ARITMETICA DEGLI O-grande

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

$$O(f(n)) + O(g(n)) = \max(O(f(n)), O(g(n)))$$

↑ SOLO SE IL NUMERO DI ADDENDI È FISSATO

LA COSA CORRETTA È: ALTRIMENTI:

$$O(f(n)) + O(g(n)) = O(f(n) + g(n))$$

ESEMPIO

$$O(1) + O(1) = \max(O(1), O(1)) = O(1)$$

$$O(1+1) = O(2)$$

ENTRAMBE LE STRADE SONO CORRETTE

MA

$$\underbrace{O(1) + O(1) + \cdots + O(1)}_{n} = O(\underbrace{1 + 1 + \cdots + 1}_{n \text{ volte}}) = O(n)$$

$\neq$ $\max \Rightarrow O(1)$

ESEMPIO: COMPLESSITÀ DI UNA FUNZIONE RICORSIVA + USO DI OPERAZIONI NON ELEMENTARI

```python
def somma_lista_ricorsiva(l):
    if l == []:
        return 0
    else:
        return l[0] + somma_lista_ricorsiva(l[1:])
```

$O(1)$

n = dimensione input (lunghezza l)

- liste l vuota, $n = 0$

$T(0) = O(1) + O(1) = O(1)$

complessità per $n=0$

- liste non vuote

$T(n) = O(n) + T(n-1) + O(1)$

INFINITO DI ORDINE INFERIORE

$$\begin{cases} T(0) = O(1) \\ T(n) = O(n) + T(n-1) \end{cases}$$

$T(3) = O(3) + T(3-1) = O(3) + O(2) + T(1) =$

$T(2) = O(3) + O(2) + O(1) + T(0)$

$= O(3) + O(2) + O(1) + O(1)$

$= O(1) + O(1) + O(1) + O(1)$

$= O(1)$ T non c'è!!

$T(n) =$ ~~~ = UNA COSA SENZA T.

$l[1:] \rightarrow O(n)$

somma_lista_ricorsiva

$T(n-1)$

$l[0] +$ somma list

$O(1)$

$T(n)$ è il tempo che impiega la funzione somma_liste_ricorsiva quando l'input ha lunghezza n

$$\begin{cases} T(0) = O(1) \\ T(n) = O(n) + T(n-1) \end{cases}$$

$$T(n) = O(n) + T(n-1) = O(n) + O(n-1) + T(n-2)$$

$$= O(n) + O(n-1) + O(n-2) + T(n-3) + \dots$$

$$= O(n) + O(n-1) + O(n-2) + \dots + O(2) + T(1) =$$

$$= O(n) + O(n-1) + O(n-2) + \dots + O(1) + T(0) =$$

$$= O(n) + O(n-1) + O(n-2) + \dots + O(1) + O(1)$$

(n addendi)

$$= O(n + (n-1) + (n-2) + (n-3) + \dots + 1)$$

$$= O\left(\frac{(n+1)\cdot n}{2}\right) = O\left(\frac{n^2}{2} + \frac{n}{2}\right) = O(n^2)$$

FORMULA DI GAUSS PER LA SOMMA DEI NUMERI DA 1 FINO AD N

Esempio: Ricerca Binaria

```python
def binary_search_aux(l, v, start, end):
    """
    Funzione ausiliaria di binary_search. Restituisce u
    porzione di lista l che va dalla posizione start al
    inclusi) se esiste, altrimenti restituisce -1. La l

    Attenzione, non è detto che venga restituita la pri
    """
    if start > end:  O(1)
        return -1
    mid  = (start + end) // 2  O(1)
    if v == l[mid]:    O(1)
        return mid
    elif v > l[mid]:  O(1)
        return binary_search_aux(l, v, mid+1, end)
    else:
        return binary_search_aux(l, v, start, mid-1)
```

