Università degli Studi di Cagliari
Corsi di Laurea in Ingegneria Chimica e Ingegneria Meccanica

FONDAMENTI DI INFORMATICA

http://people.unica.it/gianlucamarcialis

A.A. 2018/2019

Docente: **Gian Luca Marcialis**

PYTHON: ALGORITMI DI BASE
Ricerca e Ordinamento

Definizione

- Un algoritmo è un insieme **ordinato** di operazioni **non ambigue** ed **effettivamente computabili** che, quando eseguito, **produce un risultato** osservabile e si arresta **in un tempo finito**.
- Le proprietà di un algoritmo sono richiamate dai seguenti termini:
  - "Ordinato", "non ambigue": precisione
  - "Effettivamente computabili", "produce un risultato": correttezza
  - "In un tempo finito": efficienza

Proprietà degli algoritmi

- Precisione (comprensibilità o non ambiguità)
  - La descrizione dei passi deve essere non ambigua
    - Es. “per fare una torta di mele occorrono tre kg di mele, tre uova e mezzo kg di farina” vs.”legate l’arrosto con dello spago e salatelo”
  - I passi dell’algoritmo devono essere descritti in modo dettagliato *senza salti concettuali*
    - Omissione di passi: es. verifica della tensione per l’installazione di un elettrodomestico

- Correttezza
  - Per qualsiasi configurazione dei dati di ingresso, l’algoritmo deve produrre un risultato osservabile corretto

- Efficienza
  - Deve completare l’esecuzione del compito usando la quantità minima di **risorse** disponibili, ed in tempo finito

Esempio di algoritmo: determinare il maggiore di due numeri interi *x*, *y*

- 1. Calcola la differenza fra *x* e *y*
- 2. Valuta se la differenza è maggiore di 0
  - Se sì, il maggiore è *x*
  - Altrimenti, il maggiore è *y*

- Un *algoritmo* va visto come **il procedimento risolutivo di un problema**, ovvero:
  - Date le informazioni disponibili sotto forma di **dati in ingresso**
    - Es. i valori di x, y
  - È un insieme di regole che, eseguite **ordinatamente**, elaborano quei dati
  - permettendo di calcolare i risultati del problema sotto forma di **dati in uscita**
    - Es. il maggiore tra x e y

