FONDAMENTI DI INFORMATICA

NONA LEZIONE

Prof. Alfredo Accattatis

(accattatis@ing.uniroma2.it)

Tutor: Prof. Vittorio Calafiore

(vcalafio@gmail.com)

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 un permutazione della sequenza originaria tale che gli elementi costituenti soddisfano il criterio di ordine.

Algoritmi di ordinamento (2)

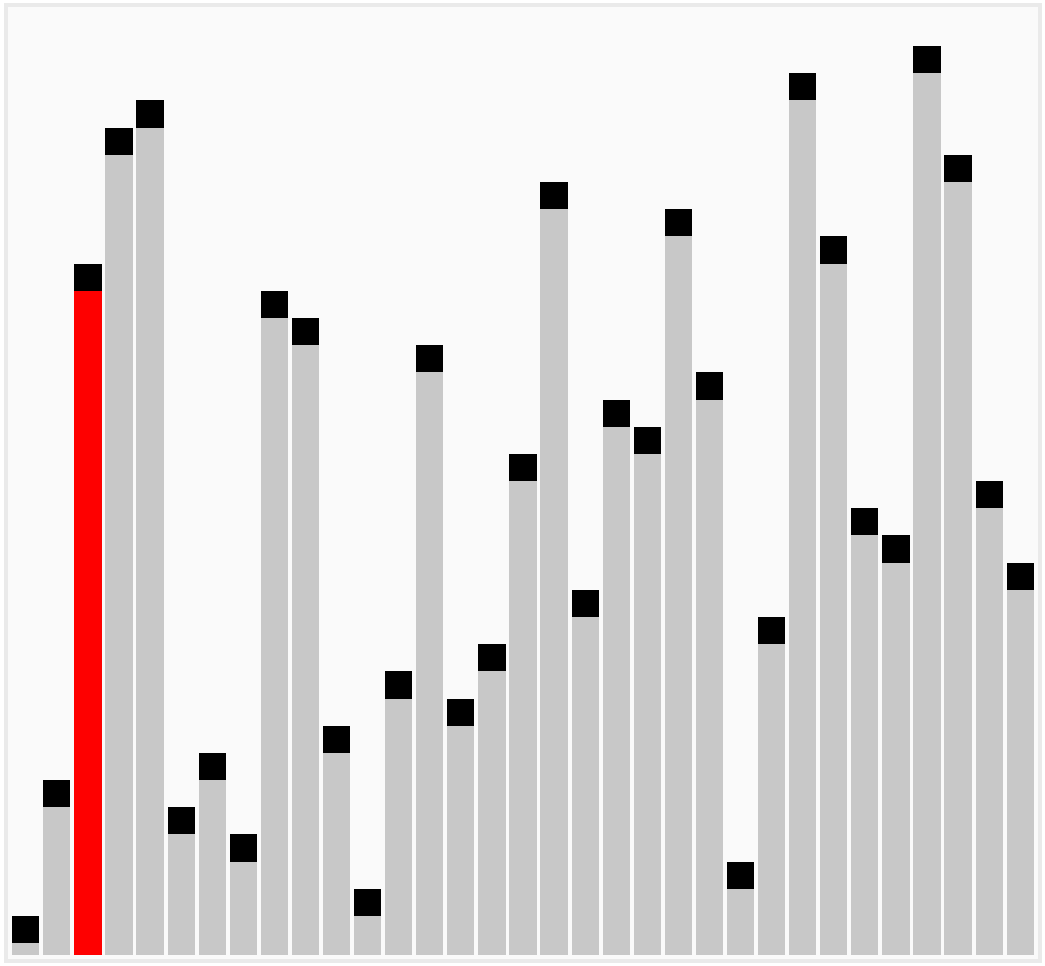
Le caratteristiche basilari di un generico algoritmo di ordinamento:

- 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

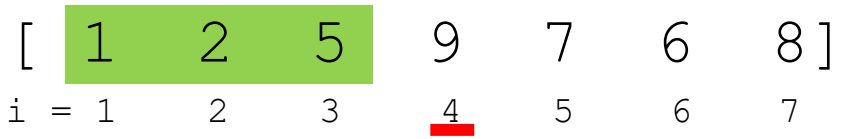
Algoritmi di ordinamento (3)

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

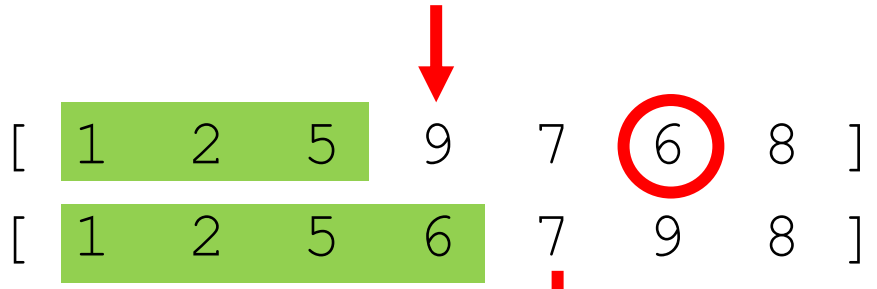
- Data la sequenza di numeri di lunghezza `n`, si assume che questa sia stata parzialmente ordinata, in ordine crescente, fino all'elemento `k-1`.
- La sotto-sequenza da `k(4)` a `n(7)` è da ordinare.



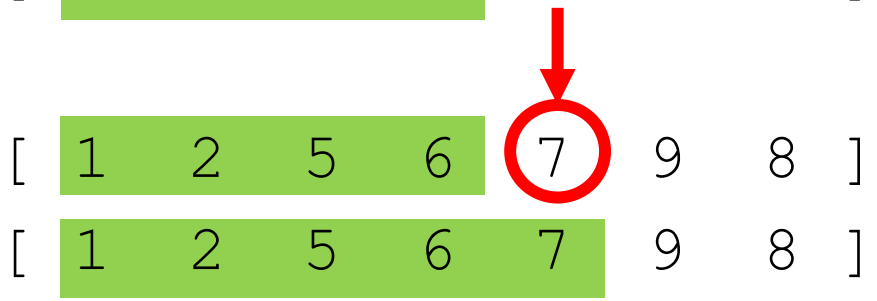
1. Trovare il minimo tra gli elementi della sotto-sequenza non ordinata (con `i` da `k` a `n`)
2. Sostituire l'elemento in posizione `k` con il minimo trovato
3. Incrementare `k`
4. Se `k < n` ripetere dal passo 1
5. Fine

Ordinamento per selezione (2)

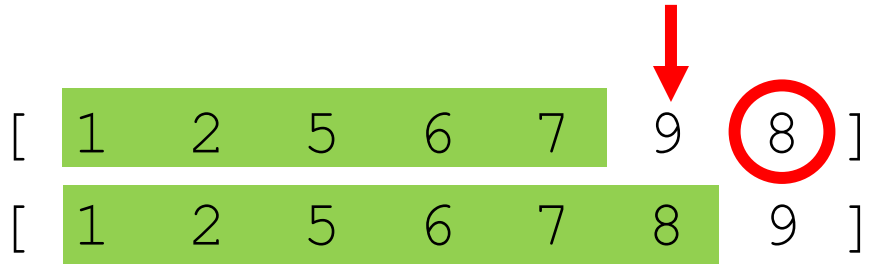
I. k = 4



II. k = 5



III. k = 6



IV. k = 7

Fine: sequenza ordinata

Selection Sort

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

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

Ordinamento per selezione (3)

