

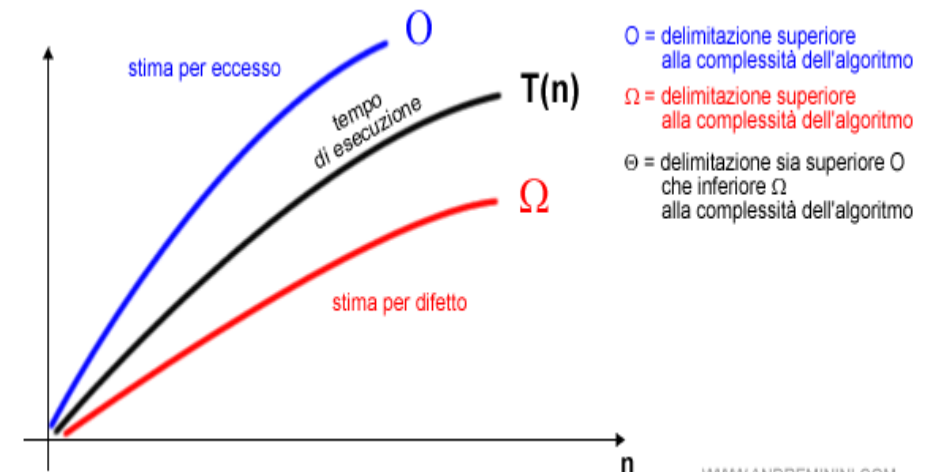
# Complessità di algoritmi e problemi

*Francesco Pugliese, PhD*

*neural1977@gmail.com*

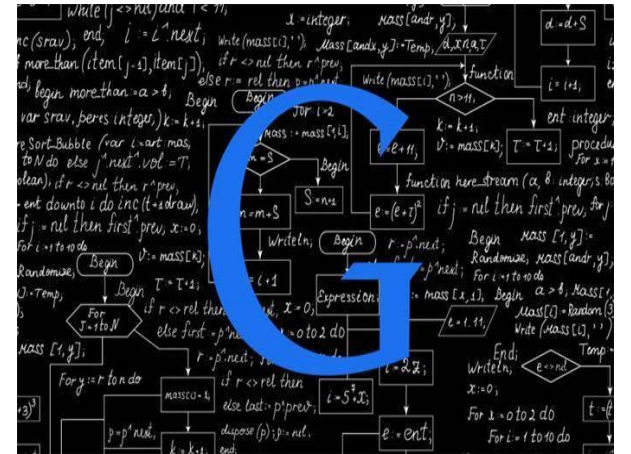
# Complessità di algoritmi e problemi

- ✓ Con **complessità di un algoritmo** o **efficienza di un algoritmo** ci si riferisce dunque alle risorse di calcolo richieste. I problemi sono classificati in **differenti classi di complessità**, in base all'efficienza del migliore algoritmo noto in grado di risolvere quello specifico problema.
- ✓ Una distinzione informale, ma di grande rilievo, è quella posta tra i cosiddetti problemi facili, di cui si conoscono **algoritmi di risoluzione efficienti**, e difficili, di cui gli unici algoritmi noti non sono efficienti.



# Complessità di algoritmi e problemi

- ✓ Ad esempio la maggior parte della **crittografia moderna** si fonda sull'esistenza di **problemi ritenuti difficili**.
- ✓ Ha enorme rilevanza lo studio di tali problemi, poiché, qualora si dimostrasse l'esistenza di un **algoritmo efficiente** per un problema ritenuto difficile, i sistemi crittografici basati su di esso non sarebbero più sicuri.
- ✓ L'esecuzione di un **programma** implica un **costo economico**, dovuto all'utilizzo delle **risorse** ( memoria, traffico sulla rete, spazio su disco, ecc. ) e di **tempo** di elaborazione.



