
- In questa rappresentazione, la mantissa è normalizzata tra 1 e 2:

$$1 \leq m < 2$$

- Quindi, la mantissa è sempre nella forma:

$$1.XXXXXXXXXX...X$$

- Si usano tutti i 23 bit per rappresentare la sola parte frazionaria (1 prima della virgola è implicito)
- Gli intervalli di numeri rappresentati sono pertanto:
$$(-2^{128}, -2^{-126}] \quad [2^{-126}, 2^{128})$$
 - Gli estremi sono esclusi perché il massimo valore assoluto di m è molto vicino a 2, ma è comunque inferiore

Numeri denormalizzati

- Un numerale si intende in questa rappresentazione quando **$e=00000000$**
- L'esponente assume il valore ***convenzionale*** -126
- La mantissa è normalizzata tra 0 e 1: **$0 < m < 1$**
- Quindi, la mantissa è sempre nella forma:
 $0.XXXXXXXXXXX...X$
- Si usano tutti i 23 bit per rappresentare la sola parte frazionaria
- La più piccola mantissa vale 2^{-23}
- Gli intervalli rappresentati sono:
 $(-2^{-126}, -2^{-149}] \quad [2^{-149}, 2^{-126})$

Standard IEEE 754 a 32 bit: estremi degli intervalli

- Più grande normalizzato: $\sim \pm 2^{128}$

X 11111110 11111111111111111111111111111111
 $\pm \quad 2^{127} \quad \sim 2$

- Più piccolo normalizzato: $\pm 2^{-126}$

X 00000001 00000000000000000000000000000000
 $\pm \quad 2^{-126} \quad 1$

- Più grande denormalizzato: $\sim \pm 2^{-126}$

X 00000000 11111111111111111111111111111111
 $\pm \quad 2^{-126} \quad (0.11\dots)_2 \approx 1$

- Più piccolo denormalizzato: $\pm 2^{-149}$

X 00000000 00000000000000000000000000000001
 $\pm \quad 2^{-126} \quad (0.00\dots1)_2 = 2^{-23}$

Esempi: conversione da virgola mobile

- Quale numero in singola precisione rappresentano i seguenti 32 bit

1 10000001 010000000000000000000000

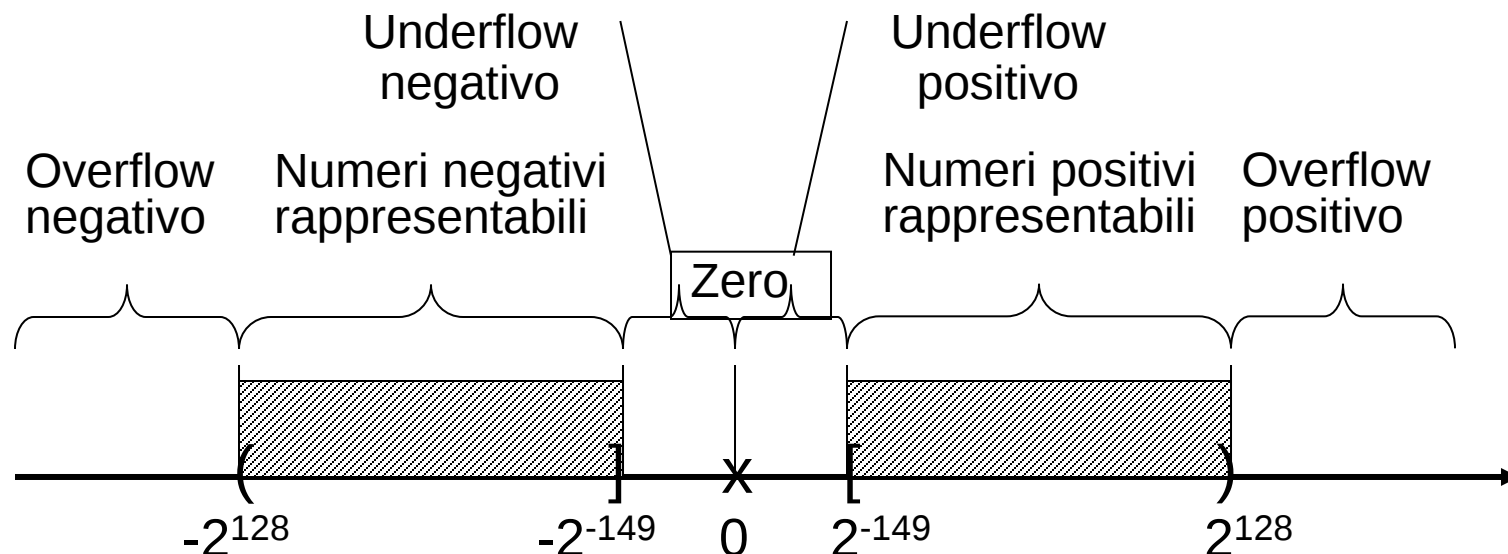
- Segno negativo (-)
- Esponente $e = 2^7 + 2^0 - 127 = 129 - 127 = 2$
- Mantissa $m = 1 + 2^{-2} = 1.25$
- Quindi il numero rappresentato è $-1.25 \cdot 2^2 = -5$