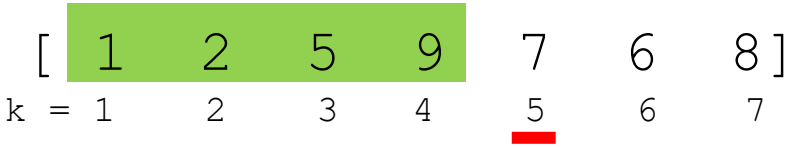
```
function v = SelectionSort( v )
%Ordina il vettore v con l'algoritmo per selezione
%INPUT: vettore di numeri
%OUTPUT: vettore di numeri in ordine ascendente

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

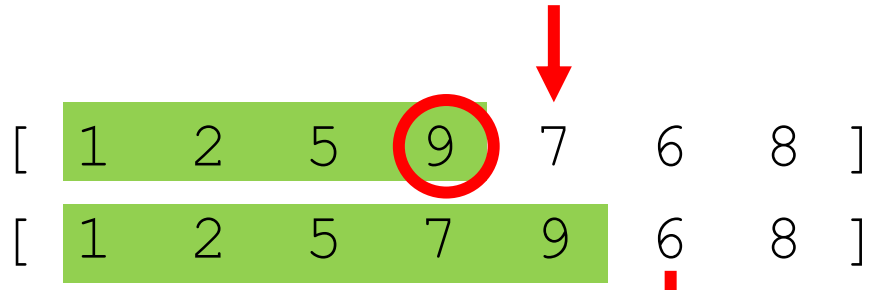
- Data la sequenza di numeri di lunghezza `n`, si assume che questa sia stata parzialmente ordinata, in ordine crescente, fino all'elemento `k-1`.
- La sotto-sequenza da `k(5)` a `n(7)` è da ordinare.



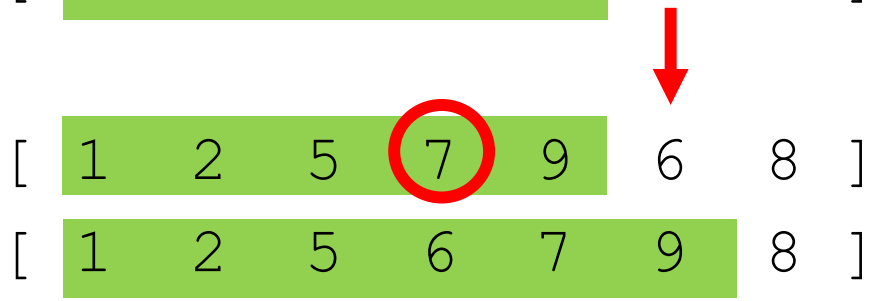
1. Considerato l'elemento `k`, individuare la sua posizione `p` con `1 < p < (k-1)`
2. Se `p < k` inserire l'elemento nella posizione `p` e spostare gli altri (scorrimento verso destra)
3. Incrementare `k`
4. Se `k <= n` ripetere dal passo 1
5. Fine

Ordinamento ad inserimento (2)

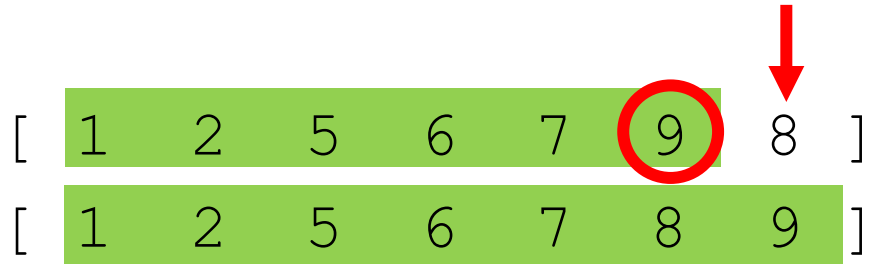
I. k = 5



II. k = 6



III. k = 7



IV. k > 7

Fine: sequenza ordinata

Insertion Sort

6 5 3 1 8 7 2 4

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

Ordinamento ad inserimento (3)

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

for k = 2:length( v )
    x = v( k );     % Elemento da posizionare
    for j = 1:k     % 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
```

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 boolena (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'algorimo 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

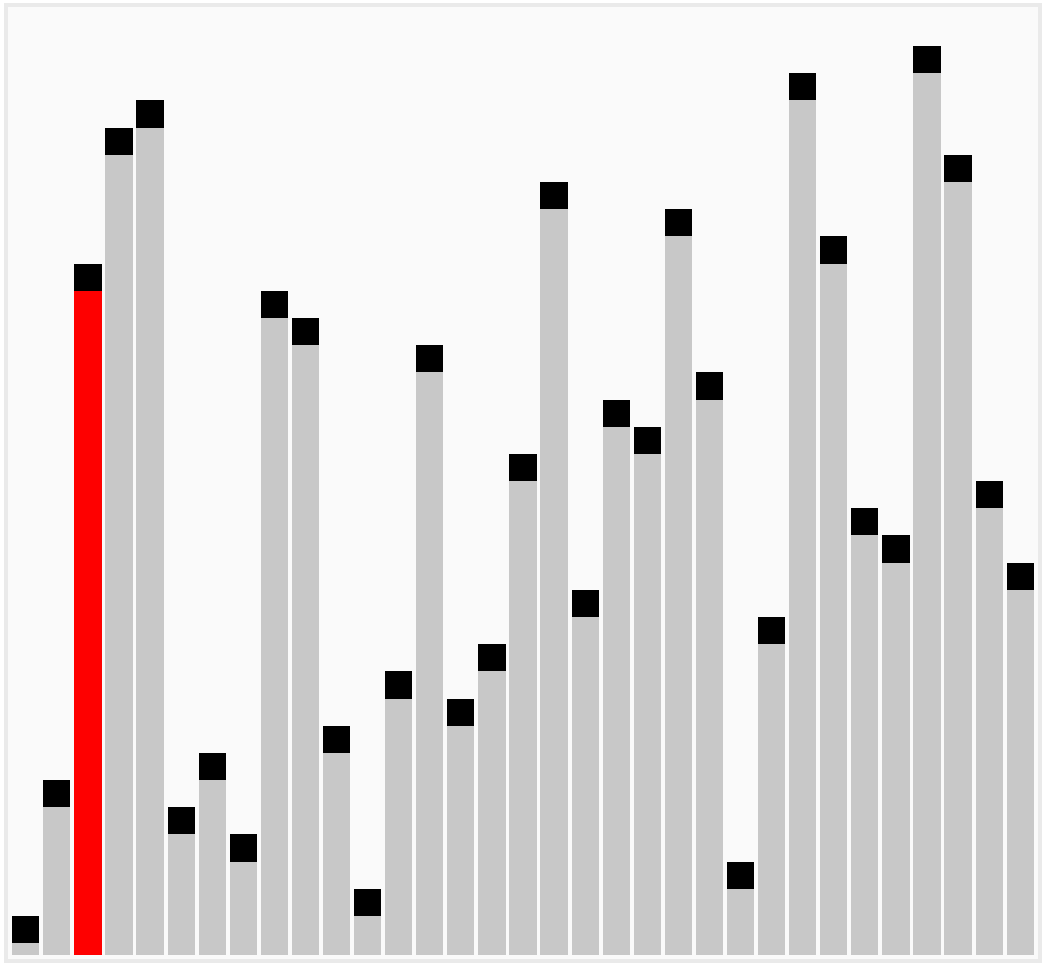
1	2	3	4	5	7

Bubble sort

6 5 3 1 8 7 2 4

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

Bubble sort



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

Bubble sort (3)

```
function v = BubbleSort( v )
% Ordina gli elementi di un vettore utilizzando BubbleSort
% INPUT: vettore di numeri non ordinato
% 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 principale
    end
end

end
```

Esercizi

1. Progettare ed implementare una funzione di ordinamento ***decrescente*** che utilizzi l'algoritmo ad 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?