# Complessità di algoritmi e problemi

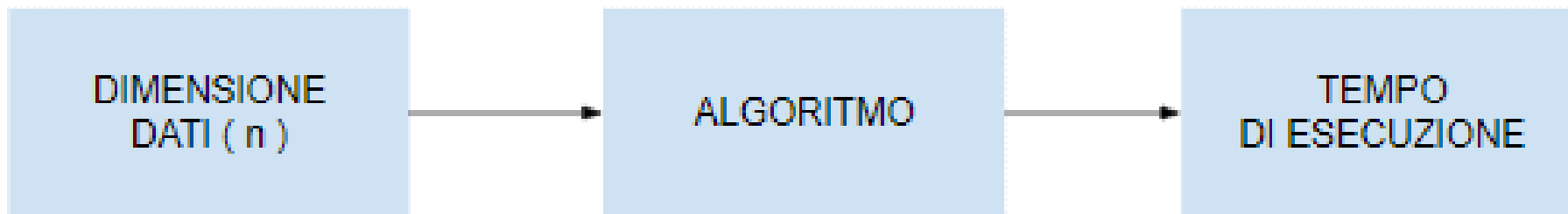
- ✓ **Complessità spaziale:** riguarda l'utilizzo delle risorse da parte di un programma.
- ✓ **Complessità temporale:** riguarda il tempo di esecuzione di un programma.
- ✓ Nello sviluppo di un algoritmo è particolarmente importante la **complessità temporale**. La **complessità spaziale** è meno importante, in quanto spesso compensata dai **progressi tecnologici** sui componenti hardware del computer ( es. hard disk e memorie ram più capienti ).



# Complessità Temporale

---

- ✓ La **complessità temporale** cresce al crescere della **dimensione  $n$  dei dati in input**. Per dimensione dell'input intendiamo la quantità dei dati in input di un algoritmo.
- ✓ Il **tempo di esecuzione** di un algoritmo è strettamente legato al funzionamento di un algoritmo. Per **raggiungere un obiettivo** esistono algoritmi più efficienti di altri.



# Complessità Temporale

---

- ✓ Esistono anche altri fattori che possono influenzare il tempo di esecuzione ma in un'**analisi dell'algoritmo** non vanno considerati.
- ✓ **Esempio.** Nel computo della **complessità** di un algoritmo non vanno considerati gli aspetti hardware, **la velocità del processore (cpu)**, né le tecniche di compilazione, il compilatore o il linguaggi di programmazione utilizzato.
- ✓ In informatica per calcolare la complessità computazionale di un algoritmo si utilizza **l'analisi asintotica**. Tuttavia, l'analisi asintotica è uno **strumento della matematica** che si applica alle funzioni, mentre il tempo di esecuzione non lo è.

