

NOTAZIONE ASINTOTICA

PIETRO DI LENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
UNIVERSITÀ DI BOLOGNA

ALGORITMI E STRUTTURE DI DATI
ANNO ACCADEMICO 2021/2022



INTRODUZIONE

- **Scopo:** analizzare il tempo di calcolo e l'occupazione di memoria degli algoritmi in termini di **dimensione dell'input**
- Qual è una buona misura per tempo di calcolo e memoria?
 - Tempo (sec), memoria (MB) \Rightarrow **legati a macchina e linguaggio!**
 - Meglio considerare il **comportamento asintotico** degli algoritmi
- **Comportamento asintotico** di un algoritmo
 - Ignora costanti additive/multiplicative e termini di ordine inferiore
 - Descrive quanto velocemente tempo/memoria crescono rispetto alla dimensione dell'input
 - Ci permette di confrontare le prestazioni di algoritmi differenti che risolvono lo stesso problema, indipendentemente dall'hardware su cui sono eseguiti

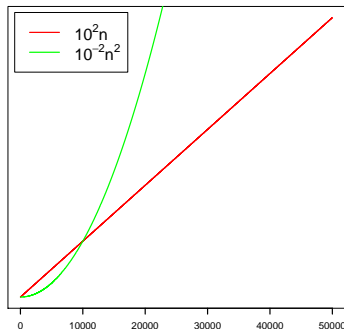
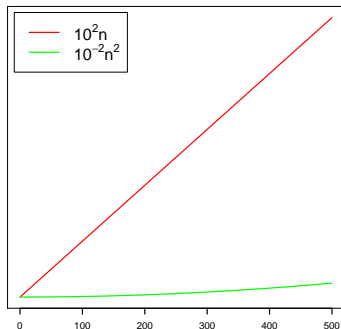
Definizione

Dato $n \geq 0$ indichiamo con $f(n) \geq 0$ la quantità di risorse (tempo di calcolo oppure occupazione di memoria) richiesta da un algoritmo su un input di dimensione n

- Solo valori non-negativi per dimensioni di input e quantità di risorse
- Tipicamente, dimensioni di input intere e costo reale ($f(n) \in \mathbb{R}$)
- Siamo interessati a valutare il **rate di crescita** di $f(n)$
 - Ignoriamo fattori costanti
 - Ignoriamo termini di ordine inferiore

ESEMPIO DI COMPORTAMENTO ASINTOTICO

- Consideriamo due algoritmi A and B per lo stesso problema
 - $f_A(n) = 10^3 n$ è la funzione di costo di A
 - $f_B(n) = 10^{-3} n^2$ è la funzione di costo di B
- Quale algoritmo ha prestazioni migliori in termini di tempo di calcolo?

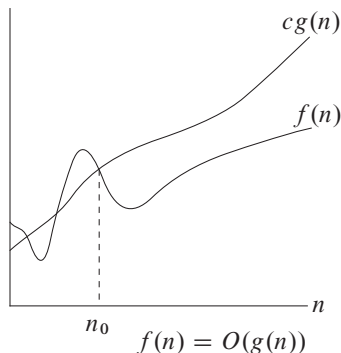


NOTAZIONE ASINTOTICA: O-GRANDE

Definizione (O-grande)

Data una funzione di costo $g(n)$ definiamo l'insieme di funzioni per cui $g(n)$ rappresenta un *limite asintotico superiore* come

$$O(g(n)) = \{f(n) \mid \exists c > 0, n_0 \geq 0 \text{ tale che } \forall n \geq n_0, f(n) \leq cg(n)\}$$



- Intuitivamente $O(g(n))$ è l'insieme di funzioni che hanno ordine di crescita inferiore o uguale a $g(n)$
- $g(n) = O(g(n))$
- Con abuso di notazione, diciamo che $f(n) = O(g(n))$ mentre la notazione corretta sarebbe $f(n) \in O(g(n))$