Rapp.
in eccesso 127

0 10000011 100110000000000000000000

- Segno positivo (+)
- Esponente $e = 2^7 + 2^1 + 2^0 - 127 = 131 - 127 = 4$
- Mantissa $m = 1 + 2^{-1} + 2^{-4} + 2^{-5} = 1.59375$
- Quindi il numero rappresentato è $1.59375 \cdot 2^4 = 25.5$

Intervallo di rappresentazione



❑ L' overflow può essere positivo

Quando si devono rappresentare numeri positivi maggiori di 2^{128}

❑ L' overflow può essere negativo

Quando si devono rappresentare numeri negativi minori di -2^{128}

❑ L' underflow può essere positivo

Quando si devono rappresentare numeri positivi minori di 2^{-149}

❑ L' underflow può essere negativo

Quando si devono rappresentare numeri negativi maggiori di -2^{-149}

Esempi: conversione in virgola mobile

- Quale è la rappresentazione a singola precisione del numero

8.5

- Segno positivo (0)
- 8.5 in binario è $1000.1 \cdot 2^0 = 1.0001 \cdot 2^3$
- Esponente e: $3 + 127 = 130 = 10000010$
- Mantissa m: 000100000000000000000000
- Quindi 0 10000010 000100000000000000000000

-13.75

- Segno negativo (1)
- 13.75 in binario è $1101.11 \cdot 2^0 = 1.10111 \cdot 2^3$
- Esponente e: $3 + 127 = 130 = 10000010$
- Mantissa m: 101110000000000000000000
- Quindi 1 10000010 101110000000000000000000

Conversione decimale-binario per i numeri con parte frazionaria

- Procedimento che a partire da una frazione $F < 1$
 - Ottiene la rappresentazione di F in binario (o una sua approssimazione)
- Sequenza ripetuta di moltiplicazioni per 2
 - Si ottiene una parte intera (che può essere 0 o 1) ed una parte decimale F_1
 - La parte intera rappresenta il bit più significativo, mentre F_1 viene moltiplicato per 2
 - Si ottiene una parte intera (che può essere 0 o 1) ed una parte decimale F_2
 - La parte intera rappresenta il secondo bit più significativo, mentre F_2 viene moltiplicato per 2
 - ... fino a quando F_i è uguale a 0 oppure abbiamo esaurito il numero di cifre a disposizione per la rappresentazione
 - Nel secondo caso introduciamo un'approssimazione

Esempio: conversione parte frazionaria

□ Conversione di 0.73 con 8 bit

$0.73 \cdot 2 = 1.46$	1		cifra più significativa
$0.46 \cdot 2 = 0.92$	0		
$0.92 \cdot 2 = 1.84$	1		
$0.84 \cdot 2 = 1.68$	1		
$0.68 \cdot 2 = 1.36$	1		
$0.36 \cdot 2 = 0.72$	0		
$0.72 \cdot 2 = 1.44$	1		
$0.44 \cdot 2 = 0.88$	0		
			cifra meno significativa

- Quindi $0.73_{10} = 0.10111010_2$
- Con 8 bit rappresentiamo il numero 0.7265625

Confronto tra numeri in virgola mobile

Per stabilire quale di due numeri in virgola mobile sia il maggiore:

- Se sono di segno *discorde*, allora il numero positivo è maggiore
- Se sono di segno *concorde*
 - Se sono *positivi*
 - Il numero con l'esponente *più grande* è il maggiore; a parità di esponente, il numero con mantissa *più grande* è maggiore
 - Se sono *negativi*
 - Il numero con l'esponente *più piccolo* è il maggiore; a parità di esponente, il numero con mantissa *più piccola* è maggiore

Confronto tra numeri in virgola mobile (2)

- Per due numeri positivi (negativi)
 - Siano a e b due numeri positivi (negativi) rappresentati in virgola mobile da $a_{31}a_{30}...a_0$ e $b_{31}b_{30}...b_0$
 - Notare che $a_{31} = b_{31} = 0$ (1)
- Per verificare quale dei due sia il maggiore, non occorre nessuna conversione
 - È sufficiente confrontarli come se fossero interi senza segno
 - Basta scorrere i bit, ed al primo bit diverso si individua il maggiore
 - Il numero con l' i -esimo bit a 1 (0)
 - Il numero con esponente/mantissa più grande (più piccolo) è il maggiore

Configurazioni particolari

- Lo standard IEEE 754 attribuisce valori convenzionali a particolari configurazioni di e ed m
 - e ed m tutti 0: rappresentano il valore 0 (*altrimenti non rappresentabile*)
 - m tutti 0 ed e tutti 1: rappresentano l'infinito ($\pm\infty$)
 - $m \neq 0$ ed e tutti 1: rappresentano la situazione *Not A Number (NaN)*, cioè un valore indefinito (ad es. il risultato di una divisione per 0 o la radice quadrata di un numero negativo)

Normalizzato	$\pm$	$0 < \text{esp} < \text{Max}$	Qualsiasi stringa di bit
Denormalizzato	$\pm$	0	Qualsiasi stringa di bit diversa da zero
Zero	$\pm$	0	0
Infinito	$\pm$	111...1	0
NaN	$\pm$	111...1	Qualsiasi stringa di bit diversa da zero

Osservazioni sulla precisione singola

- In modo assolutamente indipendente dalla rappresentazione usata, con 32 bit è possibile rappresentare “soltanto” 2^{32} valori diversi
- I numeri rappresentati in virgola mobile hanno una densità maggiore vicino allo zero

IEEE 754 a 64 bit

- Segno (1 bit)
- Esponente (11 bit)
 - Rappresentato in eccesso 1023
 - L'intervallo di rappresentazione è $[-1023, 1024]$
- Mantissa (52 bit)
 - Normalizzata come nella singola precisione
- Configurazione riservate come nella singola precisione per la rappresentazione di
 - 0
 - Numeri denormalizzati (positivi e negativi)
 - Infinito (positivo e negativo)
 - NaN (Not a Number)
- Esercizio
 - Qual'è l'intervallo di rappresentazione dei numeri a doppia precisione?

