

# Complessita

- Complessita
- Algoritmi classici

# Contenuti

Notazione Big O - Quanto veloce è il nostro algoritmo?

Ricerca

Algoritmo: Ricerca binaria Algoritmo: Ricerca lineare

Ordinamento

Algoritmo: Bubble sort (ordinamento a bolle)

Struttura Dati: Linked list (lista concatenata)

Struttura Dati: Coda (queue)

# Motivi

Conoscere algoritmi e strutture dati aiuta nel problem solving e a programmare meglio

Sono argomenti chiesti come esercizi durante le interviste di lavoro tecniche

# Motivi

E' raro scrivere da zero algoritmi di ordinamento.

Esistono algoritmi conosciuti estremamente efficienti.

Applicare algoritmi conosciuti a problemi specifici permette di sviluppare soluzioni efficaci

# Argomenti principali

Notazione Big O (complessità di un algoritmo)

Ricerca lineare e binaria

Ordinamento a bolle e coda

Linked List

# Notazione Big O - Quanto veloce è il nostro algoritmo?

Velocità processamento dati

Risparmio di memoria

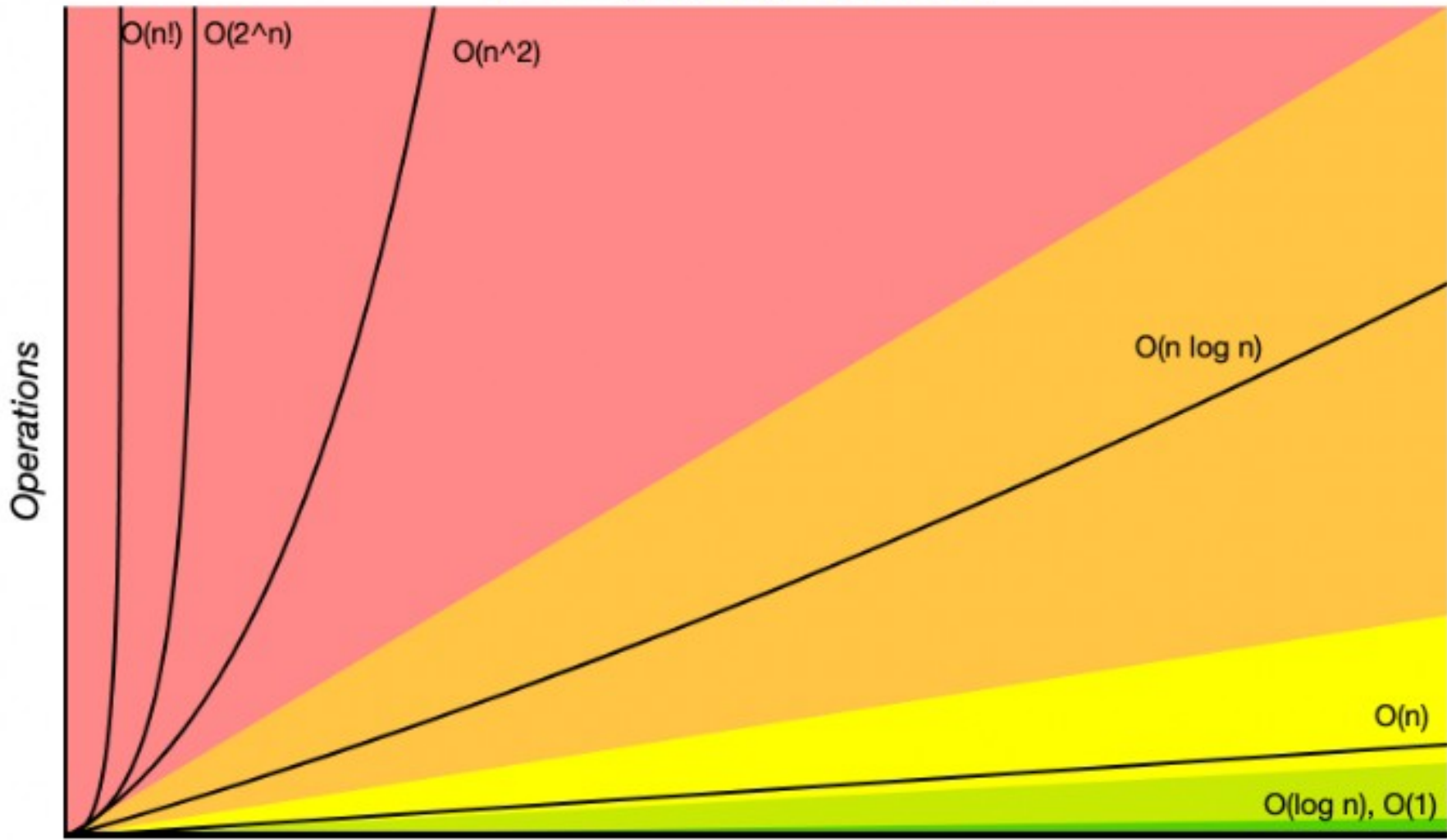
Big O ci aiuta a capire quanto veloce è un algoritmo ed eventualmente anche quante memoria occupa.

# Big O

Quanto **cresce** il **numero** di **operazioni**  
al crescere del nostro **input**

(Operazione: macro sequenza di istruzioni)

Horrible Bad Fair Good Excellent



Input

# Complessita lineare : $O(n)$

```
def sum_char_codes(string: str):  
    sum = 0  
    for char in string:  
        sum += ord(char) #Converte in Unicode  
    return sum  
  
result = sum_char_codes("Hello")  
print(result)  
  
>>> 532
```

# Complessita Lineare : $O(n)$

```
def sum_char_codes(string: str) -> int:  
    sum = 0  
    for char in string:  
        sum += ord(char)  
    for char in string:  
        sum += ord(char)  
    return sum
```

$\cancel{O(2n)} \rightarrow O(n)$

# Complessita : $O(n)$

```
def sum_char_codes(string: str) -> int:  
    sum = 0  
    for char in string:  
        symbol = ord(char)  
        if symbol == 69:  
            return sum  
        sum += symbol  
    return sum
```

**CONSIDERA SEMPRE CASO PEGGIORE !**

# Complessita : $O(n^2)$

```
def sum_char_codes(string: str) -> int:  
    sum = 0  
    for char in string:  
        for char in string:  
            symbol = ord(char)  
            sum += symbol  
    return sum
```

# Riassunto Complessita

Crescita **numero** operazioni rispetto **numero** di  
dati di **input**

Le costanti sono omesse

Si misura il caso **peggiore**

# Example: Bubble Sort

- Complessita:  $O(n^2)$
- Calcolo complessita:  $n + (n-1) + (n-2) + (n-3) + (n-4) \dots - (n - (n-1)) = \dots$  (Metodo di Gauss) ...  
si arriva a  $O(n(n-1)/2) \sim O(n^2 - n/2) \sim O(n^2)$