ESEMPIO: NOTAZIONE O-GRANDE

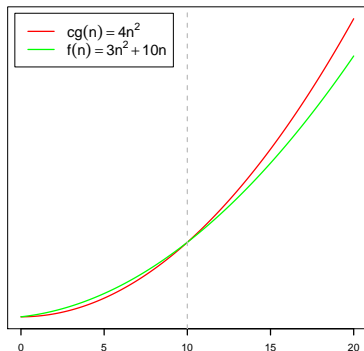
- Siano $g(n) = n^2$ e $f(n) = 3n^2 + 10n$. Dimostriamo che $f(n) = O(g(n))$
- Dobbiamo trovare due costanti $c > 0$ e $n_0 \geq 0$ tali che

$$\forall n \geq n_0, f(n) \leq cg(n) \implies 3n^2 + 10n \leq cn^2$$

- La costante c deve soddisfare la seguente disequazione

$$c \geq \frac{3n^2 + 10n}{n^2} = 3 + \frac{10}{n}$$

- Verificata $\forall c \geq 13$ e $\forall n_0 \geq 1$
- N.B. Possiamo anche scegliere
 $n_0 = 10$ e $c \geq 4$

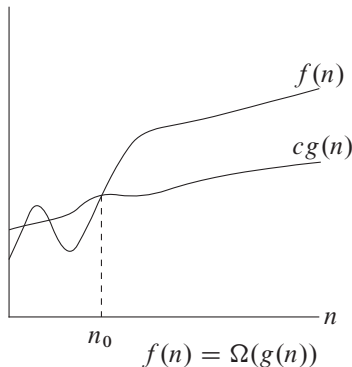


NOTAZIONE ASINTOTICA: OMEGA-GRANDE

Definizione (Ω -grande)

Data una funzione di costo $g(n)$ definiamo l'insieme di funzioni per cui $g(n)$ rappresenta un *limite asintotico inferiore* come

$$\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 \geq 0 \text{ tale che } \forall n \geq n_0, f(n) \geq cg(n)\}$$



- Intuitivamente, $\Omega(g(n))$ è l'insieme di funzioni che hanno ordine di crescita superiore o uguale a $g(n)$
- $g(n) = \Omega(g(n))$

ESEMPIO: NOTAZIONE OMEGA-GRANDE

- Siano $g(n) = n^2$ e $f(n) = n^3 + 2n^2$. Dimostriamo che $f(n) = \Omega(g(n))$
- Dobbiamo cercare due costanti $c > 0$ e $n_0 \geq 0$ tali che

$$\forall n \geq n_0, f(n) \geq cg(n) \implies n^3 + 2n^2 \geq cn^2$$

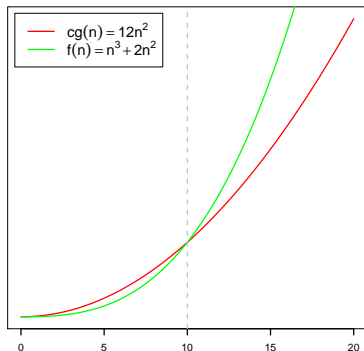
- La costante c deve soddisfare la seguente disuguaglianza

$$c \leq \frac{n^3 + 2n^2}{n^2} = n + 2$$

- Verificata $\forall 0 < c \leq 2, \forall n_0 \geq 0$

- N.B. Possiamo anche scegliere

$$n_0 = 10 \text{ e } c \leq 12$$

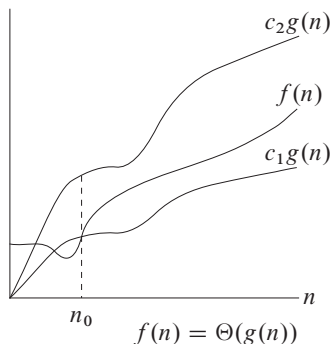


NOTAZIONE ASINTOTICA: THETA

Definizione (Θ)