Matlab: vantaggi

- Visual Analyser: anni di lavoro, milioni di linee di codice…
- Semplice demo in Matlab: decine di linee…minuti di lavoro!

```
function VA()

Fs = 44100;             % Frequenza di campionamento
T = 1/Fs;               % Periodo
L = 4096;               % Lunghezza del segnale
t = (0:L-1)*T;          % Asse dei tempi

h=struct;
h.stop = false;

%Finestra disegno principale
    h.MainFigure = figure('Name','VA Matlab', ...
                    'HandleVisibility','on', ...
                    'Position',[40 400 1100 300 ],...
                    'Toolbar','figure','Menubar','figure',...
                    'CloseRequestFcn',@closeGUI);

% Bottone di stop acquisizione
    h.btnBrowse=uicontrol(h.MainFigure,'Style','pushbutton',...
                'String','Stop', ...
                'Units', 'normalized', ...
                'Callback', @btnOpen_Callback);
%                'Position',[.895 .15 .1 .7], ...
%                'Position',[.1 .1 .1 .7], ...
```

```
disp('Dire qualcosa adesso.')

while h.stop == false

    %Acquisizione campioni audio
    acquiredAudio = deviceReader();

    % Disegno campioni audio
    subplot(1,2,1)
    p = plot(acquiredAudio);
    p.Color = "green";
    title('Segnale nel dominio del tempo')
    ylim([-0.1 0.1]);
    drawnow;

    % Trasformata di Fourier del segnale acquisito
    Y = fft(acquiredAudio);

    % Aggiustamenti
    P2 = abs(Y/L);
    P1 = P2(1:L/2+1);
    P1(2:end-1) = 2*P1(2:end-1);
    f = Fs*(0:(L/2))/L;

    % Disegno spettro di ampiezza
    subplot(1,2,2)
    p = semilogx(f,P1);
    p.Color = "red";
    title('Spettro di ampiezza del segnale')
    xlabel('f (Hz)')
    ylabel('|P1(f)|')
    drawnow;

    if h.stop
        break;
    end
end
```

```
disp('Fine acquisizione.')
release(deviceReader)

% Funzione usata per fermare l'acquisizione dati
function btnOpen_Callback(hObject, eventdata)
        h.stop = true;
end

% Funzione richiamata quando si chiude la finestra
function closeGUI(src, evnt)
   h.stop = true;
   release(deviceReader)
   delete(h.MainFigure)
   disp('Fine programma.')
end


end
```

- Vi ricordate la slide introdotta nelle prime lezioni che accennava al….

- …TEOREMA del CAMPIONAMENTO?

Teorema del campionamento

(Shannon Nyquist)

Lo tratteremo in maggior dettaglio. Tramite esso è possibile tradurre qualsiasi grandezza «del mondo esterno» in un formato comprensibile al mondo degli elaboratori (numeri codificati in binario)

E' fondamentale ed ha reso l'Informatica uno strumento universale

Spettro di un segnale

Un segnale periodico «generale» (non sinusoidale) risulta composto da una sommatoria di segnali sinusoidali. Ossia esso può essere matematicamente rappresentato come una somma (infinita) di segnali sinusoidali, a diverse ampiezze e frequenza: **sviluppo in serie di Fourier**.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(nt) + b_n \sin(nt))$$

$$a_n = \int_{-\pi}^{+\pi} f(t) \cos nt \, dt$$

$$b_n = \int_{-\pi}^{+\pi} f(t) \sin nt \, dt$$

Spettro di un segnale

Le sinusoidi risultano essere, secondo questa scomposizione, dei segnali «atomici» non scomponibili in ulteriori segnali. Sono le ARMONICHE

Si definisce SPETTRO DI UN SEGNALE

o

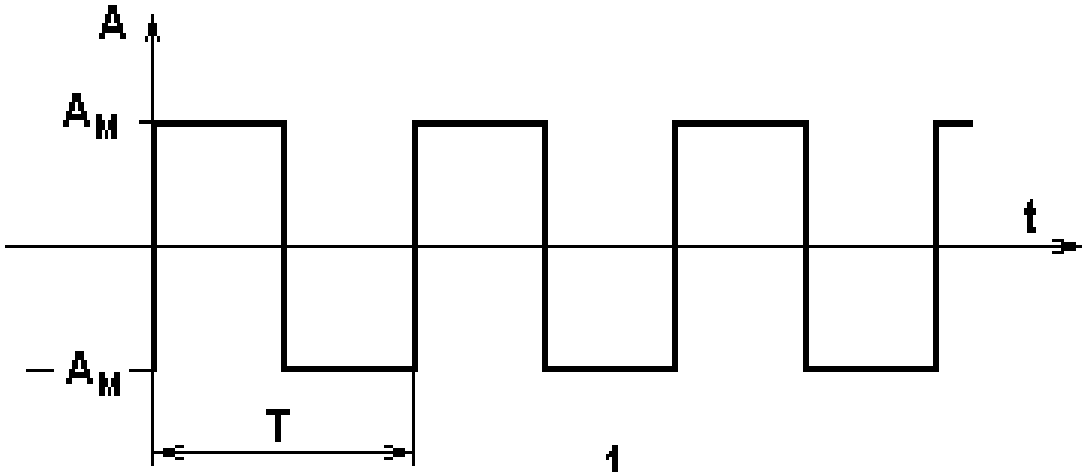
rappresentazione nel dominio della frequenza,