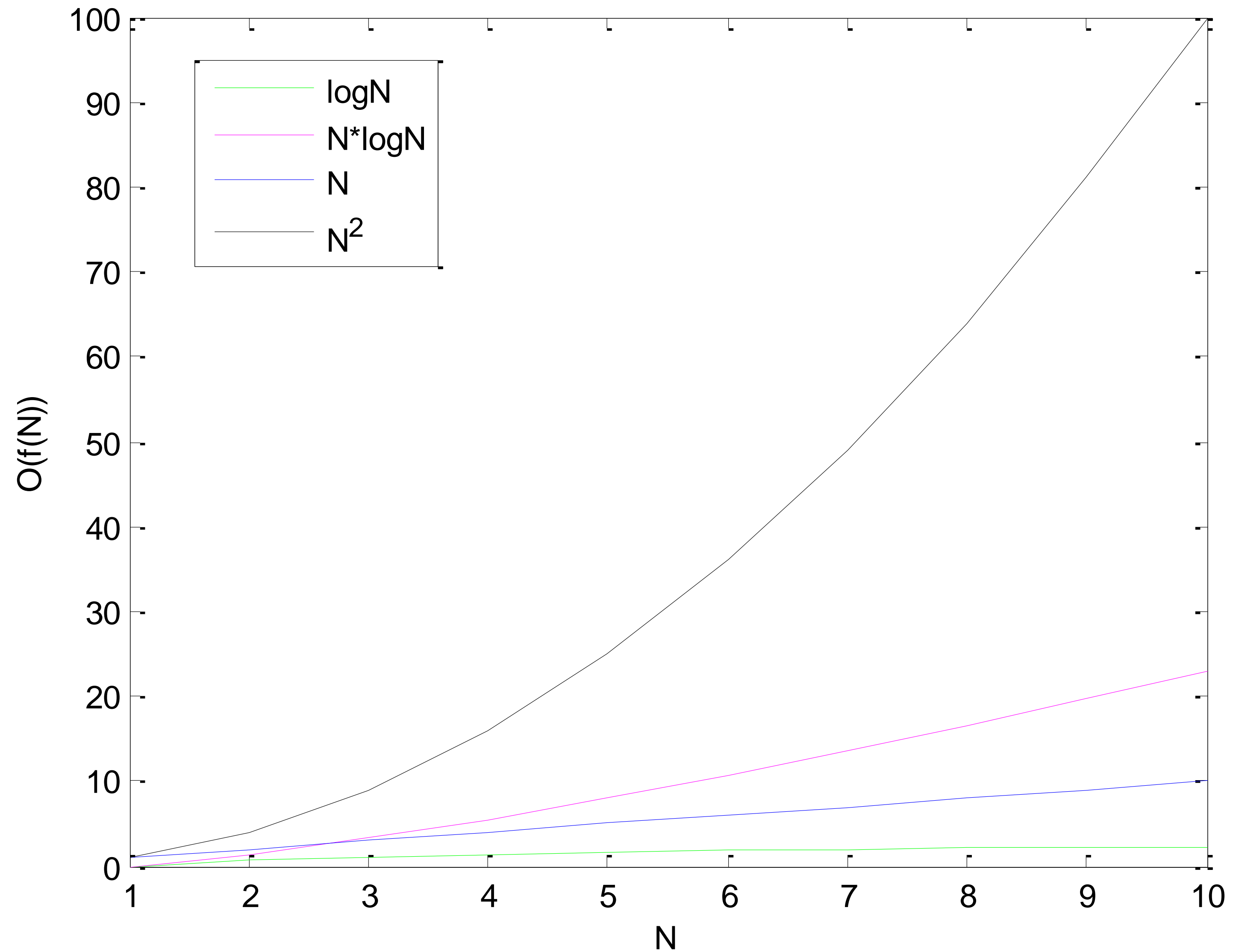
La complessità computazionale

- Efficienza: ottimizzazione delle “risorse”
- La risorsa “tempo” (efficienza temporale)
  - Un algoritmo è tanto più efficiente tanto meno tempo necessita per calcolare alla soluzione
  - Es. un algoritmo risolve un problema in 10 sec anziché 1 ora
- La risorsa “memoria” (efficienza spaziale)
  - Un algoritmo è tanto più efficiente tanto meno memoria necessita per calcolare alla soluzione
  - Es. un algoritmo risolve un problema impiegando 10 MB anziché 1 GB
- L’efficienza di un algoritmo può essere espressa attraverso una misura della sua “complessità di calcolo”, ovvero della frequenza di esecuzione delle istruzioni che lo compongono
  - Tale frequenza dipende di solito dalla “dimensione” dell’input, ovvero dal numero di informazioni che l’algoritmo necessita per **calcolare** la soluzione ➔ si parla di **complessità computazione** dell’algoritmo
  - Indicata con N la dimensione dell’input, per N «molto grande» la complessità computazionale asintotica si indica con **O(f(N))**

Problemi trattabili e intrattabili: qualche esempio

- Problemi trattabili
  - Trovare un elemento in un archivio (ordinato o no)
    - Complessità degli algoritmi: $O(\log N)$, $O(N)$
  - Ordinare un archivio
    - Complessità degli algoritmi: $O(N\log N)$, $O(N^2)$
- Problemi intrattabili, classe $O(e^N)$
  - Scomposizione in fattori primi di un numero
  - Il commesso viaggiatore
    - Un viaggiatore deve percorrere un'insieme di città fra loro collegate, passandovi una sola volta, e minimizzando la distanza complessiva percorsa
    - Soluzione algoritmica di complessità esponenziale
      - per chi fosse interessato ad approfondire v. Intelligenza Artificiale

La complessità asintotica: qualche esempio



Sommario

- Ricerca sequenziale O(N)
- Ricerca binaria O(logN)
- Ordinamento per selezione $O(N^2)$

Ricerca sequenziale

- Scopo: trovare un elemento in un insieme o lista
- Come sappiamo, data una lista o un dizionario `L`, o qualsiasi dato iterabile, possiamo sapere se è il dato `x` sia in L con l'istruzione `x in L`
- Tuttavia come è concepito l'algoritmo?
  - Estraggo un elemento da L,
    - Valuto se l'elemento corrisponde a quello da trovare
      - Se sì, restituisco «trovato»
      - Altrimenti, ripeto l'operazione fino a che non ho estratto tutti gli elementi
  - Restituisci «non trovato»

Ricerca sequenziale

- Sulla base di quanto scritto, una possibile implementazione è:

```
def ricerca_sequenziale(x,L):
    for elemento in L:
        if x==elemento:
            return True
        else
            return False   -33 punti
    return False
```

Ricerca binaria o informata

- In questo caso, la lista è ordinata, ad esempio, si ha $L[i] \leq L[i+1]$, qualunque $0 \leq i < N-1$ (ordine crescente, con N lunghezza della lista).
- Possiamo sfruttare tale informazione (da cui il nome dell'algoritmo), per ridurre considerevolmente i tempi della ricerca:
  - 0. Prendi l'elemento centrale della lista L[N/2]
  - 1. Se x==L[N/2], restituisci «trovato»
  - 2. Se x<L[N/2], riprendi da 0 sulla lista L[0:N/2]
  - 3. Se x>L[N/2], riprendi da 0 sulla lista L[N/2+1:N]
  - 4. Restituisci «non trovato»