Data una funzione di costo $g(n)$ definiamo l'insieme di funzioni *asintoticamente equivalenti* a $g(n)$ come

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 \geq 0 \text{ t.c. } \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$



■ Intuitivamente $\Theta(g(n))$ è l'insieme di funzioni il cui ordine di crescita è uguale a quello di $g(n)$

■ $g(n) = \Theta(g(n))$

■ **Teorema.** $f(n) = \Theta(g(n))$
se e solo se

$$f(n) = O(g(n)) \text{ e } f(n) = \Omega(g(n))$$

ESEMPIO: NOTAZIONE THETA

■ Siano $g(n) = n^3$ e $f(n) = n^3 + 2n^2$. Dimostriamo che $f(n) = \Theta(g(n))$

■ Dobbiamo cercare tre costanti $c_1 > 0$, $c_2 > 0$ e $n_0 \geq 0$ tali che

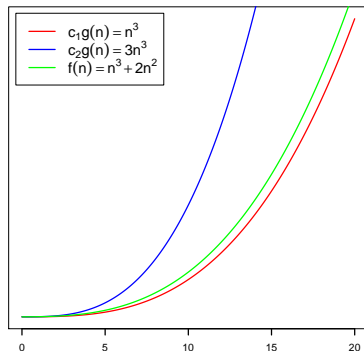
$$\forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \implies c_1 n^3 \leq n^3 + 2n^2 \leq c_2 n^3$$

■ Le costanti c_1, c_2 devono soddisfare le seguenti disuguaglianze

$$c_1 \leq \frac{n^3 + 2n^2}{n^3} = 1 + \frac{2}{n} \leq c_2$$

■ Verificata

$$\forall c_1 \leq 1, c_2 \geq 3 \text{ e } \forall n_0 \geq 1$$



NOTAZIONE ASINTOTICA: o -PICCOLO

Definizione (o -piccolo)

Data una funzione di costo $g(n)$ definiamo l'insieme di funzione che sono *dominate asintoticamente* da $g(n)$ come

$$o(g(n)) = \{f(n) \mid \forall c > 0, \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, f(n) < cg(n)\}$$

- In cosa differisce dalla notazione O -grande?
 - $f(n) = O(g(n)) \Rightarrow f(n) \leq cg(n)$ per qualche costante $c > 0$
 - $f(n) = o(g(n)) \Rightarrow f(n) < cg(n)$ per tutte le costanti $c > 0$
- Per ogni funzione di costo $g(n) \neq o(g(n))$
 - Esempio: $2n = o(n^2)$, $2n^2 \neq o(n^2)$
- Per definizione, se $f(n) = o(g(n))$ allora $f(n) = O(g(n))$
 - Il contrario è generalmente non vero

NOTAZIONE ASINTOTICA: ω -PICCOLO

Definizione (ω -piccolo)

Data una funzione di costo $g(n)$ definiamo l'insieme di funzioni che *dominano asintoticamente* $g(n)$ come

$$\omega(g(n)) = \{f(n) \mid \forall c > 0, \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, f(n) > cg(n)\}$$

- In cosa differisce dalla notazione Ω -grande?
 - $f(n) = \Omega(g(n)) \Rightarrow f(n) \geq cg(n)$ per qualche costante $c > 0$
 - $f(n) = \omega(g(n)) \Rightarrow f(n) > cg(n)$ per tutte le costanti $c > 0$
- Per ogni funzione di costo $g(n) \neq \omega(g(n))$
 - Esempio: $n^2/2 = \omega(n)$, $n^2/2 \neq \omega(n^2)$
- Per definizione, se $f(n) = \omega(g(n))$ allora $f(n) = \Omega(g(n))$
 - Il contrario è generalmente non vero

NOTAZIONE ASINTOTICA E LIMITI

- L'ordine di crescita asintotico può essere confrontato utilizzando **limiti**
- Se il seguente limite **esiste ed è zero**