il grafico di ampiezza e fase

ottenuto scomponendo il segnale nelle sue ARMONICHE

detta anche BANDA del segnale

Esempio : onda quadra



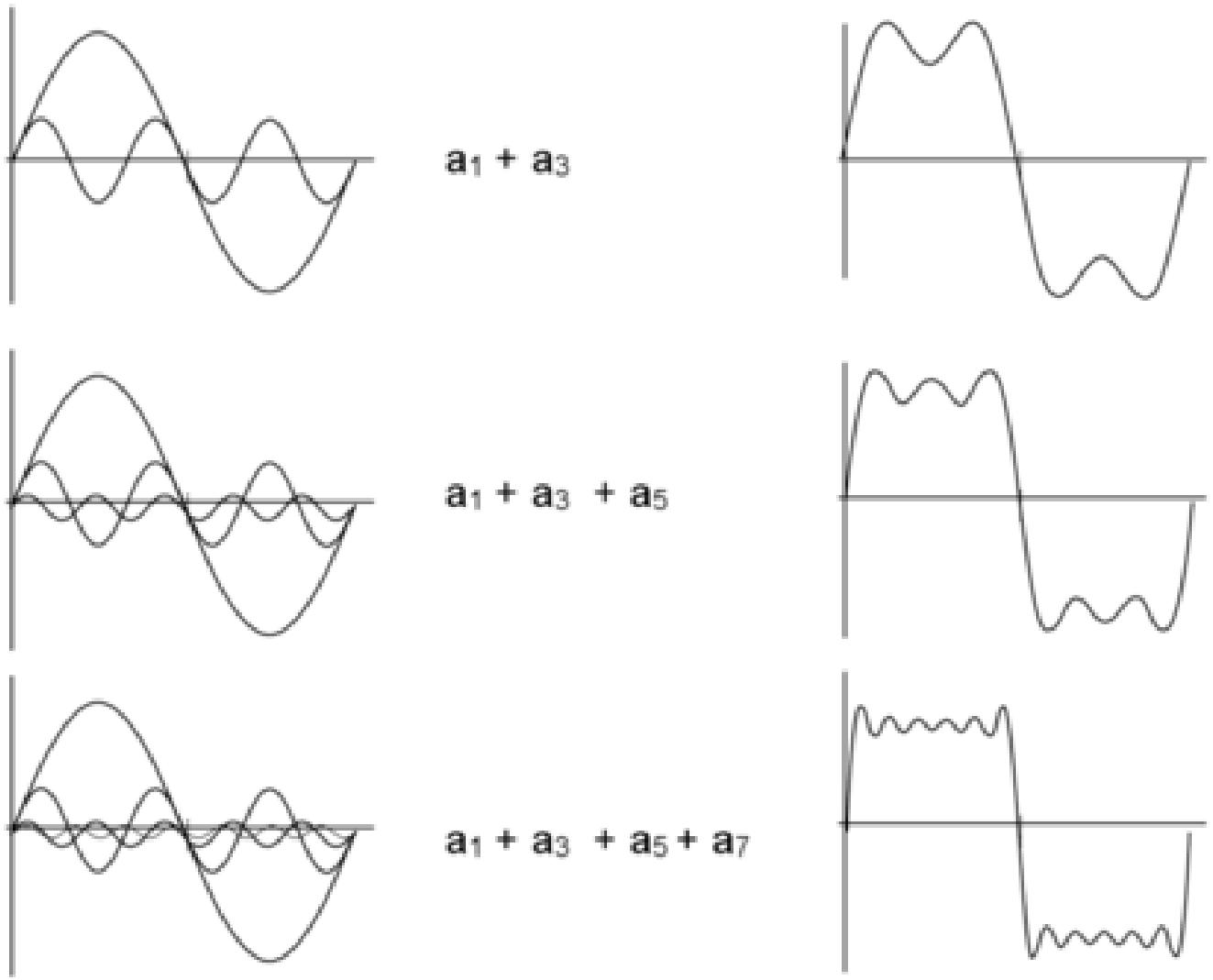
$f_o = \frac{1}{T}$

T = periodo

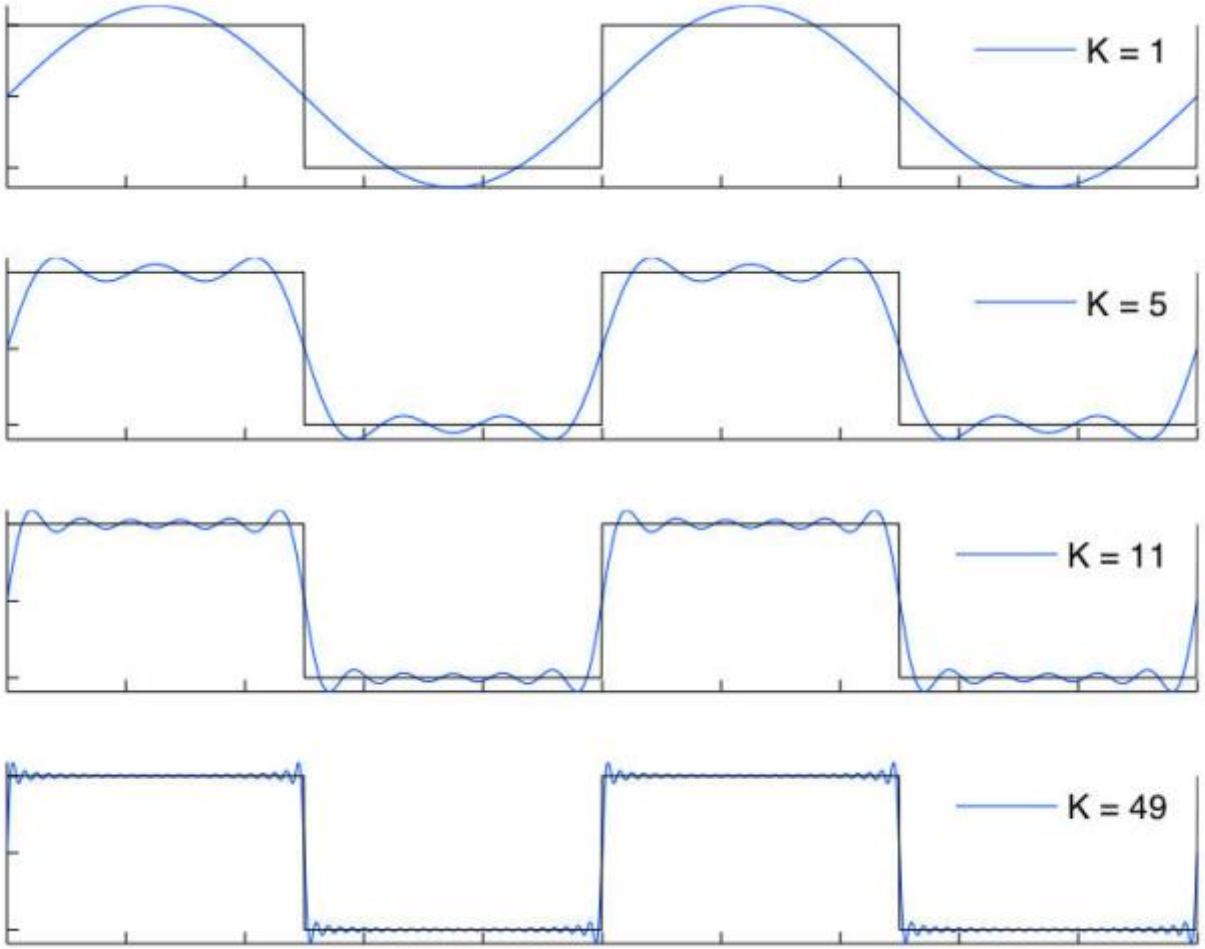
f_o = frequenza

A_M = valore massimo

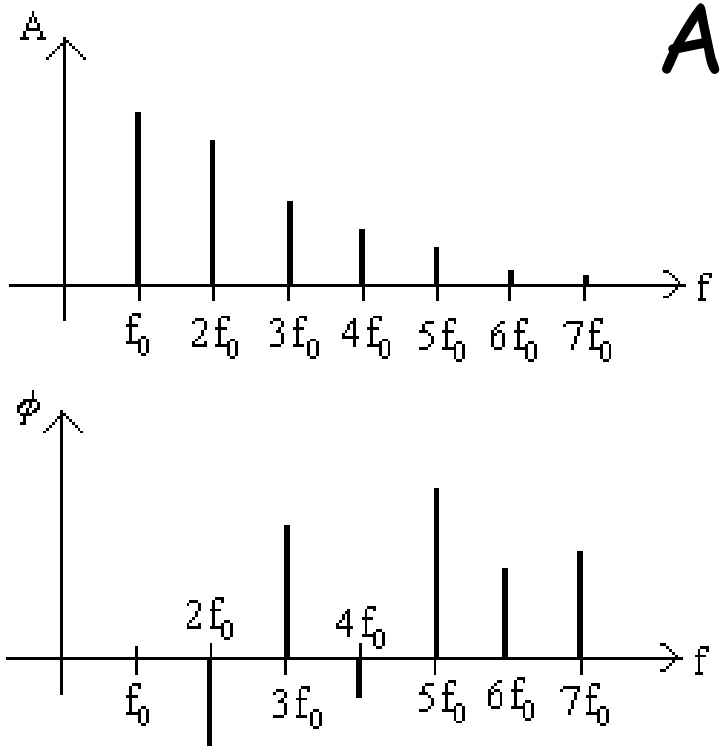
Armoniche componenti



Armoniche componenti



Banda di un segnale, Spettro : formato da Ampiezza e fase



Ampiezza

Fase

Trasformata di FOURIER

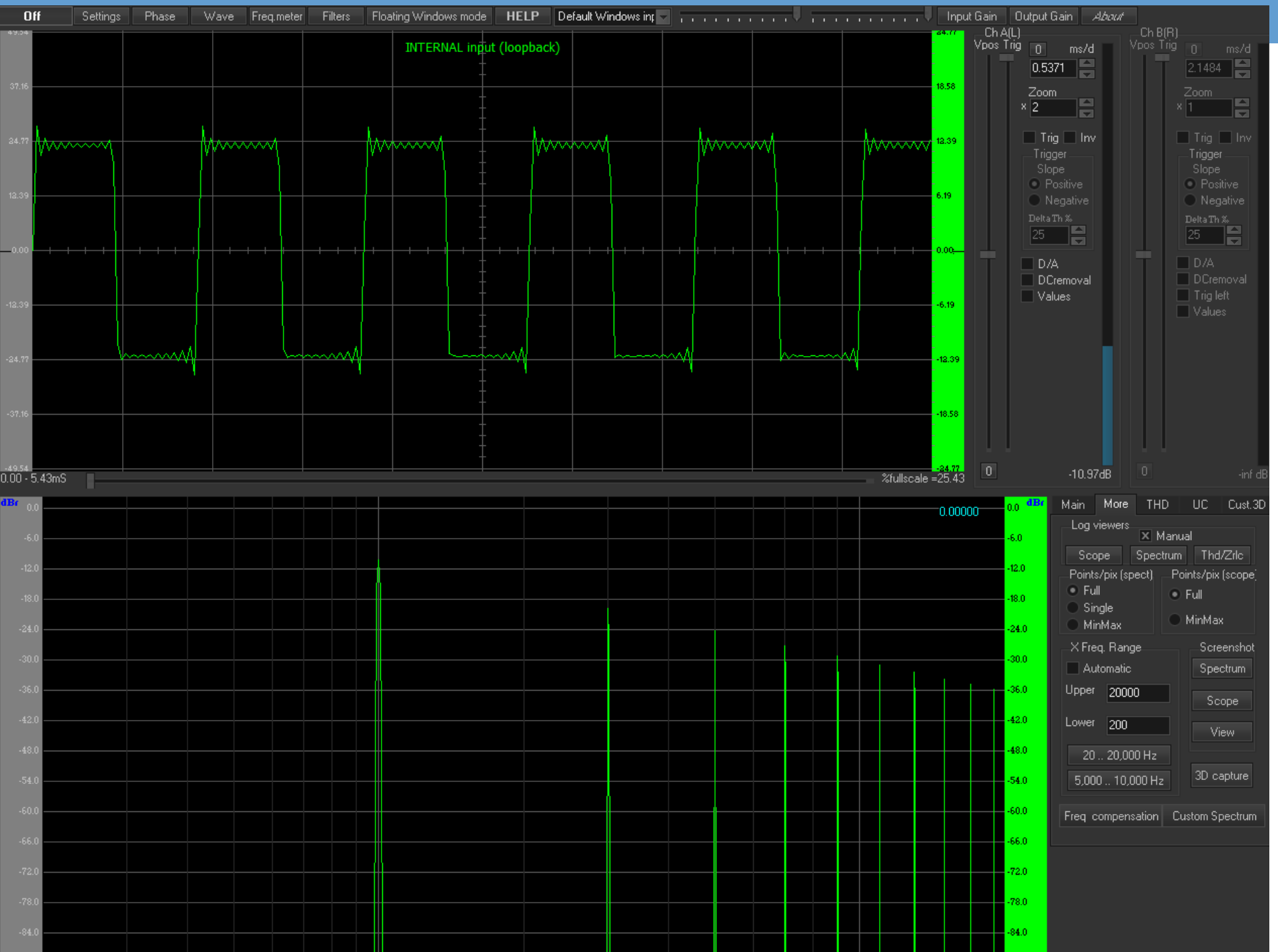
- *Generalizzazione* dello sviluppo in serie: ogni segnale, anche NON periodico può vedersi come somma di sinusoidi. E una trasformazione matematica che consente di ricavare, dato il segnale nel «dominio del tempo» s(t) la funzione X(f), ossia ricavare lo SPETTRO del segnale, detta anche *rappresentazione nel dominio della frequenza*

