Table of Contents

Fondamenti di Programmazione

Andrea Sterbini

lezione 7 - 24 ottobre 2022

RECAP: COME analizzare un problema?

- cerchiamo di capire **come lo faremmo su carta**
  - descrivendo gli **input ed output**
  - eventuali **controlli da fare sui dati**
  - possibili **errori da generare**
  - possibili **effetti collaterali**
  - possibili **scelte non indicate**

- lo dividiamo in **problemi più piccoli**
  - ripetendo il metodo di analisi
- **continuando a frammentarlo** finchè

    - la sua **descrizione è così semplice**
    - da poter essere **implementato**
- mano a mano **testiamo le sottofunzioni** realizzate
    - **semplificando enormemente il debug**

RECAP: k massimi di N numeri

Se **tengo tutti i dati in memoria**

- versione modifica distruttiva (tempo $O(N * k)$)
- versione copia e modifica (tempo $O(N * k)$)
- versione ordinamento e estrazione (tempo $O(N*log(N))$

Qualche micro nozione di complessità temporale dei programmi

$O(f(n))$: nel caso peggiore il tempo impiegato è ***"simile"*** alla funzione $f(n)$ (con **n** che è la dimensione dei dati in input)

ovvero "si comporta" o "cresce" come la funzione $f(n)$

Dal più veloce al più lento:

- $O(1)$: tempo costante (non dipende dall'input)
- $O(log(N))$: tempo logaritmico rispetto all'input (per ogni raddoppio di N il tempo aumenta di 1)
- $O(N)$: tempo proporzionale all'input (se N raddoppia il tempo raddoppia)
- $O(N * log(N))$: tempo proporzionale all'input * il suo logaritmo (p.es. sort)
- $O(N^2)$: tempo quadratico (se N raddoppia il tempo*4)
- $O(2^N)$: tempo esponenziale (se N aumenta di 1, il tempo raddoppia)

logaritmo = numero di cifre binarie

Come si comportano gli ordini di grandezza

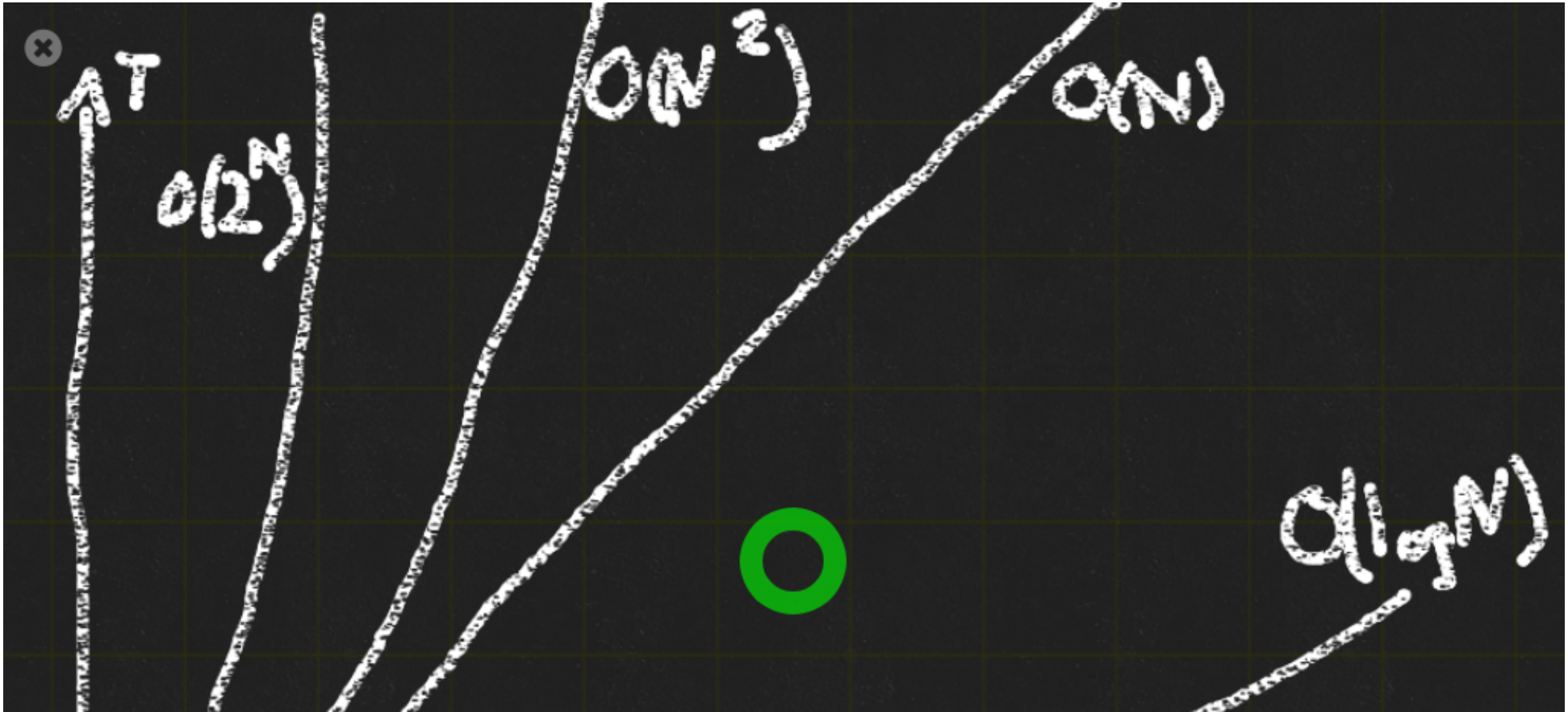
Visto che stiamo cercando **il caso peggiore**:

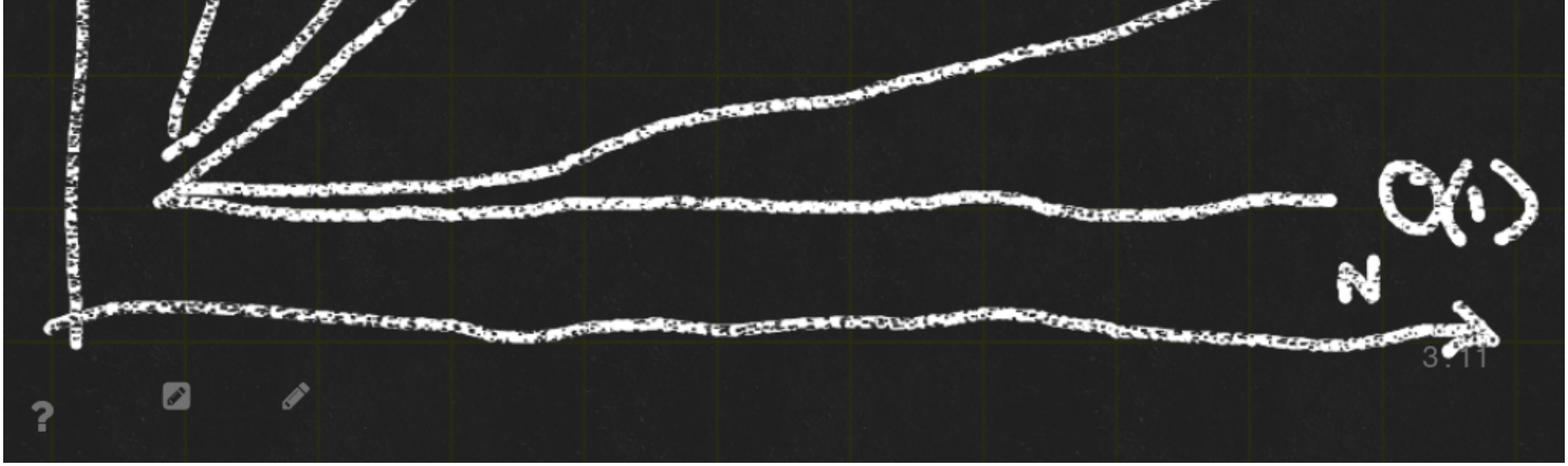
Se N è la dimensione dell'input da elaborare:

la formula	diventa	perchè
costante * O(F(N))	O(F(N))	ignoriamo i fattori costanti
O(F(N)) + O(G(N))	O(max(F(N),G(N)))	vogliamo il caso peggiore
O(F(N)) * O(G(N))	O(F(N)*G(N))	si moltiplicano

