Complessità

La **complessità si misura in funzione di n** che sarebbe la dimensione dell'input.

La **complessità di un if else** è il "costo" della condizione da verificare + il massimo tra i blocchi contenuti nell'if o nell'else

La **complessità di un ciclo for** è la $\sum_{i=0}^{m-1} k$ dove i è il contatore del ciclo e k è il corpo, quindi sarebbe la somma m-1 volte della complessità del corpo del for (se il for cicla 10 volte sommo 10 volte la complessità del corpo)

La **complessità di un ciclo while** è la $\sum_{i=0}^{m}(h+k)$ dove i è il contatore del ciclo ,h è la condizione da verificare e k è il corpo

Il tempo impiegato per risolvere un problema dipende sia dall'algoritmo utilizzato sia dalla "dimensione" dei dati a cui si applica l'algoritmo

Dato che calcolare la complessità di un algoritmo in funzione dei dati in input è molto difficile usiamo la **Notazione Asintotica**:
Non calcoliamo esplicitamente il tempo (o lo spazio) impiegato da un algoritmo, ma piuttosto come questi parametri crescono al variare della dimensione dell'input

Esistono diversi tipo di complessità definiti con **O(n)**:
Si hanno così algoritmi (funzioni) di complessità asintotica di ordine:

- Costante: $1, \ldots$
- Sotto-lineare: $\log n,\ n^c$ con $c < 1$
- Lineare: n
- Polinomiale: $n \log n,\ n^2,\ n^3,\ \ldots,\ n^c$ con $c > 1$
- Esponenziale: $c^n,\ \ldots,\ n^n,\ \ldots$

Funzione Constante
f(n)=c c costante

Funzione Logaritmica
$f(n)=\log_b n \qquad (b>1)$

Funzione Lineare
$f(n) = c * n$

Funzione n log n
$f(n) = n * log * n$

Funzione Esponenziale

$f(n) = b^n$

Funzione Polinomiale

$f(n)=a_0 + a_1 n + a_2 n^2 + \ldots + a_d n^d$

Funzione Quadratica

$f(n) = c * n^2$

```
for (i=0;i<n;i++)
for(j=0;j<n;j++)
do something
```

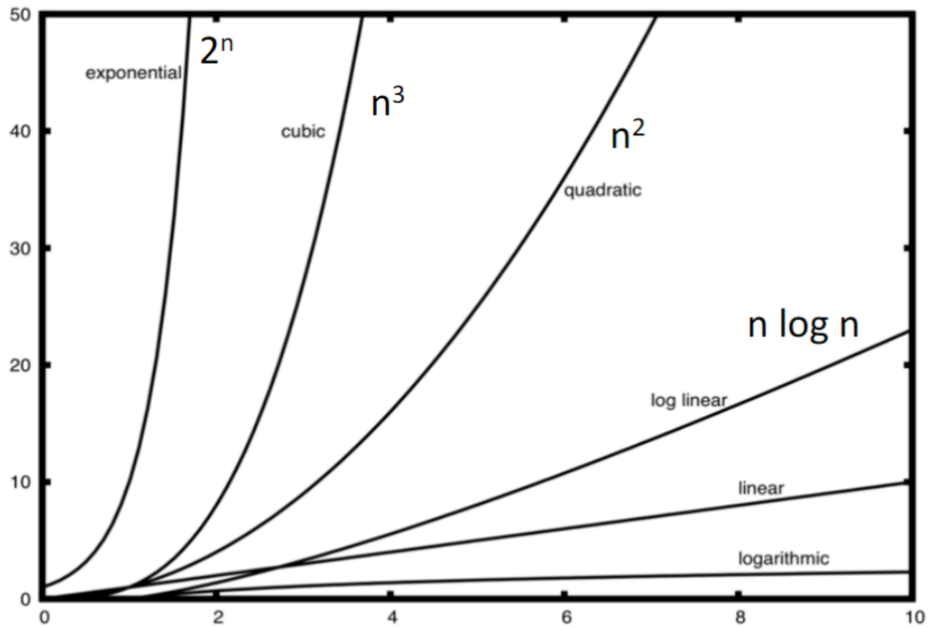
Funzione Cubica

$f(n) = c * n^3$

```
for (i=0;i<n;i++)
for(j=0;j<n;j++)
for(k=0;k<n;k++)
do something
```

Qui di seguito mostriamo alcuni ordini di grandezza tipici, elencati in maniera crescente.

f(n)	
1	
log n	
$\sqrt{n}$	
n	
n log n	
n^c	c>1
c^n	c>1



- O-grande dice solo **quanto può crescere al massimo** un algoritmo.
- Ma **potrebbero esistere limiti superiori migliori**, cioè più stretti.

Infatti, se $T(n) = O(n^4)$, è anche vero che: $T(n) = O(n^7)$

Ω-grande
l'algoritmo ci mette almeno così tanto tempo (nel caso peggiore, nel medio, o nel migliore a seconda del contesto).

Θ-grande
Descrive la complessità esatta di un algoritmo

O-grande →limite superiore
Ω-grande → limite inferiore
Θ-grande →complessità esatta

Un algoritmo A che risolve un problema P è ottimale se:

1. P ha complessità Ω(f(n))
2. A ha complessità O(f(n))

```
// c è una costante positiva
for (int i = 0; i <= n; i += c) {
//espressioni con costo O(1)
}
```

O(n)

```
// c è una costante positiva
for(int i = 1; i <=n; i += c) {
    for (int j = 1; j <=n; j += c) {
    //espressioni con costo O(1)
       }
  }
```

O(n²)

```
// c è una costante positiva
for (int i = 1; i <= n; i *= c) {
   //espressioni con costo O(1)
}
```

O(log n)

```
// c è una costante positiva > 1
for(int i = 2; i <=n; i = pow(i,c)) {
   //espressioni con costo O(1)
}
```

O(log log n)

Qual è la complessità del seguente algoritmo?

```
void func(int n) {
int count=0;
for(i=n/2; i<=n; i++) {
    for(j=1; j<=n; j=2*j) {
        for(k=1; k<=n; k=k*2){
            count++; }
} } }
```

O(n)

O(log n)

O(log n)

$\boxed{O(n \log^2 n)}$