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

allora $f(n) = o(g(n)) \implies f(n) = O(g(n))$

- Se il seguente limite **esiste ed è infinito**

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

allora $f(n) = \omega(g(n)) \implies f(n) = \Omega(g(n))$

- Se il seguente limite **esiste ed è una costante positiva**

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k > 0$$

allora $f(n) = \Theta(g(n))$

se $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 \rightarrow O \text{ / o/} \\ \infty \rightarrow \Omega \text{ / omega/} \\ k > 0 \rightarrow \Theta \text{ / theta/} \end{cases}$

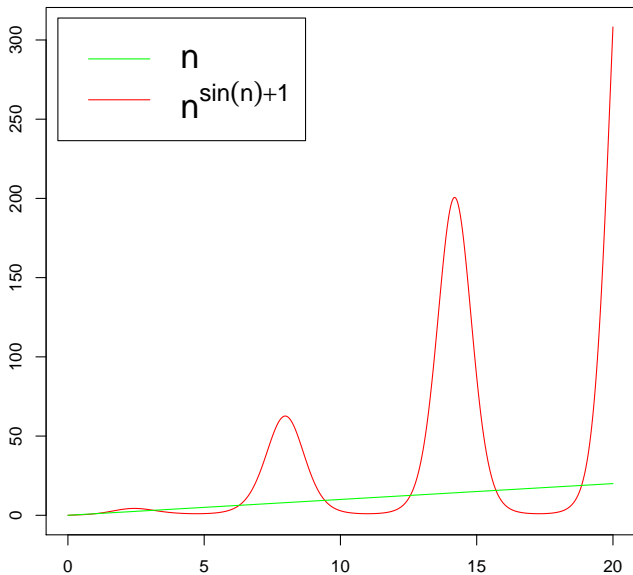
INTERPRETAZIONE INTUITIVA

- Interpretazione del confronto tra ordine di crescita asintotica di funzioni in analogia con il confronto tra numeri reali

Funzioni	Numeri reali
$f(n) = O(g(n))$	$f \leq g$
$f(n) = o(g(n))$	$f < g$
$f(n) = \Omega(g(n))$	$f \geq g$
$f(n) = \omega(g(n))$	$f > g$
$f(n) = \Theta(g(n))$	$f = g$

- Comunque, a differenza di quanto avviene nel dominio dei numeri reali, non tutti gli ordini di crescita asintotica sono confrontabili
 - Se a, b sono numeri reali, solo una tra $a < b, a = b, a > b$ è vera
 - $f(n) = n, g(n) = n^{\sin(n)+1}$ non sono confrontabili
 - $f(n) \neq O(g(n))$ poiché quando $\sin(n) = -1$ allora $g(n) = 1$
 - $f(n) \neq \Omega(g(n))$ poiché quando $\sin(n) = 1$ allora $g(n) = n^2$

FUNZIONI NON CONFRONTABILI



ALCUNE PROPRIETÀ DELLA NOTAZIONE ASINTOTICA

Transitività

$$f(n) = O(g(n)) \text{ e } g(n) = O(h(n)) \implies f(n) = O(h(n))$$

Vale anche per Ω , Θ , o e ω

Riflessività

$$f(n) = O(f(n))$$

Vale lo stesso per Ω e Θ ma non per o e ω

Simmetria

$$g(n) = \Theta(f(n)) \iff f(n) = \Theta(g(n))$$

Simmetria trasposta

- $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$
- $f(n) = o(g(n)) \iff g(n) = \omega(f(n))$

ALCUNE REGOLE UTILI

Somma

Se $f_1(n) = O(g_1(n))$ e $f_2(n) = O(g_2(n))$ allora

$$f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$$

Prodotto

Se $f_1(n) = O(g_1(n))$ e $f_2(n) = O(g_2(n))$ allora

$$f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$$

Moltiplicazione per una costante

Se $f(n) = O(g(n))$ e $a > 0$ allora

$$a \cdot f(n) = O(g(n))$$

NOTAZIONE ASINTOTICA IN EQUAZIONI

- Come dovremmo interpretare la seguente formula?