Esempi:

- $O(5 * n) ==> O(n)$ (si ignorano i fattori costanti)
- $O(log(n)) + O(1) ==> O(log(n))$ (il logaritmo cresce mentre una costante no)
- $O(log(n)) + O(n) ==> O(n)$ (il logaritmo cresce più lentamente di n)
- $O(n) * O(n) ==> O(n^2)$
- $O(n) * O(log(n)) + O(n) ==> O(n * log(n))$ (n*log cresce più rapidamente di n)





Nuovo vincolo: vogliamo usare poca memoria

Non ci serve ricordare o ordinare **tutti i valori** di L!!!!

Basta ricordare **SOLO k valori** (i migliori finora)

k massimi N numeri estratti a caso

- definisco ed inizializzo una lista vuota per i k valori (tempo costante)
- estraggo/leggo i valori e per ciascuno (N volte)
  - **aggiorno la lista dei k migliori già visti** (tempo ???)
- ritorno la lista dei k valori finale (tempo costante)

Per aggiornare i k valori con un nuovo X

- se la lista di valori ha meno di k elementi
  - aggiungo il nuovo valore (tempo costante)

- se X è <= del minimo dei k valori (tempo $O(k)$)

- lo posso ignorare (sono tutti meglio)

- altrimenti è maggiore del minimo
  - tolgo il valore più piccolo (tempo $O(k)$)
  - aggiungo il nuovo valore (tempo costante)

In [11]:

```python
import random
# per ottenere i k massimi di tantissimi valori
# estratti a caso
def k_massimi_da_seq_casuale(k, N, seed):
    # settare il generatore al valore iniziale
    random.seed(seed)
    # definisco ed inizializzo una lista vuota
    # per i k valori
    massimi = []
    # genero i valori in input e per ciascuno
    for _ in range(N):
        X = random.randint(-1000000,+1000000)
        # aggiorno la lista dei k migliori già visti
        aggiorna_k_massimi(massimi, X, k)
    # ritorno la lista dei k valori finale
    return massimi

def aggiorna_k_massimi_dummy(massimi, X, k):
    massimi.append(X)
    if len(massimi) > k:
        massimi.pop(0)

#k_massimi_da_seq_casuale(10, 10000, 0)
```

In [16]:
```python
# per aggiornare la lista dei k valori
# con un nuovo valore X
def aggiorna_k_massimi(L, X, k):
    # se la lista di valori ha meno di k elementi
    if len(L) < k:
        # aggiungo il nuovo valore
        L.append(X)
        return
    # se X è minore o uguale del minimo dei k valori
    minimo = min(L)
    if X <= minimo:
        # lo posso ignorare
        return
    # altrimenti è maggiore del minimo
        # tolgo il valore più piccolo
    L.remove(minimo)
        # aggiungo il nuovo valore
    L.append(X)

valori = []
random.seed(0)
for _ in range(20):
    valori.append(random.randint(-1000000, +1000000))
valori.sort()
print(valori)

k_massimi_da_seq_casuale(4,20,0)
```

```
[-915099, -541893, -457013, -363908, -249117, -192083, -150792, -117998, -504, 19064, 722
20, 223440, 589545, 643744, 740327, 770880, 866976, 869947, 904450, 925676]
```

Out[16]:
```
[866976, 925676, 869947, 904450]
```

In [17]:
```python
%timeit k_massimi_da_seq_casuale(3,    1000000, 0)
%timeit k_massimi_da_seq_casuale(30,   1000000, 0)
%timeit k_massimi_da_seq_casuale(300,  1000000, 0)
%timeit k_massimi_da_seq_casuale(3000, 1000000, 0)
None
```

```
1.44 s ± 82.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.96 s ± 171 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
7.57 s ± 629 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
56 s ± 3.06 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Nuova idea: tenere la lista di k elementi ordinata

e se tenessi la lista di k valori ordinata???

- trovare il minimo è costante (ultimo elemento)
- eliminare il minimo è costante
- per mantenere la lista ordinata
  - aggiungere l'elemento (tempo costante)
  - riordinare la lista (tempo $O(k * log(k))$)

In [22]:

```python
# Versione che tiene ordinata la lista di K elementi
def aggiorna_lista_k_massimi_ordinati(Lordinata, X, K):
    # se len(Lordinata) < K
    if len(Lordinata) < K:
        # aggiungo il valore
        Lordinata.append(X)
        # mantengo ordinata la lista (tempo O(K*log(K)))
        Lordinata.sort(reverse=True)
        return
    # se X <= minimo
    if Lordinata[-1] >= X:
        # posso ignorarlo, esco
        return
    # altrimenti se X è maggiore del minimo
        # lo aggiungo e tolgo il minimo
    Lordinata[-1] = X
    # mantengo ordinata la lista (tempo O(K*log(K)))
    Lordinata.sort(reverse=True)


```

In [23]:
```
# per ottenere i k massimi di tantissimi valori
# estratti a caso
def k_massimi_da_seq_casuale_ordinata(k, N, seed):
        # settare il generatore al valore iniziale
    random.seed(seed)
    # definisco ed inizializzo una lista vuota
    # per i k valori
    massimi = []
    # genero i valori in input e per ciascuno
    for _ in range(N):
        X = random.randint(-1000000,+1000000)
        # aggiorno la lista dei k migliori già visti
        aggiorna_lista_k_massimi_ordinati(massimi, X, k)
    # ritorno la lista dei k valori finale
    return massimi

# Tempo totale: proporzionale a N*K*log(K)
valori = []
random.seed(0)
for _ in range(20):
    valori.append(random.randint(-1000000, +1000000))
valori.sort()
print(valori)

k_massimi_da_seq_casuale_ordinata(4,20,0)
```

```
[-915099, -541893, -457013, -363908, -249117, -192083, -150792, -117998, -504, 19064, 722
20, 223440, 589545, 643744, 740327, 770880, 866976, 869947, 904450, 925676]
```

Out[23]:
```
[925676, 904450, 869947, 866976]
```

In [24]:
```
%timeit k_massimi_da_seq_casuale_ordinata(3,    1000000, 0)
%timeit k_massimi_da_seq_casuale_ordinata(30,   1000000, 0)
%timeit k_massimi_da_seq_casuale_ordinata(300,  1000000, 0)
%timeit k_massimi_da_seq_casuale_ordinata(3000, 1000000, 0)
None
```

```
1.13 s ± 116 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.19 s ± 128 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.89 s ± 904 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.68 s ± 102 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

NOTA: L'aggiornamento della lista ordinata può essere fatto più rapidamente se

- troviamo la posizione dove inserire X
  con una ricerca binaria (tempo *O(log(k))*)
- e poi un insert (tempo $O(k)$)

Ovvero tempo $O(k + log(k)) ==> O(k)$

Quindi la soluzione potrebbe scendere a

- Tempo totale: proporzionale a $N * K$
  (se $K << N$)

Implementazione della ricerca binaria per chi vuole, a casa

Vediamo che vuol dire $O(k * log(k) * N)$

k	$log(k)$	N	**1**	**10**	**100**	**1000**	**10000**
10	3.32		3.3	3.3e+02	3.3e+03	3.3e+04	3.3e+05
100	6.64		6.6	6.6e+02	6.6e+04	6.6e+05	6.6e+06
1000	9.97		1e+01	1e+03	1e+05	1e+07	1e+08
10000	13.29		1.3e+01	1.3e+03	1.3e+05	1.3e+07	1.3e+09

E invece che vuol dire $O(k * N)$

k	N	**1**	**10**	**100**	**1000**	**10000**
10		1	10	1000	10_000	100_000
100		1	10	10000	100_000	1_000_000
1000		1	10	10000	100_000	10_000_000
10000		1	10	10000	100_000	100_000_000

Wooclap: quanto ho capito gli argomenti di oggi ?

1 Vai a www.wooclap.com

2 Immettere il codice dell'evento nel banner superiore

Codice evento
F22LEZ7