Errore assoluto ed errore relativo

- Rappresentando un numero reale n nella notazione floating point si commette un errore di approssimazione
- In realtà viene rappresentato un numero razionale n' con un numero limitato di cifre significative
- **Errore assoluto**: $e_A = n - n'$
- **Errore relativo**: $e_R = e_A / n = (n - n') / n$
- Se la mantissa è normalizzata, l'errore relativo **massimo** è costante su tutto l'intervallo rappresentato, ed è pari ad un'unità sull'ultima cifra rappresentata
 - Esempio: 10 cifre frazionarie $e_R = 2^{-10}$
- Nella notazione non normalizzata, l'errore relativo massimo non è costante

Argomenti

- Approfondimenti sui floating-point:
 - Arrotondamento
 - Epsilon macchina
 - Problemi comuni

Arrotondamento in virgola mobile

Double-precision (64-bit)

Bits	Usage
63	Sign (0 = positive, 1 = negative)
62 to 52	Exponent, biased by 1023
51 to 0	Fraction f of the number $1.f$

Single-precision (32-bit)

Bits	Usage
31	Sign (0 = positive, 1 = negative)
30 to 23	Exponent, biased by 127
22 to 0	Fraction f of the number $1.f$

La rappresentazione in virgola mobile (IEEE 754) di un numero reale è un'approssimazione poiché utilizza un numero finito di bit.

$$\mathbb{R} \longrightarrow \pm 1,xxxxxxxxx_2 \cdot 2^{\pm yyy_2}$$

Il numero reale viene approssimato: si produce un errore.

Epsilon macchina

Il numero in virgola mobile x e il successivo differiscono di un quantità finita non nulla: tra i due non esiste altro numero rappresentabile dal computer.

Epsilon macchina $\text{eps}(x)$ fornisce la distanza tra $|x|$ e il numero in virgola mobile successivo.

x	$\text{eps}(x)$
1	$2,2204 \cdot 10^{-16}$
100	$1,4211 \cdot 10^{-14}$
10^5	$1,1642 \cdot 10^{-10}$

I numeri non sono equidistanti tra di loro: il peso del bit meno significativo della mantissa dipende dall'esponente.

Problemi di aritmetica in virgola mobile

Esempio pratico delle grandezze in gioco

Distanza sole-terra $1.5 \cdot 10^{11}$ m

Diametro capello umano $1,5 \cdot 10^{-5}$ m

```
>> eps(1011) = 1,5259*10-5
```



Numeri molto diversi in ordine di grandezza: possiamo avere problemi nel calcolo aritmetico

```
>> a = 10-16;  
>> b = 10-8;  
>> x = a + 1 - b*b;  
>> y = 1;  
>> x == y
```

ans = 0 (cioè FALSE) !

Evitare problemi comuni

Calcolo aritmetico

```
>> x = 4
```

```
>> f(x) = 1 - 3*(x/3 - 1)
```

```
f(4) = 2.2204e-16
```

L'errore si risolve sviluppando la funzione

$$f(x) = 1 - x + 3$$

$$= 4 - x$$

```
f(4) = 0
```

STRINGHE

Manipolazione di stringhe

- **Stringa**: qualsiasi sequenza di caratteri delimitata da virgolette singole
- Le stringhe sono sostanzialmente vettori le cui componenti sono caratteri
 - Infatti le operazioni definite sui vettori sono valide **quasi** tutte anche con le stringhe
 - **Esempi di stringhe:**
 - `''`
 - `'x'`
 - `'cat'`
 - `'Hello there'`
 - `'123'`
 - **Sottostringa (Substring)**: porzione di una stringa
 - `'there'` è una sottostringa di `'Hello there'`

Tipi di caratteri

- Una stringa può contenere
 - Lettere
 - Numeri
 - Segni di punteggiatura
 - Spazi bianchi
 - Caratteri di controllo
- I “caratteri di controllo” non sono “stampabili” ma “fanno qualcosa” (e.g. tab , new line, carriage return)

Come creare variabili stringhe

Command Window

```
>> word='cat';  
>> strvar=input('enter a string ','s');  
enter a string  THIS IS A STRING  
>> strvar
```

```
strvar =
```

```
THIS IS A STRING
```

```
fx >>
```

Stringhe come Vettori

```
>> length(word)
ans =
     3
>> length('cat')
ans =
     3
>> length(' ')
ans =
     1
>> length('')
ans =
     0
```

```
>> mystr='Hi'
mystr =
Hi
>> mystr(1)
ans =
H
>> mystr'
ans =
H
i
>> sent = 'Hello there'
sent =
Hello there
>> sent(4:8)
ans =
lo th
```