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$$

Esiste qualche funzione $f(n)$ in $\Theta(n)$ tale che

$$2n^2 + 3n + 1 = 2n^2 + f(n)$$

- Come dovremmo interpretare la seguente formula?

$$2n^2 + \Theta(n) = \Theta(n^2)$$

Esiste qualche funzione $f(n)$ in $\Theta(n)$ tale che

$$2n^2 + f(n) = \Theta(n^2)$$

- Allo stesso modo, possiamo utilizzare in equazioni anche O , Θ , o e ω

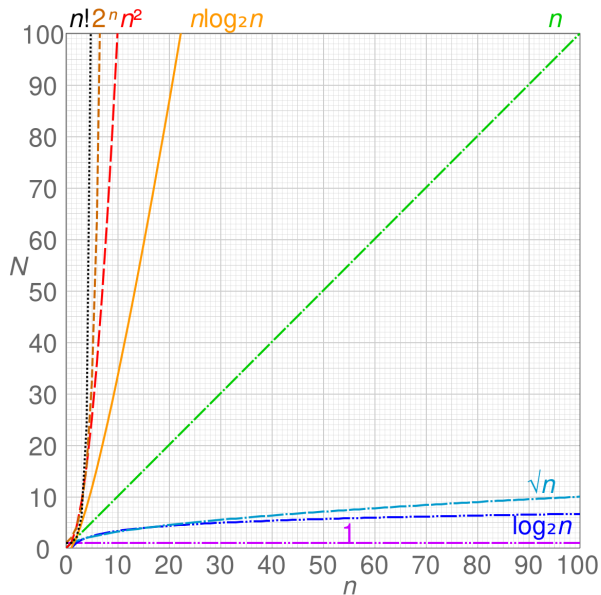
ORDINI DI CRESCITA (MOLTO COMUNI)

Ordine di crescita	Nome
$O(1)$	Costante
$O(\log n)$	Logaritmico
$O(\log^k n)$	Polilogaritmico, $k \geq 1$
$O(n^k)$	Sublineare, $0 < k < 1$
$O(n)$	Lineare
$O(n \log n)$	Pseudolineare
$O(n^k)$	Polinomiale, $k > 1$
$O(n^2)$	Quadratico, per $k = 2$
$O(n^3)$	Cubico, per $k = 3$
$O(c^n)$	Esponenziale, base $c > 1$
$O(n!)$	Fattoriale
$O(n^n)$	Esponenziale, base n

N.B. Utilizziamo la seguente notazione per i logaritmi

- $\log = \log_2 n$
- $\log^k n = (\log n)^k$

CONFRONTO TRA ORDINI DI CRESCITA



COSTO E COMPLESSITÀ COMPUTAZIONALE

Definizione (costo computazionale)

Un **algoritmo** \mathcal{A} ha **costo computazionale** $O(f(n))$ rispetto ad una certa risorsa di calcolo se la quantità di risorse necessaria per eseguire \mathcal{A} su un qualsiasi input di dimensione n è $O(f(n))$.

Definizione (complessità computazionale)

Un **problema** \mathcal{P} ha **complessità computazionale** $O(f(n))$ rispetto ad una certa risorsa di calcolo se esiste un algoritmo che risolve \mathcal{P} con costo computazionale $O(f(n))$ rispetto a tale risorsa di calcolo.