- X(f) = F(s(t))

- La DFT è la versione per segnali tempo discreto:

$$X_q = \mathcal{F}_d(x_n) = \sum_{k=0}^{N-1} x_k e^{-i\frac{2\pi}{N}kq} \qquad q = 0, 1, \ldots, N-1$$

- Una versione veloce: FFT Fast Fourier Transform.


Off
Settings
Phase
Wave
Freq.meter
Filters
Floating Windows mode
HELP
Default Windows inp
Input Gain
Output Gain
About
INTERNAL input (loopback)
Ch A(L)
Vpos Trig
0
ms/d
0.5371
Zoom
x 2
Trig
Inv
Trigger
Slope
Positive
Negative
Delta Th %
25
D/A
DCremoval
Values
-10.97dB
Ch B(R)
Vpos Trig
0
ms/d
2.1484
Zoom
x 1
Trig
Inv
Trigger
Slope
Positive
Negative
Delta Th %
25
D/A
DCremoval
Trig left
Values
-inf dB
0.00 - 5.43mS
%fullscale =25.43
dBr
0.00000
Main
More
THD
UC
Cust.3D
Log viewers
Manual
Scope
Spectrum
Thd/Zrlc
Points/pix (spect)
Full
Single
MinMax
Points/pix (scope)
Full
MinMax
X Freq. Range
Automatic
Upper
20000
Lower
200
20 .. 20,000 Hz
5,000 .. 10,000 Hz
Screenshot
Spectrum
Scope
View
3D capture
Freq compensation
Custom Spectrum

Teorema del campionamento

- Ora possiamo finalmente enunciare il teorema del campionamento
- Dato un segnale analogico tempo continuo a banda limitata, esso può essere completamente rappresentato (ed eventualmente ricostruito) da un segnale tempo discreto da esso derivato per mezzo di un'operazione di campionamento

(misura cadenzata)

Teorema del campionamento

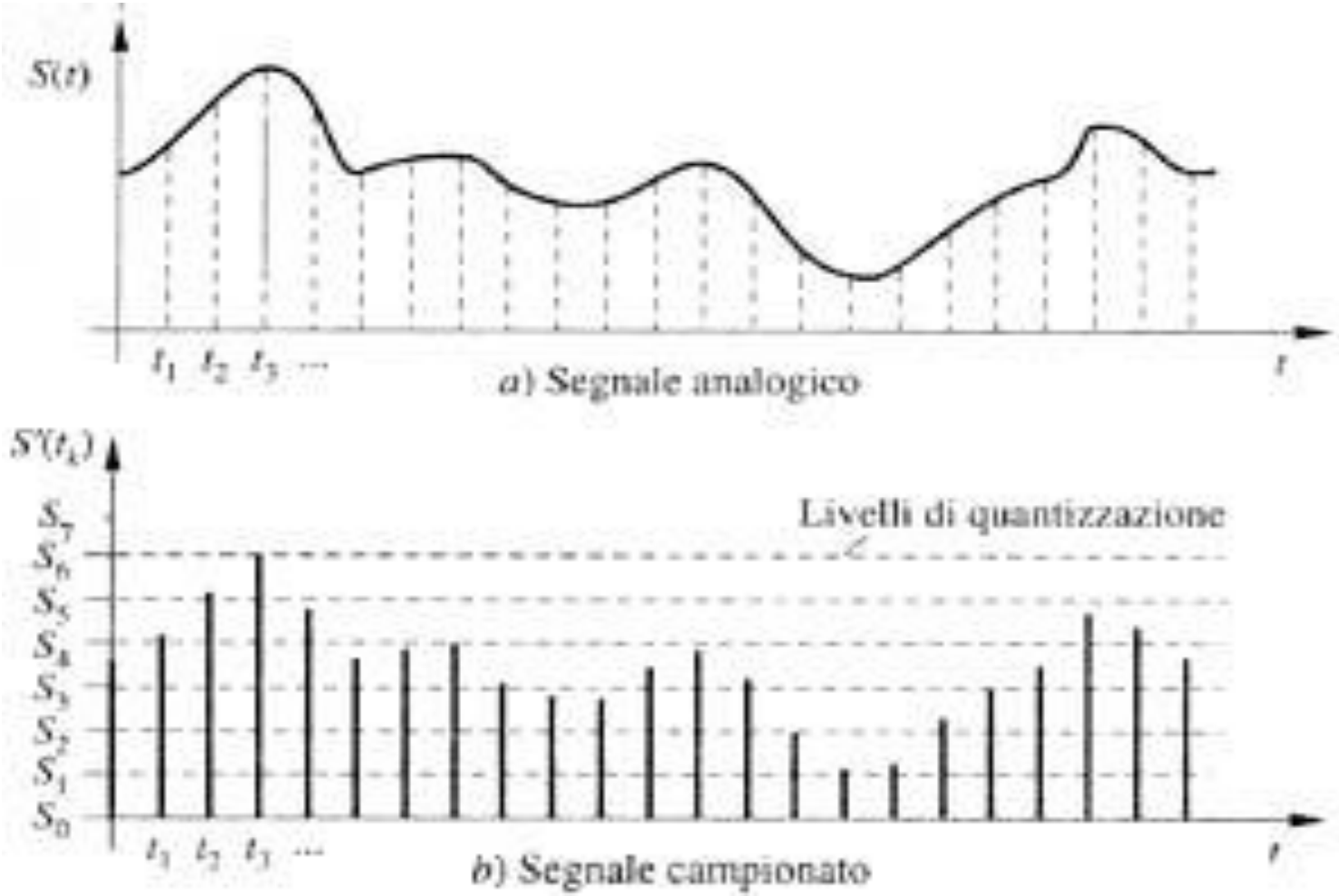
- sia *X(t)* un segnale a *banda limitata*, ossia la trasformata di Fourier *X(f)* sia uguale a zero per | f | > B (dove B è la banda del segnale). Ossia, sia *X(t)* un segnale il cui contenuto armonico vari da zero a un massimo (per esempio da 0 a 20000 Hz)
- sia la *frequenza di campionamento* maggiore ed almeno uguale al doppio di B, ossia si prelevino *campioni* del segnale con una frequenza almeno doppia di quella della massima frequenza presente nel segnale (ossia B).
- Allora (tesi)….