Matrici di caratteri o di stringhe?

```
>> wordmat=[ 'Hello'; 'Howdy']  
wordmat =  
Hello  
Howdy  
>> size( wordmat )  
ans =  
      2      5  
>> wordmat( 2, 4 )  
ans =  
d  
>> wordmat( 1, : )  
ans =  
Hello  
>>
```

```
>> greetmat = ['Hello'; 'Goodbye']  
Error using vertcat  
Dimensions of matrices being  
concatenated are not  
consistent.
```

```
>> greetmat = ['Hello  ';  
'Goodbye']  
greetmat =  
Hello  
Goodbye
```

CARATTERI

Operazioni sulle stringhe

- Concatenazione
- Creazione di stringhe personalizzate
- Rimozione spazi
- Conversione minuscole/maiuscole e viceversa
- Confronti
- Ricerca

Concatenazione (1)

```
>> f='bird';
>> l='house';
>> [f l]
ans =
birdhouse
>> strcat(f,l)
ans =
birdhouse
```

strcat e **[]** NON hanno lo stesso comportamento

```
>> s1='xxx ';
>> length(s1)
ans =
6
>> s2=' yy';
>> length(s2)
ans =
6
>> [s1 s2]
ans =
xxx yy
>> length(ans)
ans =
12
>> strcat(s1,s2)
ans =
xxx yy
>> length(ans)
ans =
9
>> strcat(s2,s1)
ans =
yyxxx
>> length(ans)
ans =
9
```

Rimuove gli "spazi bianchi"

Concatenazione (2)

- char → concatenazione verticale

```
>> clear greatmat
```

```
>> greatmat=char('Hello','Goodby');
```

```
>> greatmat
```

```
greatmat =
```

```
Hello
```

```
Goodby
```

```
>> size(greatmat)
```

```
ans =
```

```
2
```

```
6
```

La taglia della matrice dipende dalla stringa, tra quelle concatenate, di dimensione maggiore

Creazione di stringhe di caratteri “blanks”

- Aggiungere spazi bianchi alla fine di una stringa può tornare utile per concatenare righe

- `blanks()`

```
>> bs4=blanks(4);
```

```
>> size(bs4)
```

```
ans =
```

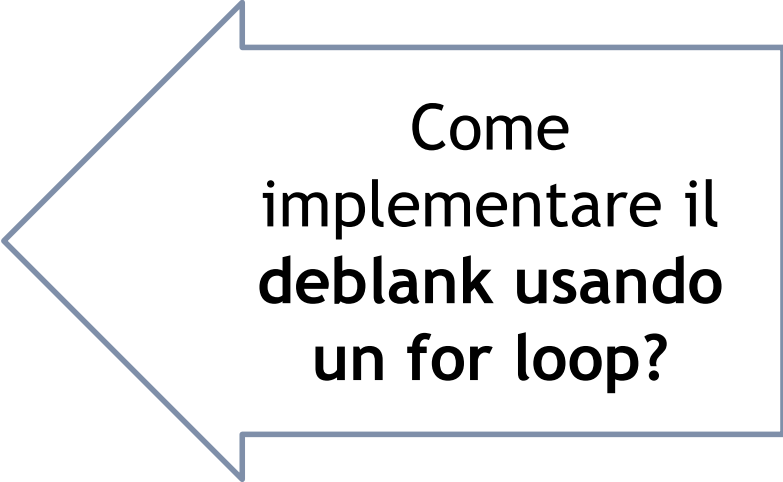
```
1      4
```

```
>> tx=['Yes We can'; 'I did ' , bs4]
```

- Potrebbe esserci utile anche il processo inverso, ossia la rimozione...

Rimozione spazi bianchi

```
>> names=char('Sue','Cathy','Xavier');  
>> size(names)  
ans =  
      3      6  
>> n1=names(1,:)   
n1 =  
Sue  
>> length(n1)  
ans =  
      6  
>> deblank(n1)  
ans =  
Sue  
>> length(ans)  
ans =  
      3
```



**Come
implementare il
deblank usando
un for loop?**

mydeblank

```
function [ s1 ] = mydeblank( s )
%Elimina gli spazi bianchi dalla fine della stringa
%Stringa in s1
n=length(s);
if( size(s,1)==1 || size(s,2)==1 )
    for i=1:n
        if s(i) ~= ' '
            s1(i)=s(i);
        else
            break;
        end
    end
else
    s1=s;
    disp('Impossibile rimuovere spazi bianchi');
end
end
```

Homework

- La funzione appena definita “mydeblank” non funziona se gli spazi bianchi sono contenuti prima della fine della stringa.
- Esempio:
 - `mydeblank('this string is tricky ')`
- Ritournerà `'this'` invece di `'this string is tricky'`
- Fatelo voi!

Creare stringhe personalizzate (formattate)

- `sprintf`

```
>> s2=sprintf('il valore è %.2f',pi)
```

```
s2 =
```

```
Il valore è 3.14
```

Con `sprintf` è possibile creare stringhe sulla base dei valori delle variabili e salvare il tutto ancora in una variabile.

