

# Efficienza Sperimentale

Informatica@SEFA 2018/2019 - Laboratorio 5

Massimo Lauria <massimo.lauria@uniroma1.it>  
<http://massimolauria.net/courses/infosefa2018/>

Lunedì, 19 Novembre 2018

# Efficienza teorica vs sperimentale

Operazioni elementari su una macchina astratta

VS

Tempo di esecuzione in un ambiente operativo reale

# Ambiente operativo

Molte cose possono influire sul tempo di esecuzione di un programma.

- algoritmo
- implementazione dell'algoritmo
- linguaggio compilato/interpretato
- compilatore/interprete
- velocità della CPU/RAM/Disco/...
- memoria libera

# Eseguite i vostri programmi

Non limitatevi a scrivere i programmi e sperare che siano corretti come se steste facendo un compito in classe.

- la programmazione è la creazione di un artefatto
- eseguite i vostri programmi
- eseguiteli anche quando siete convinti che siano corretti
- testali su input sempre più grandi

# Esercizio

Provate ad implementare

- ricerca lineare
- ricerca binaria
- i vari algoritmi di ordinamento

e provatene l'efficienza sottoponendo input sempre più grandi.

# Grafici dei tempi di esecuzione

Vogliamo fare un plot comparativo del tempo di esecuzione di vari algoritmi.

- ricerca sequenziale vs ricerca binaria
- insertion sort vs bubble sort

# Ingredienti necessari

1. implementazioni degli algoritmi
2. dati di test
3. misurare i tempi
4. fare il plot

# 1. Implementazione degli algoritmi

**Datevi da fare!**

## 2. Dati di test

Nel modulo `infosefa.py` troverete una funzione `numeriacaso` per produrre una lista di numeri a caso, così da poter testare le vostre funzioni.

```
from infosefa import numeriacaso      1
                                       2
lista = numeriacaso(10,-5,5)          3
                                       4
lista_ordinata = numeriacaso(10,0,15,ordinati=True) 5
                                       6
print(lista)                          7
print(lista_ordinata)                 8
```

```
[-1, 2, -4, 2, 0, -4, 5, 0, -4, 4]
[0, 3, 3, 5, 5, 7, 7, 10, 10, 14]
```

# Come usare numeriacaso

## Importando il modulo

```
import infosefa
infosefa.numeriacaso(100,0,1)
```

1  
2

## oppure importando solo la funzione

```
from infosefa import numeriacaso
numeriacaso(100,0,1)
```

1  
2

# Documentazione di numeriacaso

```
from infosefa import numeriacaso  
help(numeriacaso)
```

1

2

Help on function numeriacaso in module infosefa:

```
numeriacaso(N, minimo, massimo, ordinati=False)
```

Produce una lista di numeri generati a caso.

Produce una lista di N elementi, ognuno dei quali preso a caso (con uguale probabilità) tra tutti i numeri interi compresi tra 'minimo' e 'massimo', estremi inclusi.

Se  $N < 0$  o  $\text{minimo} > \text{massimo}$  la funzione solleva un `ValueError`.

Se 'ordinati' è vero la lista restituita è ordinata.

### 3. Misurare i tempi di esecuzione

Se volete misurare tempi di esecuzione:

- dati omogenei
- dovete ripetere diverse volte e fare una media
- se i tempi sono piccoli ripetete **molte** volte
- non contate il tempo necessario a generare i dati

### 3. Esempio - Fibonacci

```
def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n-1)+fib(n-2)

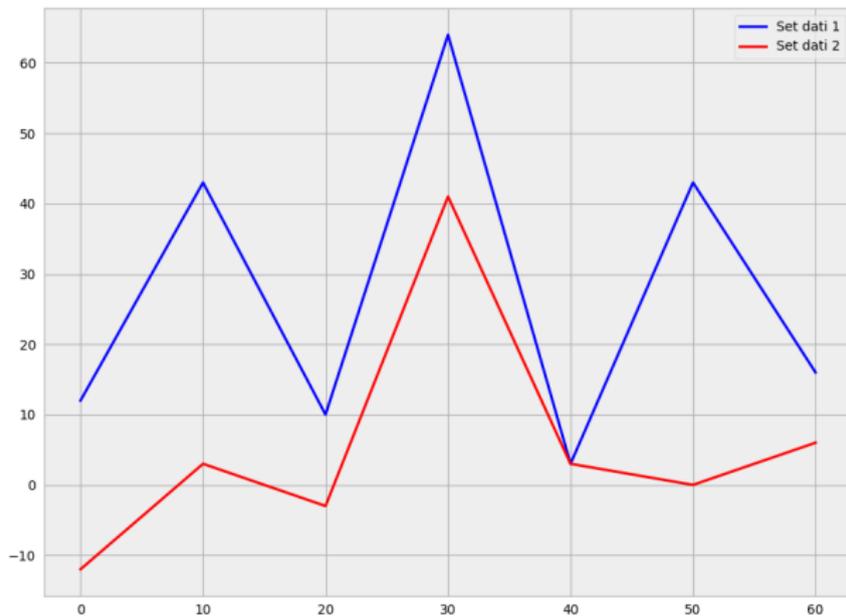
def ifib(n):
    cur,prev=1,1
    if n <= 2:
        return 1
    for i in range(3,n+1):
        cur,prev = cur+prev,cur
    return cur
```