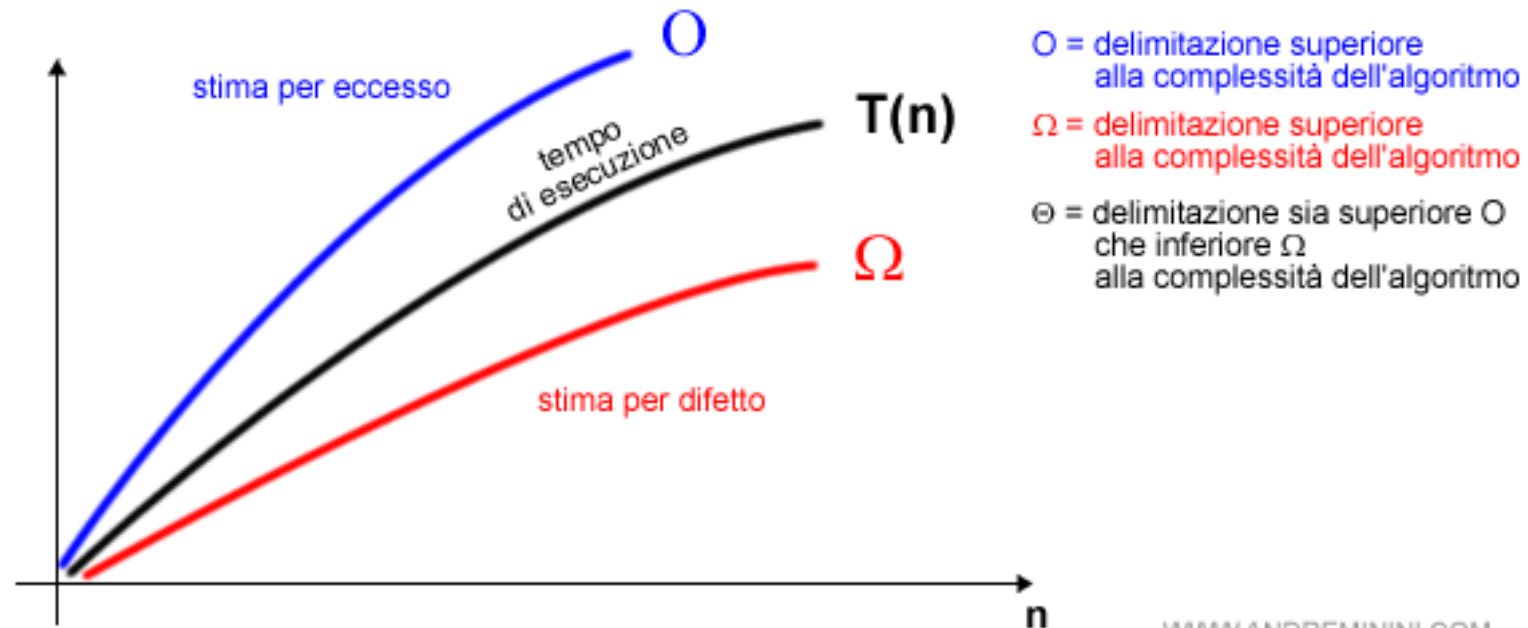
# Complessità Temporale

---

- ✓ Quindi, per usare l'analisi asintotica devo **trasformare il tempo di esecuzione dell'algoritmo in una funzione  $T(n)$**  in funzione della dimensione  $n$  dei dati input. In genere la funzione  $T(n)$  misura il numero di comandi eseguiti dall'algoritmo.
- ✓ Data **un'istanza di dimensione  $n$** , nel caso peggiore l'algoritmo ha una complessità temporale  **$O(f(n))$  se  $T(n)=O(f(n))$** .
- ✓ Dove  $n$  è il **numero delle righe eseguite** ( dimensione  $n$  dei dati ) mentre  **$f(n)$**  è un limite superiore del tempo di esecuzione dell'algoritmo nell'ipotesi peggiore.

# Complessità Temporale

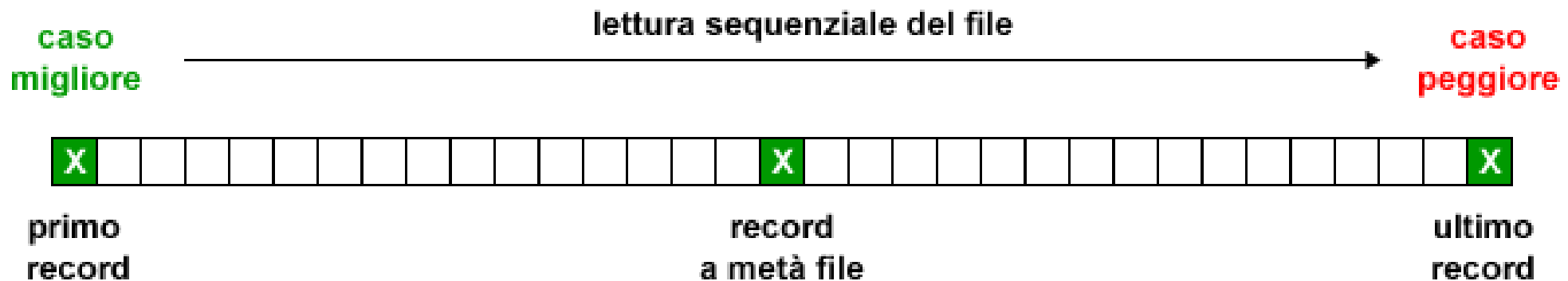
- ✓ Oltre a **O** grande si utilizzano anche le altre notazioni del **calcolo asintotico**, ossia omega ( **$\Omega$** ) e theta ( **$\Theta$** ). Qui ci limitiamo ad accennare la loro esistenza per non appesantire la spiegazione. La differenza tra O,  $\Omega$  e  $\Theta$  è la delimitazione superiore, inferiore o media della funzione  $T(n)$ . Per approfondire la differenza rimando alla lettura del **calcolo asintotico**.