Consultate l'help di Matlab per ulteriori dettagli

Conversione maiuscole/minuscole e viceversa

```
>> s1='Hello'
```

```
s1 =
```

```
Hello
```

```
>> s=upper(s1)
```

```
s =
```

```
HELLO
```

```
>> s3=lower(s)
```

```
s3 =
```

```
hello
```

Confronto di stringhe (1)

```
>>s1='Hello';  
>>s2='Hillo';  
>> s1==s2  
ans =  
      1      0      1      1      1
```

```
>> sum(s1==s2)==length(s1)
```

```
ans =  
      0
```

```
>> s2(2)='e'
```

```
s2 =  
Hello
```

```
>> sum(s1==s2)==length(s1)
```

```
ans =  
      1
```

```
>>
```



Se s1 è uguale ad s2 la variabile “ans” ritorna un vettore riempito di “1”. Perciò, la somma degli elementi del vettore sarà uguale alla lunghezza del vettore.

Confronto di stringhe (2)

```
>> strcmp(s1, s2)
```

```
ans =
```

```
1
```

```
>> strcmp(s2, s1)
```

```
ans =
```

```
1
```

```
>> s1=[s1 '3456']
```

```
s1 =
```

```
Hello3456
```

```
>> strcmp(s1, s2)    %è sensibile alla lunghezza
```

```
ans =
```

```
0
```

Homework

- Scrivere una function che si comporta come strcmp. Dall'help di Matlab si evince che:

`strcmp` Compare strings.

`TF = strcmp(S1,S2)` compares the strings `S1` and `S2` and returns logical 1 (true) if they are identical, and returns logical 0 (false) otherwise.

La funzione prevede la gestione di stringhe di taglia differente.

Ricerca stringhe

```
>> s1='This is a string for test';
```

```
>> strfind(s1, 's')
```

```
ans =
```

```
         4         7        11        24
```

```
>> strfind(s1, 'for')
```

```
ans =
```

```
18
```

La funzione ritorna la posizione del carattere , ossia la posizione in s1 del carattere s...

...oppure la posizione iniziale della sottostringa d'ingresso

Homework

- Scrivere una funzione che si comporti come `strfind`

String Array

- Tutto quanto visto sinora ci fa capire che:
- Matlab vede le matrici di stringhe come matrici di caratteri
- Infatti valgono i “constraint” dimensionali classici
- Esiste un ulteriore tipo: string array da r2017a
- Tra apici doppi

Algoritmi di ordinamento

I procedimenti di ordinamento dei dati rientrano nell'insieme dei problemi classici dell'informatica.

PROBLEMA

Partendo da una sequenza di elementi omogenei è necessario ordinarli secondo un criterio prestabilito, p.e. ordine crescente.



ALGORITMO



RISULTATO

Si costruisce una permutazione della sequenza originaria tale che gli elementi costituenti soddisfano il criterio di ordine.

Algoritmi di ordinamento (2)

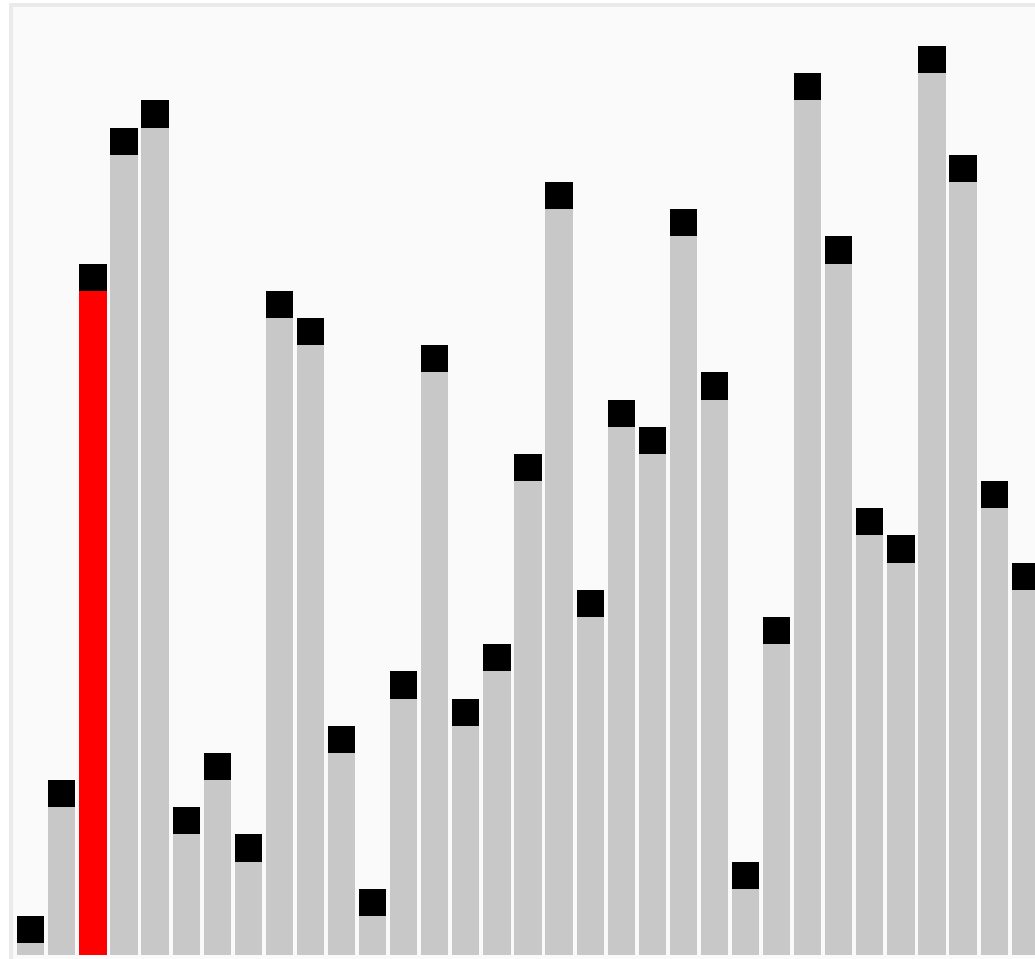
Le caratteristiche basilari di un generico algoritmo di ordinamento:

- Procedimento *iterativo*
- In generale sono composti da:
 - Ripetizione (es. ciclo)
 - Criterio di ordinamento (es. confronto)

Approfondimenti:

https://it.wikipedia.org/wiki/Algoritmo_di_ordinamento

Ordinamento



Fonte: https://it.wikipedia.org/wiki/Bubble_sort

Algoritmi di ordinamento (3)

Analizziamo i seguenti algoritmi elementari su sequenze numeriche da ordinare in modo ascendente:

- Ordinamento per selezione (**Selection Sort**)
ad ogni iterazione i si seleziona l'elemento più piccolo che viene spostato nella posizione i -esima.
- Ordinamento ad inserimento (**Insertion Sort**)
Considerando di avere già ordinato una porzione della sequenza da 1 a k , allora alla iterazione $k+1$ si inserisce l'elemento della porzione non ordinata nella posizione corretta di quella ordinata facendo “scorrere” la sequenza verso destra.
- Ordinamento a “bolle” (**Bubble sort**)
Viene effettuato il confronto tra elementi adiacenti: la relativa coppia viene ordinata

Ordinamento per selezione

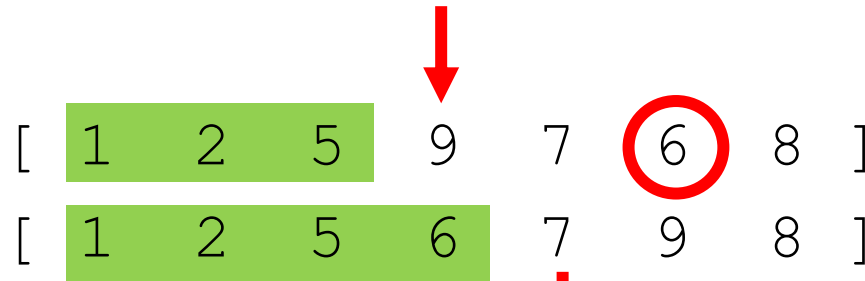
- Data la sequenza di numeri di lunghezza n , si assume che questa sia stata parzialmente ordinata, in ordine crescente, fino all'elemento $k-1$.
- La sotto-sequenza da k (4) a n (7) è da ordinare.

[	1	2	5	9	7	6	8]
$i =$	1	2	3	<u>4</u>	5	6	7

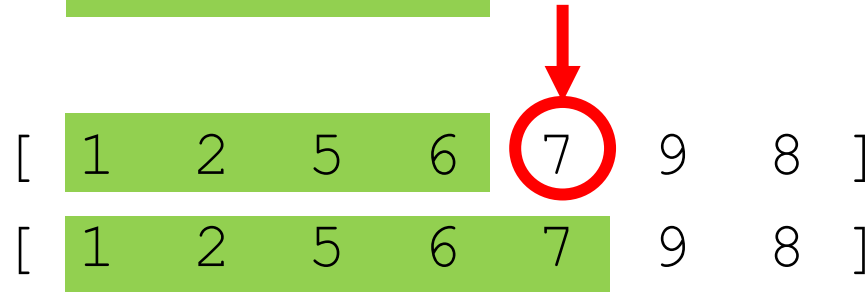
1. Trovare il minimo tra gli elementi della sotto-sequenza non ordinata (con i da k a n)
2. Sostituire l'elemento in posizione k con il minimo trovato
3. Incrementare k
4. Se $k < n$ ripetere dal passo 1
5. Fine

Ordinamento per selezione (2)

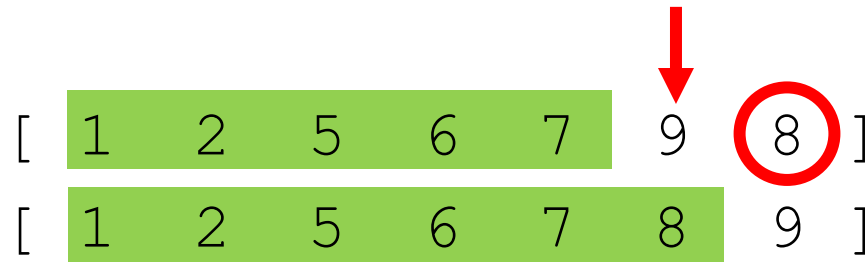
I. $k = 4$



II. $k = 5$



III. $k = 6$



IV. $k = 7$

Fine: sequenza ordinata

Selection Sort

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Ordinamento per selezione (3)

```
function v = SelectionSort( v )
%Ordina il vettore v con l'algoritmo per selezione
%INPUT: vettore di numeri
%OUTPUT: vettore di numeri in ordine ascendente

for k = 1 : ( length( v ) - 1 )
    indiceMin = k; %inizializzazione della posizione del minimo
    for j = (k + 1):length( v ) % trova il minimo tra gli n-k
        if v( j ) < v( indiceMin )
            indiceMin = j; %posizione dell'elemento di valore minimo
        end
    end
    temp = v( indiceMin ); % scambia la posizione del minimo
    v( indiceMin ) = v( k );
    v( k ) = temp;
end

end
```