### 3. Esempio (II) - Fibonacci

```
from time import process_time      1
                                    2
start = process_time()             3
for i in range(100):               4
    fib(25)                         5
end = process_time()               6
print("Induttivo: {}".format((end-start)/100)) 7
                                    8
start = process_time()             9
for i in range(1000):              10
    ifib(25)                       11
end = process_time()              12
print("Iterativo: {}".format((end-start)/1000)) 13
```

```
Induttivo: 0.02093623
Iterativo: 1.65600000000001018e-06
```

## 4. Grafici



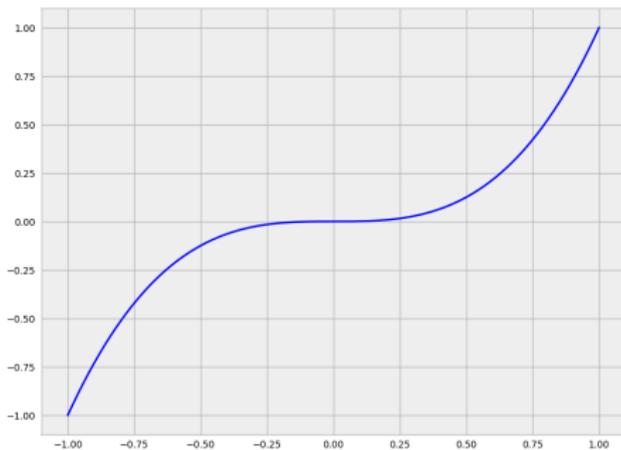
## 4. Grafici (codice sorgente)

```
from matplotlib.pyplot import plot,savefig,legend      1
                                                        2
x = [0,10,20,30,40,50,60]                             3
dati1 = [12,43,10,64,3,43,16]                         4
dati2 = [-12,3,-3,41,3,0,6]                          5
                                                        6
plot(x,dati1,label='Set dati 1')                      7
plot(x,dati2,label='Set dati 2')                     8
legend(loc='best')                                    9
savefig("assets/lab05_esempio1.png")                 10
```

- ▶ plot inserisce un grafico nella figura
- ▶ potete mettere più grafici nella stessa figura
- ▶ savefig salva la figura in un file a vostra scelta

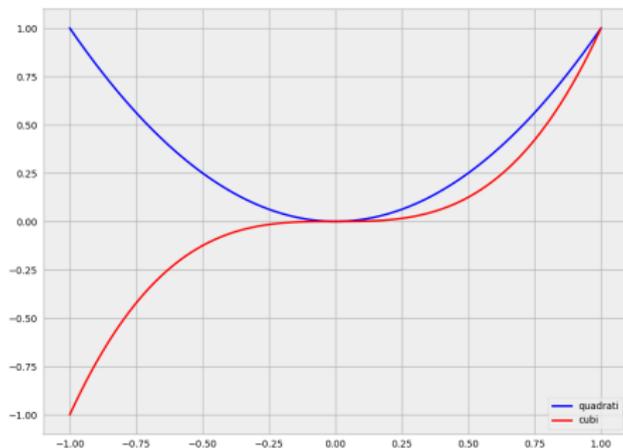
# plot(x,y)

```
from matplotlib.pyplot import plot,savefig      1
x = [ i/100 for i in range(-100,101)]         2
y = [val**3 for val in x]                       3
plot(x,y)                                       4
savefig("assets/lab05_esempio2.png")          5
```



# Plot di più funzioni

```
from matplotlib.pyplot import plot,savefig,legend 1
x = [ i/100 for i in range(-100,101)] 2
plot(x,[val**2 for val in x],label='quadrati') 3
plot(x,[val**3 for val in x],label='cubi') 4
legend(loc='best') 5
savefig("assets/lab05_esempio3.png") 6
```



# Un esempio di plot di runtime