# Complessità Temporale

- ✓ In ogni caso, per valutare la **complessità di un algoritmo** si utilizza sempre l'ipotesi del **caso peggiore**.
- ✓ **Perché si utilizza il caso peggiore?** Facciamo un esempio pratico, un algoritmo cerca sequenzialmente un **dato X** in un file composto da **1000 record**, partendo dal primo all'ultimo.



# Complessità Temporale

---

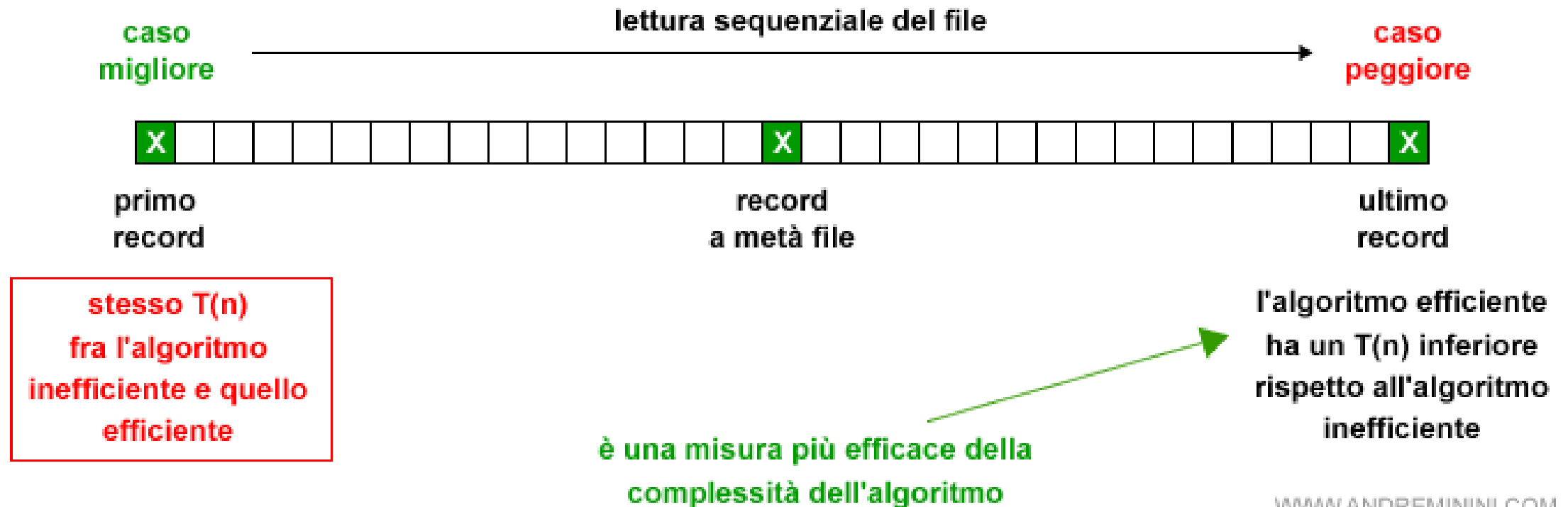
✓ Possono verificarsi tre casi:

- **Caso migliore.** Il **dato X** si trova al primo record (1 iterazione).
- **Caso intermedio.** Il **dato X** si trova alla metà del file (500 iterazioni).
- **Caso peggiore.** Il **dato X** si trova alla fine del file (1000 iterazioni). Per arrivarci l'algoritmo deve leggere tutto il file.

✓ Nell'**analisi di un algoritmo** si utilizza il caso peggiore, perché è quello che ne misura meglio l'efficienza dal punto di vista tecnico.

# Complessità Temporale

- ✓ Viceversa, il caso migliore non permette di distinguere un algoritmo inefficiente da uno efficiente.