Ordinamento ad inserimento

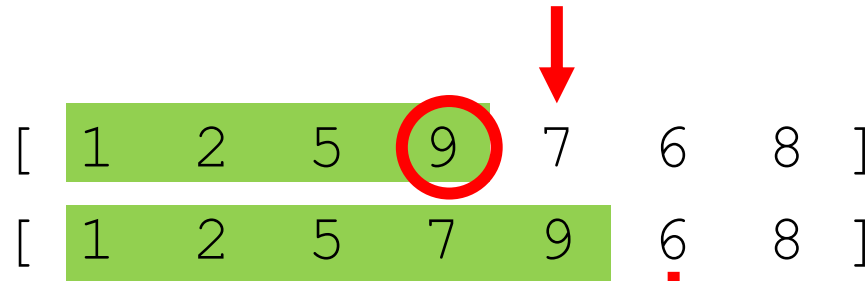
- Data la sequenza di numeri di lunghezza n , si assume che questa sia stata parzialmente ordinata, in ordine crescente, fino all'elemento $k-1$.
- La sotto-sequenza da k (5) a n (7) è da ordinare.

	[	1	2	5	9	7	6	8]
$i =$		1	2	3	4	<u>5</u>	6	7

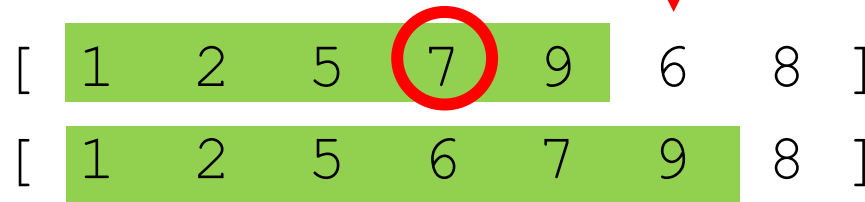
1. Considerato l'elemento k , individuare la sua posizione p con $1 < p < (k-1)$
2. Se $p < k$ inserire l'elemento nella posizione p e spostare gli altri (scorrimento verso destra)
3. Incrementare k
4. Se $k \leq n$ ripetere dal passo 1
5. Fine

Ordinamento ad inserimento (2)

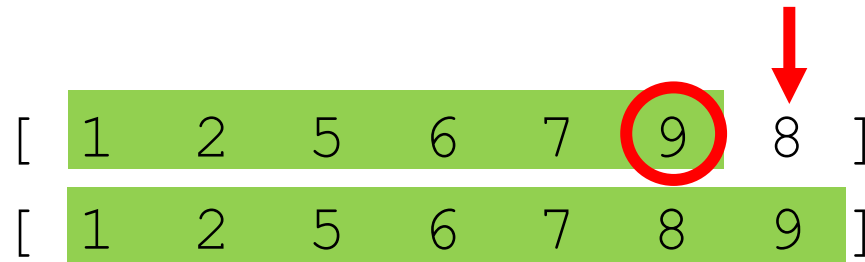
I. $k = 5$



II. $k = 6$



III. $k = 7$



IV. $k > 7$

Fine: sequenza ordinata

Insertion Sort

6 5 3 1 8 7 2 4

Ordinamento ad inserimento (3)

```

function v = InsertionSort( v )
%Ordina gli elementi di un vettore con l'algoritmo ad
%inserimento
%INPUT: vettore di numeri
%OUTPUT: vettore di numeri in ordine ascendente

for k = 2:length( v )
    x = v( k );           %elemento da posizionare
    for j = 1:k           % n questo ciclo cerca la posizione
        if v( j ) > x
            break;        %posizione trovata!
        end
    end
    if j < k               % è necessario spostare gli elementi
        for i = ( k - 1 ) : -1 : j
            v( i + 1 ) = v( i );
        end
        v( j ) = x;
    end
end
end

```

Bubble sort

- Data la sequenza di numeri di lunghezza n , si confrontano due elementi contigui e quello minore viene posizionato prima. Se avviene uno scambio, questo evento viene registrato in una variabile booleana (p.e. `scambio=TRUE`)
1. Considerata la coppia di elementi in posizione $(k, k+1)$, ne vengono confrontati i valori
 2. Se il valore in $k+1$ è minore del valore in k le posizioni vengono scambiate e `scambio=TRUE`
 3. Se lo scambio è avvenuto incrementare k , ripristinare `scambio=FALSE`, ripetere dal passo 1
 4. L'algoritmo finisce se si scorre tutta la sequenza e `scambio==FALSE` allora **FINE** perché non ci sono più "bolle" da spostare (sequenza ordinata)

Bubble sort (2)

(1)

7	2	4	5	3	1
---	---	---	---	---	---

2 7

4 7

5 7

3 7

1 7

(2)