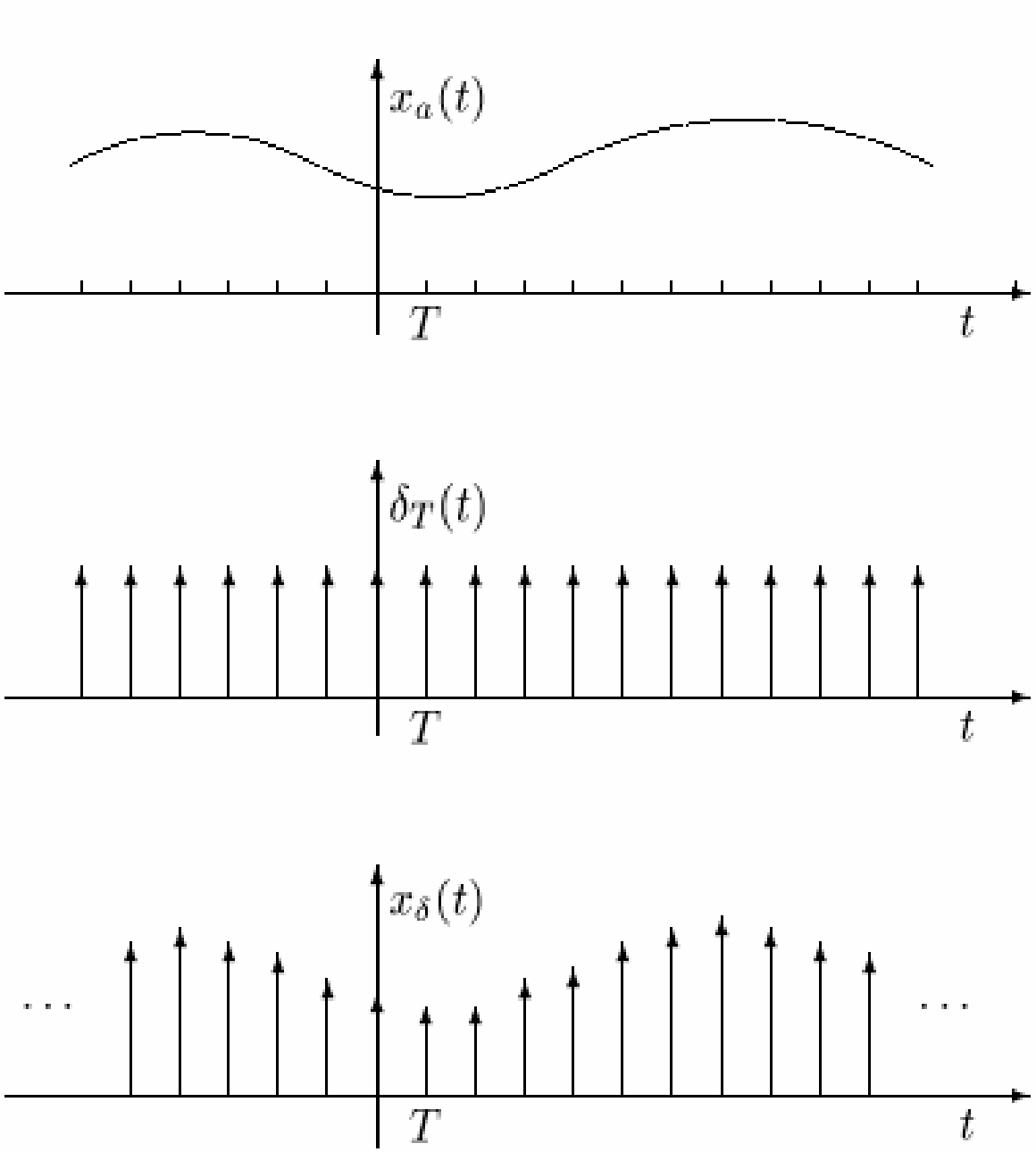
Teorema del campionamento

- il segnale *X(t)* è rappresentato *esaustivamente* dai suoi campioni (senza errori)
- il segnale può essere ricostruito con un filtro passa-basso avente frequenza di taglio Ft tale che B<Ft
- OPPURE
- il segnale *X(t)* può essere ricostruito a partire dai suoi campioni con lo sviluppo in serie definito dalla seguente relazione:
- $x(t) = \sum_{1,n} x(nT_c) sinc(\pi \frac{t - nT_c}{T_c})$

Dove: $sinc(x) = \frac{\sin(x)}{x}$

Campionare (misura cadenzata):

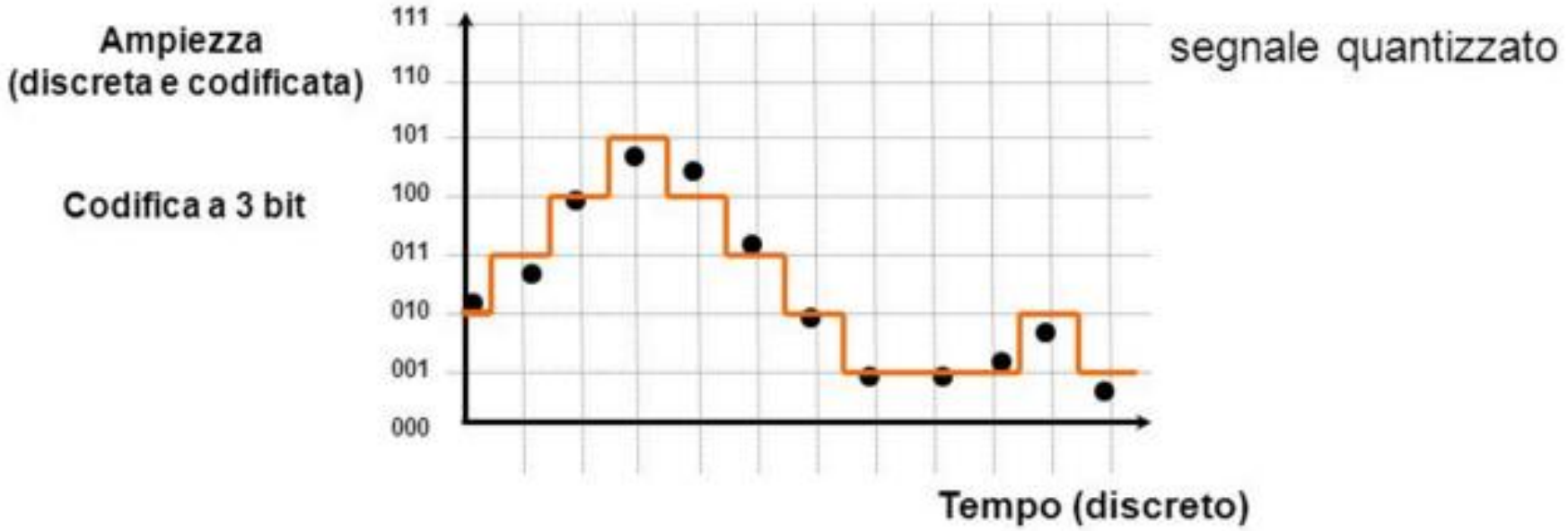



$x_a(t)$
T
t
$\delta_T(t)$
T
t
$x_\delta(t)$
$\ldots$
$\ldots$
T
t

Quantizzazione...