WWW.ANDREMININI.COM

# Calcolare la Complessità Temporale di un Algoritmo

- ✓ A volte il caso medio è utile se l'algoritmo deve essere eseguito **moltissime volte** perché il caso peggiore e quello migliore si compensano tra loro nel corso del tempo. E' comunque preferibile analizzare sempre anche il caso peggiore, perlomeno per evitare il rischio di sviluppare un **"algoritmo eterno"**.

	costo	num. esecuzione
1    output=false	C	
2    for (i=0 to M.length-2)	C	
3        for (j=i+1 to M.length-1)	C	
4            if (M[i]==M[j])	C	
5                output=true	C	
6    return output	C	

# Calcolare la Complessità Temporale di un Algoritmo

---

- ✓ Per semplicità usiamo lo stesso **costo c** per ogni istruzione. In realtà, il **costo computazionale** varia a seconda se si tratta di un'assegnazione, di un confronto, ecc.
- ✓ Inoltre, per semplicità sto utilizzando uno **pseudocodice**. Un algoritmo scritto in pseudocodice può comunque essere implementato in un **programma** utilizzando qualsiasi linguaggio di programmazione.
- ✓ L'algoritmo di sopra prende in input un **vettore M** composto da **5 elementi ( length=5 )**.
- ✓ La prima **linea (1)** e **l'ultima (2)** sono eseguite una sola volta durante l'esecuzione del **programma (n=1)**.

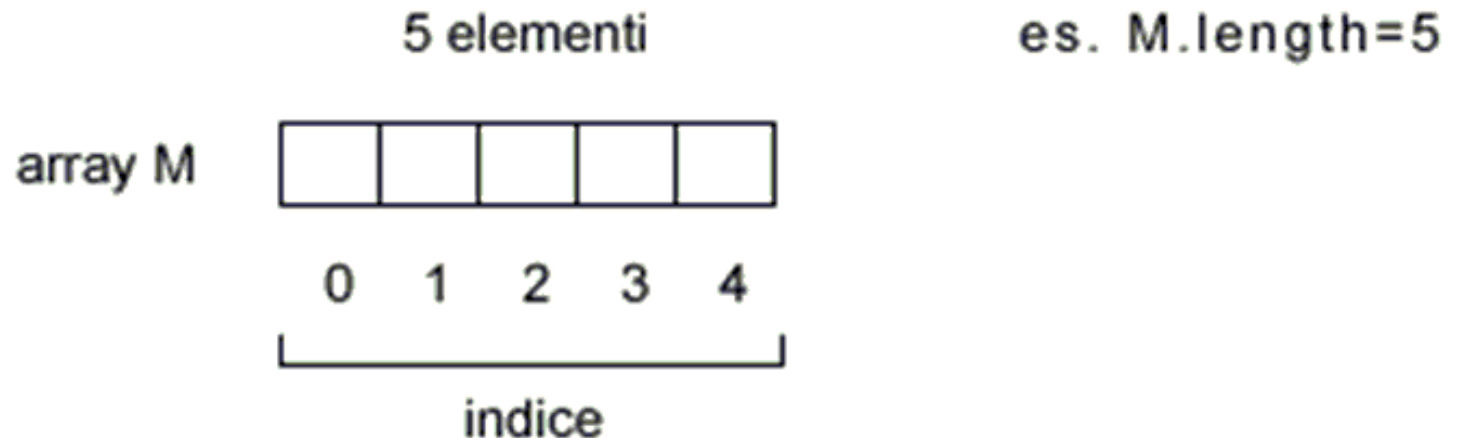
# Calcolare la Complessità Temporale di un Algoritmo

- ✓ Viceversa, il caso migliore non permette di distinguere un algoritmo inefficiente da uno efficiente.

	costo	num. esecuzione
1 <b>output=false</b>	c	1
2 <b>for</b> (i=0 to M.length-2)	c	
3 <b>for</b> (j=i+1 to M.length-1)	c	
4 <b>if</b> (M[i]==M[j])	c	
5 <b>output=true</b>	c	
6 <b>return</b> output	c	1