- Le **risorse di calcolo** principalmente considerate solo il **tempo di esecuzione** oppure l'**occupazione di memoria**
- Le due definizioni sono valide anche per le altre notazioni asintotiche

ANALISI DEL CASO OTTIMO, PESSIMO E MEDIO

- Spesso abbiamo bisogno di esprimere quali siano *almeno* (**caso ottimo**), *al peggio* (**caso pessimo**) o *in media* (**caso medio**) le risorse di calcolo richieste da un algoritmo
- **Caso ottimo**: descrive il comportamento in *condizioni ottimali*
 - E.g. ricerca sequenziale quando l'elemento cercato è il primo
- **Caso pessimo**: descrive il comportamento in *condizioni sfavorevoli*
 - E.g. ricerca sequenziale quando l'elemento cercato è l'ultimo
- **Caso medio**: descrive il *comportamento medio* su tutti i possibili input
 - E.g. costo medio di una ricerca sequenziale in una lista
- Quando sviluppiamo algoritmi siamo principalmente interessati a migliorare le prestazioni nel caso pessimo e/o medio

ESEMPIO: VALORE MINIMO

Cercare la posizione del valore minimo in un array

```
1: function MIN(ARRAY  $A[1 \cdots n]$ )  $\rightarrow$  INT  
2:    $m = 1$   
3:   for  $i = 2, \cdots, n$  do  
4:     if  $A[i] < A[m]$  then  
5:        $m = i$   
6:   return  $m$ 
```

- Il loop for viene eseguito $n - 1$ volte con un costo $O(1)$
- Caso ottimo, pessimo e medio coincidono: $O(n)$
- Possiamo essere maggiormente precisi ed utilizzare $\Theta(n)$ (perché?)

ESEMPIO: RICERCA LINEARE

Cercare la posizione di un valore all'interno di un array (-1 se non trovato)

```
1: function SEARCH(ARRAY  $A[1 \cdots n]$ , INT  $x$ )  $\rightarrow$  INT  
2:   for  $i = 1, \dots, n$  do  
3:     if  $A[i] == x$  then  
4:       return  $i$   
5:   return  $-1$ 
```

- **Caso ottimo** (x è il primo elemento): $O(1)$
- **Caso pessimo** (x non presente oppure ultimo): $\Theta(n)$
- **Caso medio**: dobbiamo semplificare il problema
 - **Probabilità uniforme**: x in posizione i con probabilità $P_i = 1/n$
 - Il tempo necessario per ispezionare la posizione i è $T_i = i$

$$\text{Costo medio} = \sum_{i=1}^n P_i T_i = \frac{1}{n} \sum_{i=1}^n i = \frac{1}{n} \frac{n(n+1)}{2} = \Theta(n)$$

- L'analisi ammortizzata è un metodo per valutare il (costo medio di una sequenza di operazioni) \Rightarrow costo ammortizzato
- Differenza tra costo ammortizzato e costo medio
 - costo medio: media del costo di una singola operazione
 - costo ammortizzato: media del costo di una sequenza di operazioni
- Per alcuni algoritmi, una data operazione può essere molto costosa in alcune situazioni e molto efficiente in altre
 - L'analisi del caso pessimo può essere troppo pessimistica
 - L'analisi del caso medio necessita assunzioni probabilistiche non sempre semplici da formulare
- L'analisi ammortizzata complementa l'analisi nel caso pessimo e medio

ESEMPIO: CONTATORE BINARIO

- Operazione di incremento di un numero binario su array
- La cifra più significativa è nella prima posizione dell'array

tutte le
operazioni
sono costanti
quindi vale il
"Solo il
while".

```
1: function INCREMENT(Array A[1...k])
```

```
2:   k cifre  
   i = k
```

```
3:   while i > 1 and A[i] == 1 do } tutti gli 1
```

```
4:     A[i] = 0
```

```
5:     i = i - 1
```

```
6:   if i ≥ 1 then
```

▷ if i = 0 counter overflow

```
7:     A[i] = 1
```

riporta

- Tempo di calcolo nel **caso ottimo**: (quando $A[k]$ contiene zero): $O(1)$
- Tempo di calcolo nel **caso pessimo** (A contiene tutti uno): $O(k)$
- Una sequenza di n incrementi ha un costo limitato da $\Omega(n)$ and $O(nk)$
- Sono queste stime precise per il tempo di calcolo?