2	4	5	3	1	7
---	---	---	---	---	---

2 4

4 5

3 5

1 5

5 7

(3)

2	4	3	1	5	7
---	---	---	---	---	---

2 4

3 4

1 4

(4)

2	3	1	4	5	7
---	---	---	---	---	---

2 3

1 3

(5)

2	1	3	4	5	7
---	---	---	---	---	---

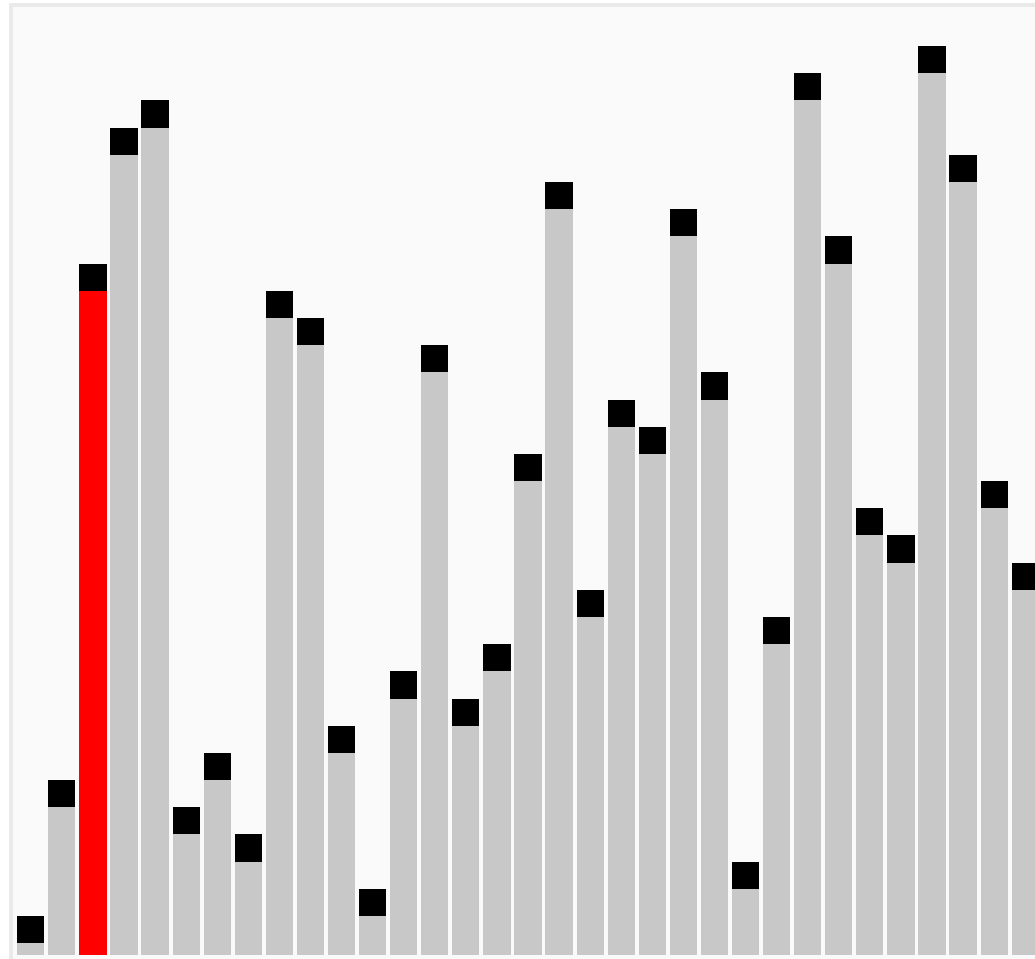
1 2

1	2	3	4	5	7
---	---	---	---	---	---

Bubble sort

6 5 3 1 8 7 2 4

Bubble sort



Fonte: https://it.wikipedia.org/wiki/Bubble_sort

Bubble sort (3)

```
function v = BubbleSort( v )
% Ordina gli elementi di un vettore utilizzando l'algoritmo
% BubbleSort
% INPUT: vettore di numeri v
% OUTPUT: vettore di numeri in ordine ascendente

for k = 1 : ( length( v ) - 1 )
    scambio = false;
    for j = 2 : ( length( v ) - ( k-1 ) )
        %verifica se gli elementi contigui sono ordinati...
        if v( j-1 ) > v( j )
            temp = v( j ); %...altrimenti scambiali
            v( j ) = v( j-1 );
            v( j-1 ) = temp;
            scambio = true;
        end
    end
    if ~scambio % nessuno scambio => vettore ordinato
        break; % interrompi il ciclo
    end
end

end

end
```

Esercizi

1. Progettare ed implementare una funzione di ordinamento **decrescente** che utilizzi l'algoritmo dell'inserimento opportunamente modificato.
2. Adattare l'algoritmo di ordinamento **Bubble Sort** sostituendo il costrutto **for** con **while**.
3. Modificare le funzioni di ordinamento presentate per calcolare il numero totale di iterazioni e il numero di spostamenti per ciascun algoritmo. Considerare come variano queste due quantità per ciascun algoritmo nelle seguenti situazioni:
 1. caso migliore (sequenza già ordinata),
 2. caso peggiore (sequenza con ordine inverso),
 3. caso tipico.Si possono trarre delle conclusioni?