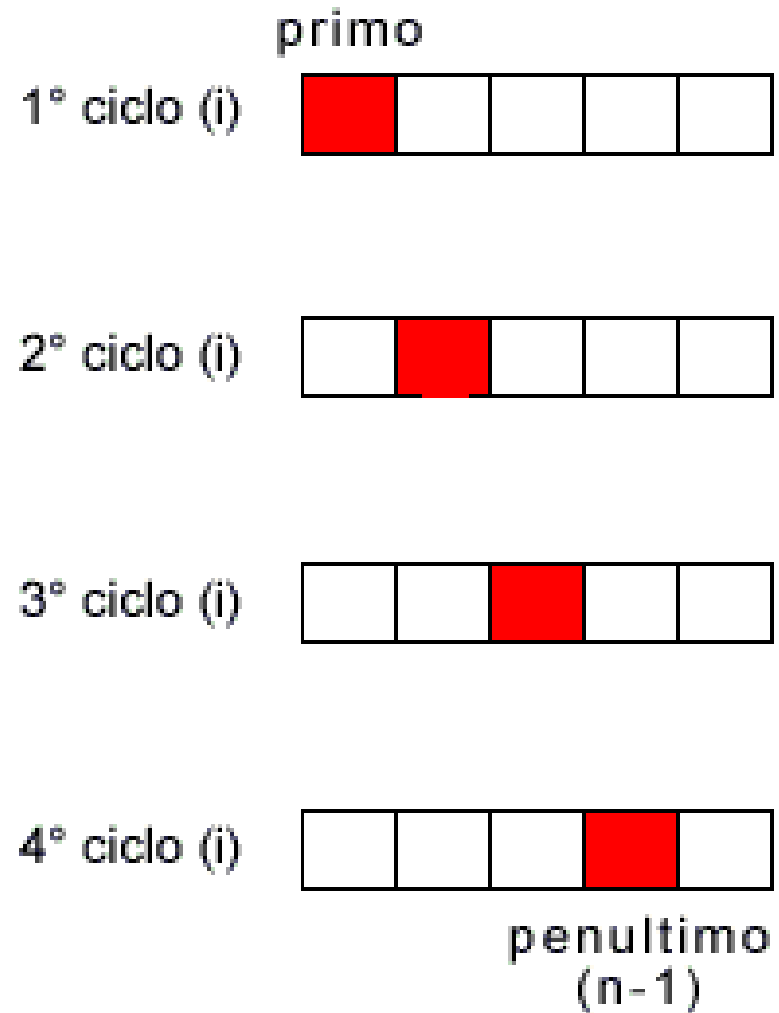
# Calcolare la Complessità Temporale di un Algoritmo

- ✓ Il calcolo si complica un po' quando devo analizzare i cicli.
- ✓ **Il ciclo esterno:** la seconda linea del codice è un'iterazione **for** che va dal primo elemento del vettore ( $i=0$ ) al penultimo ( $i=length-2$ ) ossia  $5-2=3$  ( incluso ).
- ✓ Negli **array** il primo elemento dell'array ha sempre l'indice uguale a 0. Quindi l'ultimo elemento del vettore **M** non è **M[5]** ma **M[4]**. E' opportuno non confondersi.