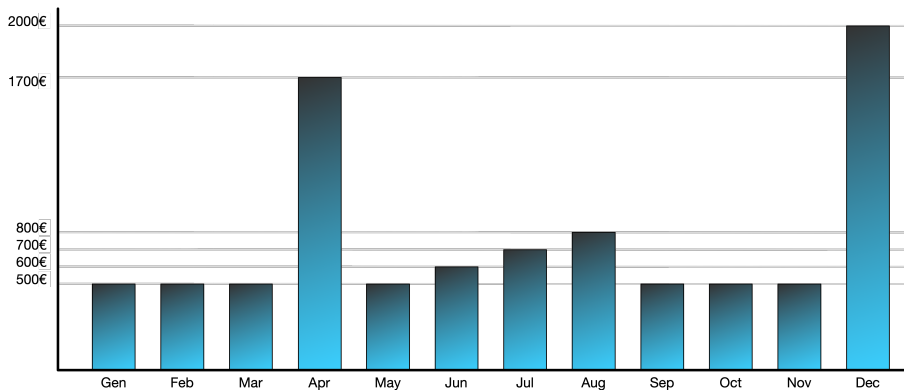
ESEMPIO: CONTATORE BINARIO

Value	A[1]	A[2]	A[3]	A[4]	A[5]	Cost
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	0	0	1	0	2
3	0	0	0	1	1	1
4	0	0	1	0	0	3
5	0	0	1	0	1	1
6	0	0	1	1	0	2
7	0	0	1	1	1	1
8	0	1	0	0	0	4
9	0	1	0	0	1	1
10	0	1	0	1	0	2

METODI PER L'ANALISI AMMORTIZZATA

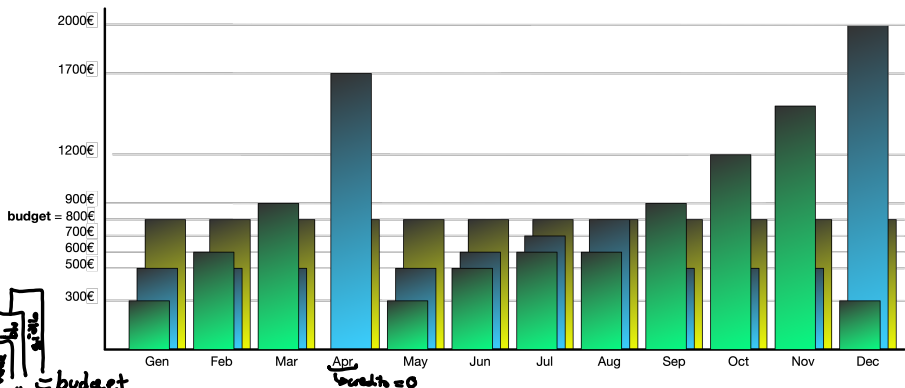
- Metodo dell'aggregazione: determiniamo un **limite superiore** al **costo totale** di una sequenza di n operazioni e dividiamo per n
- Metodo degli accantonamenti: metodo basato sulla **contabilità**
 - Assegniamo un **costo ammortizzato** ad ogni operazione
 - Ogni operazione viene addebitata con il suo costo ammortizzato
 - Dopo ogni operazione, salviamo come **credito** la differenza tra il suo costo ammortizzato e costo reale
 - Accumuliamo il credito collezionato durante l'esecuzione
 - Se il costo reale è più alto del costo ammortizzato, usiamo il credito
 - Il costo ammortizzato è **corretto** se il **credito non è mai negativo**
- Ognuno dei due metodi può essere più o meno adatto ad un problema
 - Metodo dell'aggregazione: ideale se il costo totale è ben definito
 - Metodo degli accantonamenti: ideale se ci sono diverse operazioni