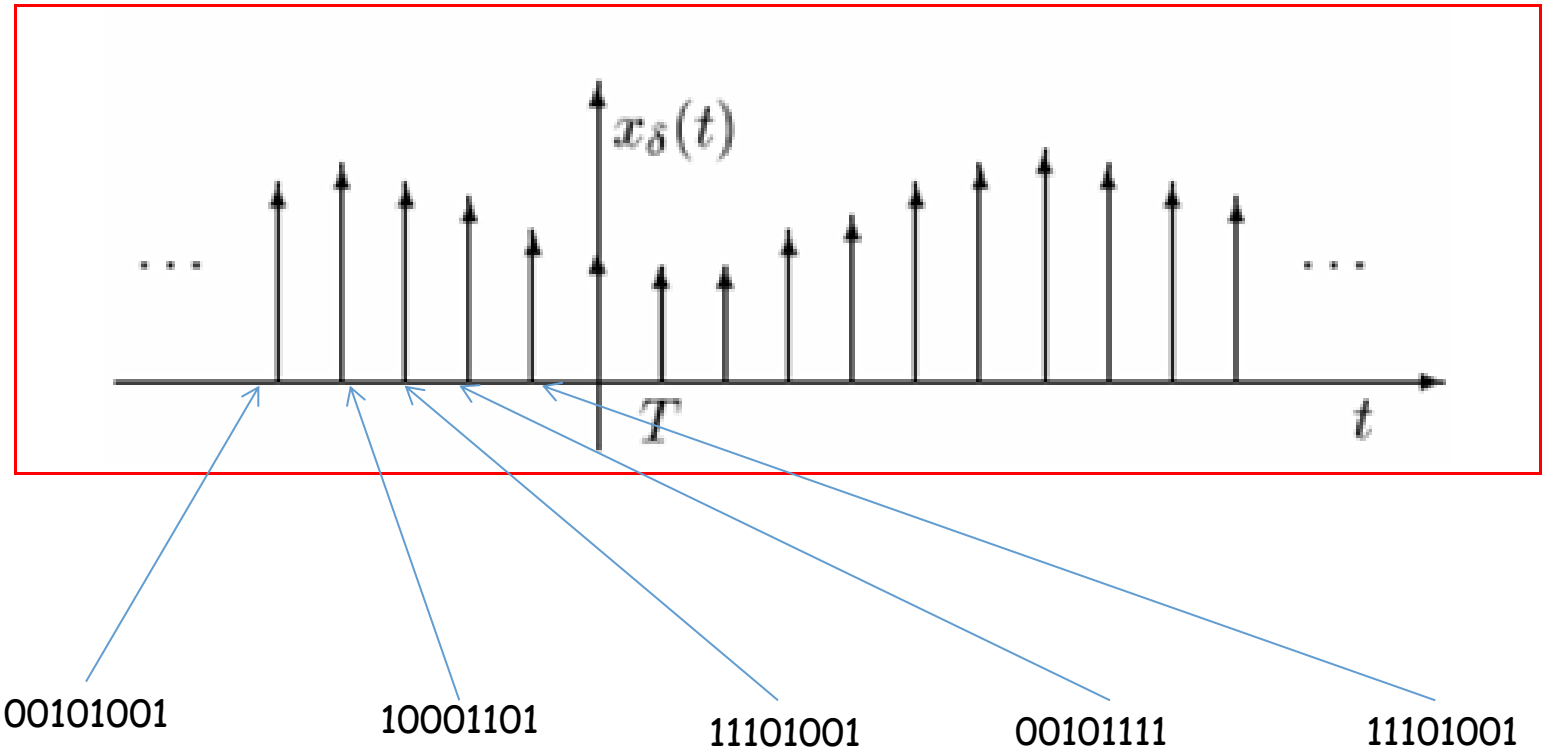
Quantizzazione

Quantizzazione

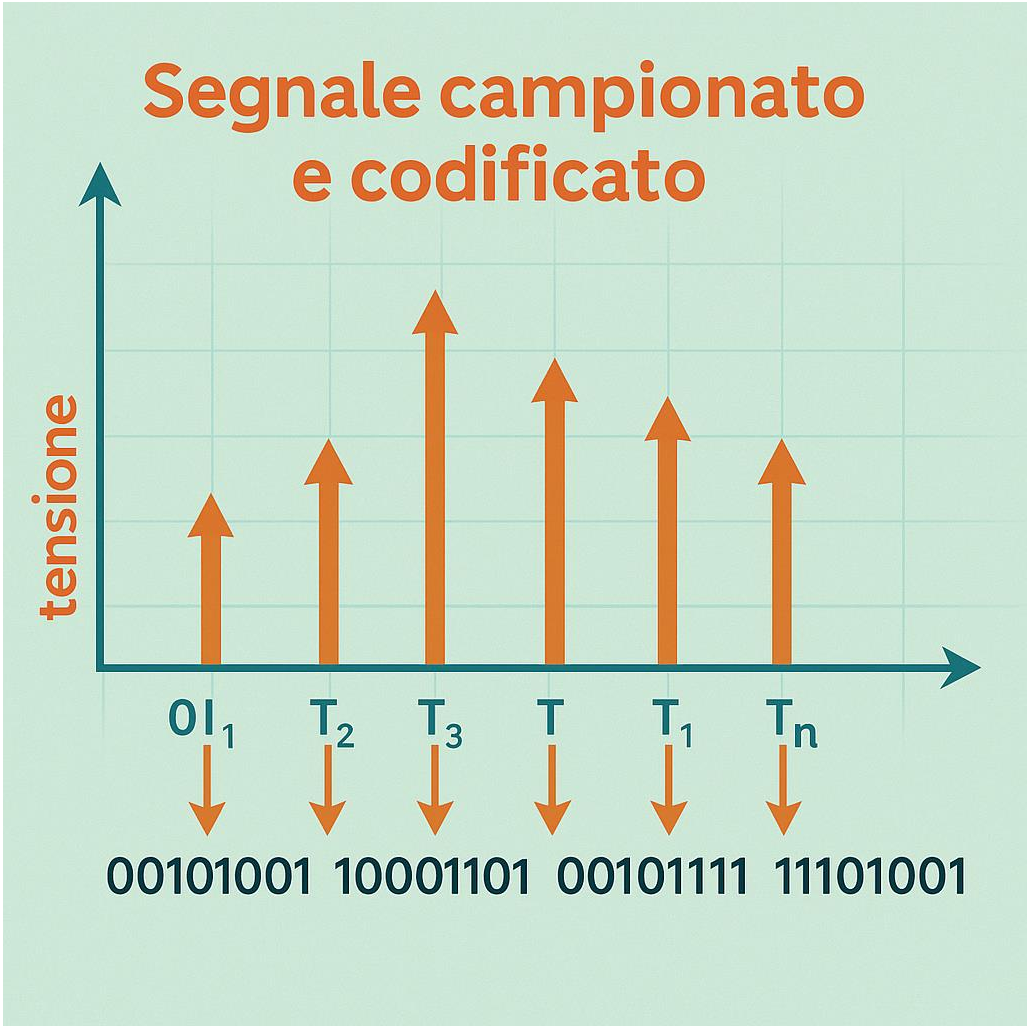


- La quantizzazione suddivide l'ampiezza in **n** intervalli uguali che vengono poi codificati in binario. Ogni valore di ampiezza del segnale campionato viene approssimato al più vicino valore discreto di ampiezza.
- Più valori (e quindi più bit) si utilizzano per suddividere le ampiezze, più il segnale risultante sarà preciso.

- Quindi, il nostro segnale analogico originario è ora rappresentato da una serie di WORD (esempio 8 bit)



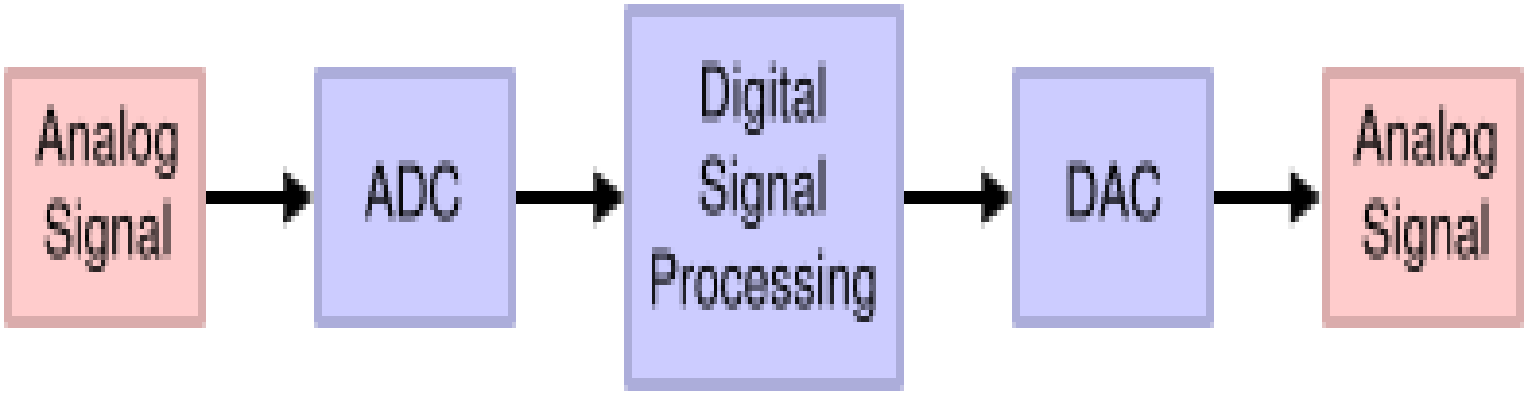
Quantizzazione



Ricapitoliamo

- Il teorema del campionamento ci consente di «trasformare» in sequenze di BIT (numeriche) i segnali analogici del mondo *esterno* al computer
- Il segnale è rappresentato in maniera esaustiva dalle sequenze numeriche a meno di un *errore* dovuto al processo di *campionamento* e poi *quantizzazione*
- I circuiti che usiamo per effettuare fisicamente le trasformazioni si chiamano CONVERTITORI A/D (Analogico – Digitale) indicati spesso con la sigla ADC.
- I circuiti che fanno l'operazione inversa, si chiamano CONVERTITORI D/A (Digitale – Analogico = DAC). In questo modo possiamo anche creare digitalmente un segnale (sintesi) e trasformarlo in segnale analogico. Esempio: MUSICA ELETTRONICA