# Calcolare la Complessità Temporale di un Algoritmo

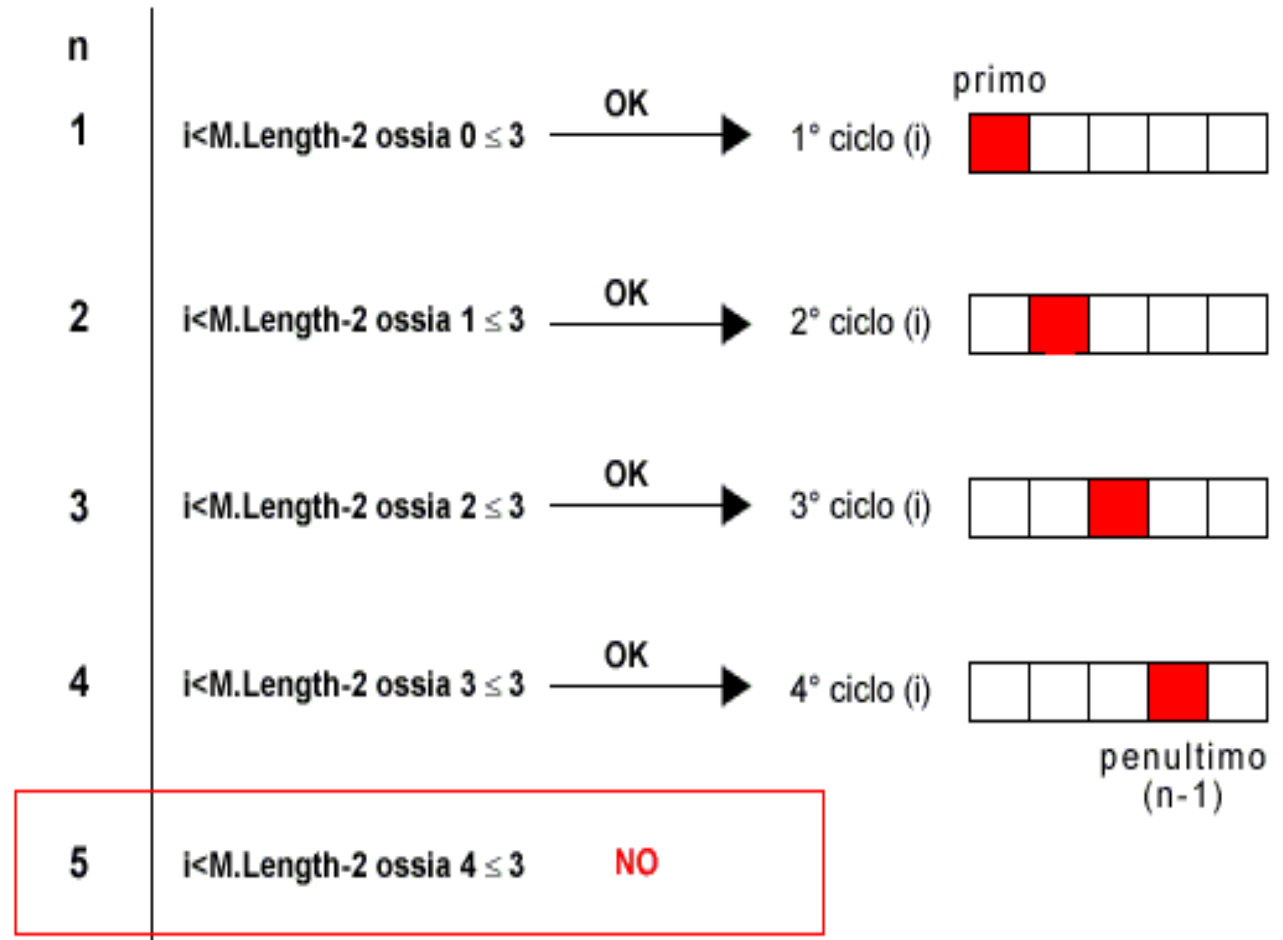
- ✓ Quindi, **il ciclo compie n-1 iterazioni** dove  $n=5$ .
- ✓ Tuttavia, per calcolare la complessità devo considerare anche **il test di chiusura** del ciclo.
- ✓ La riga di codice viene eseguita anche quando la condizione ( $i \leq M.length$ ) non viene soddisfatta e il programma esce dall'iterazione.





# Calcolare la Complessità Temporale di un Algoritmo

- ✓ Quindi, **il ciclo compie n-1 iterazioni** dove  $n=5$ .
- ✓ Pertanto, la seconda linea del codice viene eseguita  $(n-1)+1$  volte ossia **n volte**.
- ✓ A questo punto scrivo n nella colonna vicino alla riga di codice.



# Calcolare la Complessità Temporale di un Algoritmo

```
1  output=false
2  for (i=0 to M.length-2)
3      for (j=i+1 to M.length-1)
4          if (M[i]==M[j])
5              output=true
6  return output
```

costo	num. esecuzione
c	1
c	n
c	
c	
c	
c	1

# Bibliografia

---

[https://it.wikipedia.org/wiki/Teoria\\_della\\_complessit%C3%A0\\_computazionale](https://it.wikipedia.org/wiki/Teoria_della_complessit%C3%A0_computazionale)

<https://www.andreaminini.com/informatica/algoritmo/complessita-algoritmo>