AGGREGAZIONE VS ACCANTONAMENTI



- Il limite superiore è 2000€ per mese \Rightarrow 24000€ per anno
- Metodo dell'aggregazione: somma i costi mensili e dividi per 12
 - Il costo ammortizzato è $9300\text{€}/12 = 775\text{€}$ per mese

AGGREGAZIONE VS ACCANTONAMENTI



budget

■ Metodo degli accantonamenti: stimiamo un budget di 800€ al mese mese che usiamo per pagare i costi e salvare credito

- Il credito è usato per pagare i costi mensili che sfiorano il budget
- Il credito non deve mai essere negativo altrimenti siamo falliti
- Notiamo che se il budget è di 775€ al mese falliamo ad Aprile
- Il costo ammortizzato è quindi 800€ al mese

costi
=
credito
+
0

ANALISI AMMORTIZZATA CON AGGREGAZIONE

- Consideriamo una sequenza di n operazioni INCREMENT
- Metodo dell'aggregazione: sommiamo i costi per i cambi di bit
 - Il k -esimo bit è cambiato ad ogni incremento *(meno significativo)*
 - Il $(k-1)$ -esimo bit è cambiato ogni due incrementi
 - Il $(k-2)$ -esimo bit è cambiato ogni quattro incrementi
 - Il costo totale di n operazioni è quindi limitato da

$$n + n/2 + \dots + n/2^{k-1} = \sum_{i=0}^{k-1} \frac{n}{2^i} \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} = n \frac{1}{1 - 1/2} = 2n$$

costo pessimo di n operazioni di incremento

N.B. Abbiamo solo k bit da cambiare anche se $n > 2^k$ (overflow)

- Il **costo pessimo totale** di una sequenza di n operazioni è $O(n)$
- Il **costo ammortizzato** per operazione è dunque $\frac{O(n)}{n} = O(1)$

ANALISI AMMORTIZZATA CON ACCANTONAMENTI

- Consideriamo una sequenza di n operazioni INCREMENT
- Metodo degli accantonamenti: addebitiamo un costo ammortizzato di 2€ per cambiare ad 1 un bit con valore 0
 - Usiamo 1€ per pagare il cambio ad 1
 - Salviamo il restante 1€ come credito
 - Il credito sarà usato successivamente per cambiare il bit a 0
 - Addebitiamo un costo solo ai cambi ad 1, non a quelli a 0
- **Teorema.** In ogni momento ogni 1 nell'array ha un 1€ di credito
 - numero di 1 mai negativo \Rightarrow credito residuo mai negativo
- **Teorema.** Una sequenza di n incrementi costa $2n$ €
 - Un singolo incremento cambia un solo bit da 0 ad 1
 - Ogni cambio da 0 ad 1 costa 2€ \Rightarrow costo totale uguale a $2n$ €
- Il **costo ammortizzato** di ogni incremento è dunque $\frac{2n}{n} = O(1)$

ANALISI AMMORTIZZATA CON ACCANTONAMENTI

Valore	A[1]	A[2]	A[3]	A[4]	A[5]	Credito residuo	Costo totale
0	0	0	0	0	0	0	0
1	0	0	0	0	¹ 1€ 1	1	2
2	0	0	0	² 1€ 1	0€ 0	1	4
3	0	0	0	³ 1€ 1	⁴ 1€ 1	2	6
4	0	0	⁵ 1€ 1	0€ 0	0€ 0	1	8
5	0	0	⁶ 1€ 1	0€ 0	⁷ 1€ 1	2	10
6	0	0	⁸ 1€ 1	⁹ 1€ 1	0€ 0	2	12
7	0	0	¹⁰ 1€ 1	¹¹ 1€ 1	¹² 1€ 1	3	14
8	0	¹³ 1€ 1	0€ 0	0€ 0	0€ 0	1	16
9	0	¹⁴ 1€ 1	0€ 0	0€ 0	¹⁵ 1€ 1	2	18
10	0	¹⁶ 1€ 1	0€ 0	¹⁷ 1€ 1	0€ 0	2	20