Circuito universale

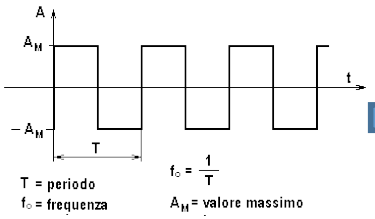


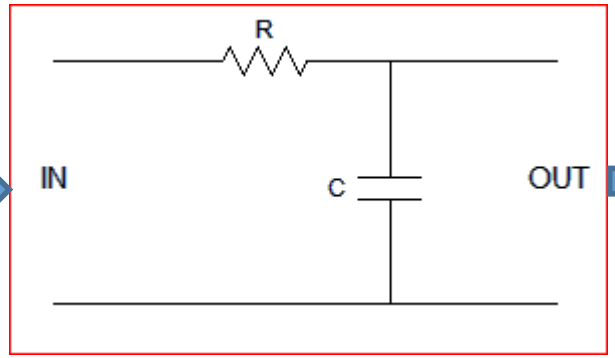
La trasformazione del segnale (funzione di trasferimento)

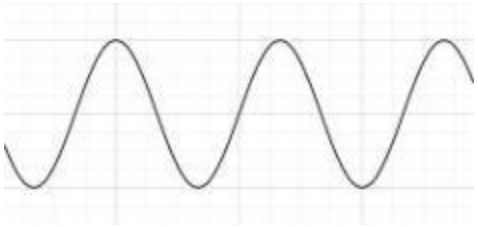
DIPENDE DA UN ALGORITMO

Filtro passa basso

- Elimina tutte le componenti di un segnale maggiori di una certa frequenza, detta frequenza di taglio
- Esempio di circuito analogico



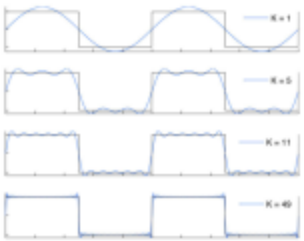


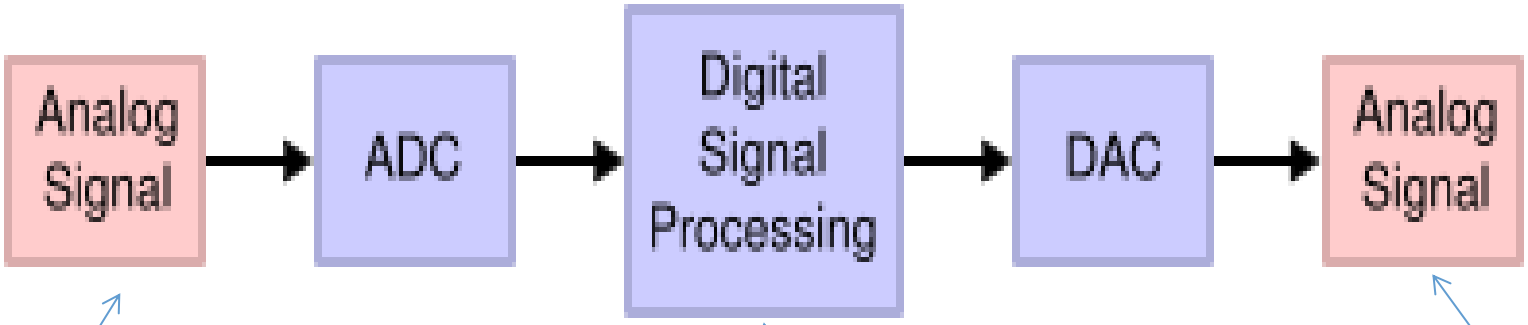


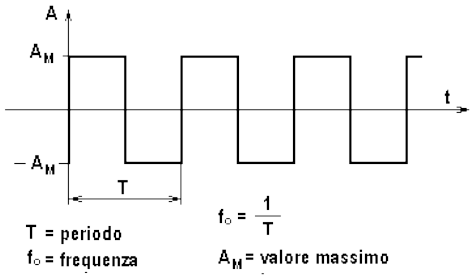
Ingresso : onda quadra, composta da tante sinusoidi

Filtro PB: ha la proprietà fisica Di far passare solo determinate frequenze (Quelle inferiori alla frequenza di taglio)

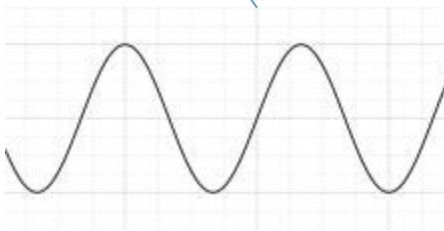
Uscita: una sola armonica (una sola sinusoide)



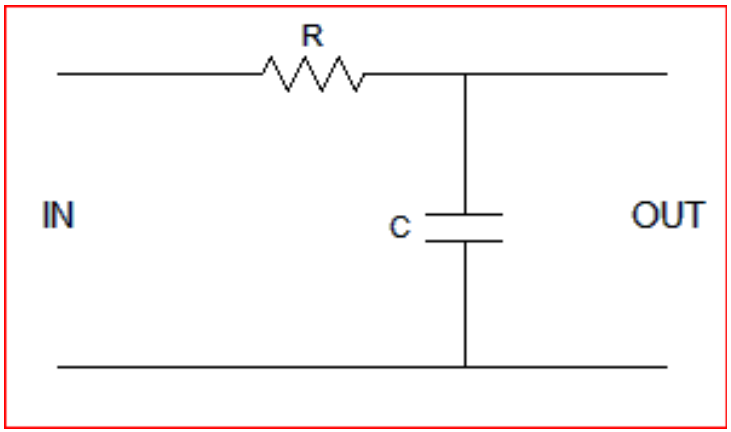





```
void transform(complex<double>* f, int N)     //calcola il vettore trasformato
{
  ordina(f, N);    //dapprima lo ordina col reverse order
  complex<double> W[N / 2]; //vettore degli zeri dell'unità.
                            //Prima N/2-1 ma genera errore con ciclo for successivo
                           //in quanto prova a copiare in una zona non allocata "W[N/2-1]"
  W[1] = polar(1., -2. * M_PI / N);
  W[0] = 1;
  for(int i = 2; i < N / 2; i++)
    W[i] = pow(W[1], i);
  int n = 1;
  int a = N / 2;
  for(int j = 0; j < log2(N); j++) {
    for(int i = 0; i < N; i++) {
      if(!(i & n)) {
        /*ad ogni step di raddoppiamento di n, vengono utilizzati gli indici */
        /*'i' presi alternativamente a gruppetti di n, una volta sì e una no.*/
        complex<double> temp = f[i];
        complex<double> Temp = W[(i * a) % (n * a)] * f[i + n];
        f[i] = temp + Temp;
        f[i + n] = temp - Temp;
      }
    }
    n *= 2;
    a = a / 2;
  }
}
```



- Dunque: abbiamo matematicamente ottenuto l'equivalente del comportamento fisico di:



- Con un ALGORITMO implementato in SOFTWARE!

Cambiare l'algoritmo equivale a cambiare il circuito…