Implementazione in Python

```
def ricerca_binaria(L, x):

  while L!=[]:              #finché la lista è non vuota
      m=len(L)/2
      if x==L[m]:
            return True     #elemento trovato
      if x<L[m]:            #divide in due la lista
            L=L[0:m]
      else:
            L=L[m+1:]

  return False #elemento non trovato
```

Implementazione alternativa in Python

```
def ricerca_binaria(L, x):

   t=0                          #indice della testa della lista
   c=len(L)                     #indice della coda della lista

   while t<c:                   #finché la lista è non vuota
       m=(t+c)/2
       if x==L[m]:
              return True       #elemento trovato
       if x<L[m]:               #divide in due la lista
              c=m
       else:
              t=m+1

   return False #elemento non trovato
```

Ordinamento di una lista

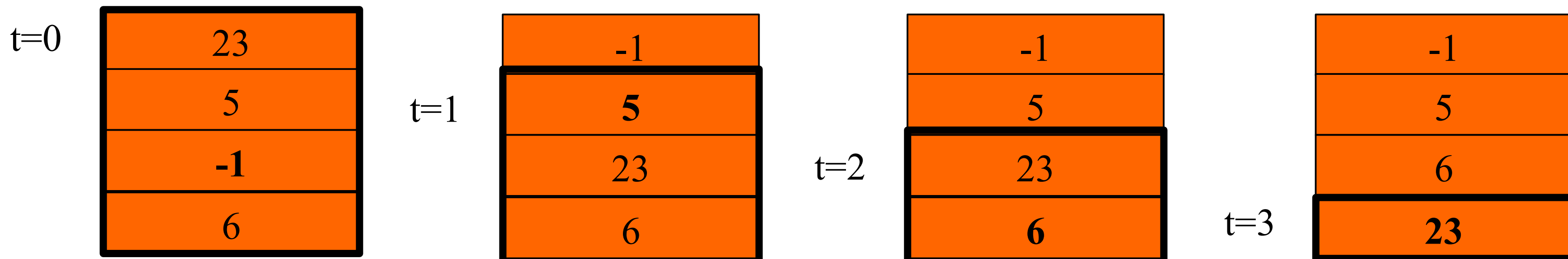
- La lista dispone di proprie funzioni di ordinamento come `sort()`.
- Esempio:

```
>>>L=[23, 5, -1, 6]
>>>L.sort()
>>>L
[-1, 5, 6, 23]
```

- Ma come funzionano? Che algoritmo seguono?

Un possibile algoritmo di ordinamento (crescente) – ordinamento per selezione

- Data una lista L di elementi non ordinati, sia `t=0` l'indice della sua **t**esta
  - 0. Cerco il minimo nella lista da `t` alla fine
  - 1. Permuto il minimo trovato con l'elemento in testa alla lista
  - 2. `t=t+1`
  - 3. Ripeto da 0 finché t<lunghezza della lista
- Per esempio:



Implementazione in Python

```
def ordina(L):
    n=len(L)
    t=0
    while t<n-1:
        j=trova_minimo(L[t:n])
        permuta(L,t,t+j)
        t=t+1
#Ci occorrono le funzioni trova_minimo e
#permuta
```

Funzione trova_minimo

```
def trova_minimo(L):
    pos_min=0
    i=1
    while i<len(L):
        if L[i]<L[pos_min]:
            pos_min=i
        i=i+1
    return pos_min
```

Funzione permuta

```
def permuta(L,i,j):
    t=L[i]
    L[i]=L[j]
    L[j]=t
```

Listato complessivo

```
def ordina(L):
        n=len(L)
        i=0
        while i<n-1:
                j=trova_minimo(L[i:n])
                permuta(L,i,i+j)
                i=i+1
```

```
def trova_minimo(L):
        pos_min=0
        i=1
        while i<len(L):
                if L[i]<L[pos_min]:
                        pos_min=i
                i=i+1
        return pos_min

def permuta(L,i,j):
        t=L[i]
        L[i]=L[j]
        L[j]=t
```

Esercizi

- Verificare se gli algoritmi sequenziale, binaria e di ordinamento funzionano anche con liste di stringhe

- Sia data una lista di tuple, ciascuna delle quali formata da tre interi indicanti giorno, mese ed anno.
  - Scrivere una funzione `precede(data1,data2)`, dove `data1` e `data2` sono due tuple come sopra. Fare in modo che la funzione restituisca *`True`* se `data1` precede `data2`, *`False`* altrimenti
  - Riscrivere l'algoritmo di ordinamento per selezione per ordinare una lista di tuple siffatte, dalla più recente alla meno recente.

Per saperne di più...

- K.A. Lambert, *Programmazione in Python*, Apogeo (Maggioli), 2012.
- C. Horstmann, R.D. Necaise, *Concetti di informatica e fondamenti di Python*, Apogeo (Maggioli), 2014.