n = dimensione input

$end - start + 1$

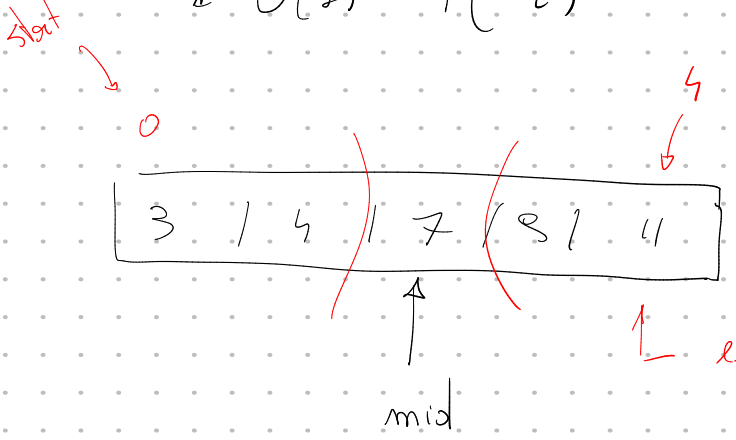
(dimensione della finestra di l che sto osservando)

tempo totale

$$O(1) + T(2^n/2)$$

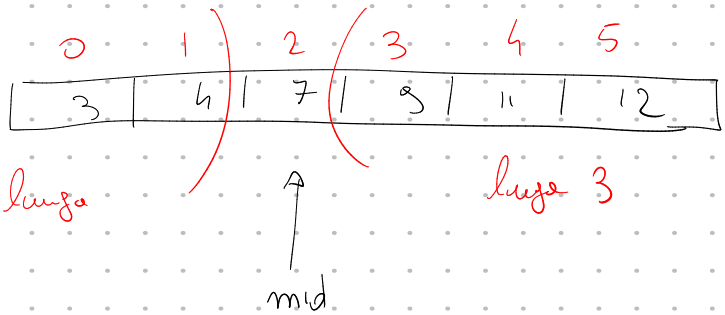
$$\begin{cases} T(0) = O(1) \quad \text{(perché se la finestra di osservazione è vuota il test start > end ha successo ed esco subito)} \\ T(1) = O(1) + T(0) = O(1) + O(1) = O(1) \\ T(2^k) = O(1) + O(1) + O(1) + O(1) + O(1) + T\left(\frac{2^k}{2}\right) \\ \quad = O(1) + T\left(\frac{2^n}{2}\right) \end{cases}$$

Ci limitiamo al caso n = potenza di 2

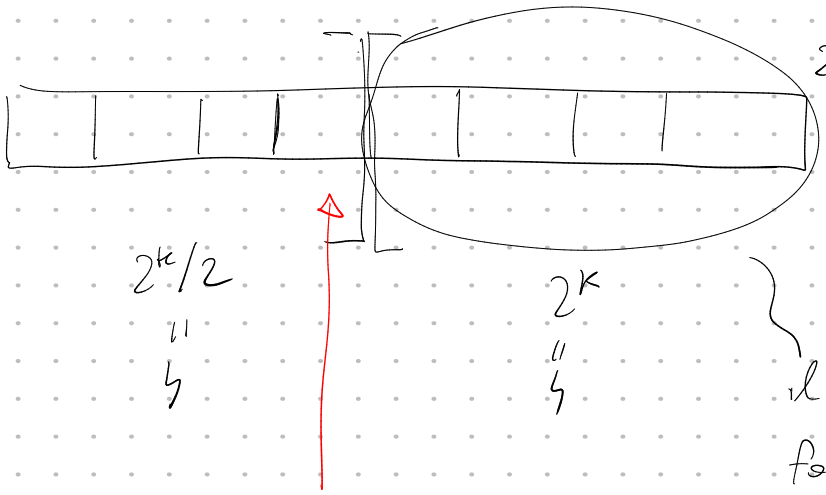


mid = $\frac{0+4}{2} = 2$

↑ entrambe le liste sono di dimensione



mid = $\frac{0+5}{2} = 2{,}5 \simeq 2$



il caso pessimo è quando faccio la chiamata ricorsiva sulla seconda metà della lista

la lista sui cui faccio la chiamata in questo caso è la metà di quella iniziale.

$$\begin{cases} T(0) = O(1) \\ T(2^k) = O(1) + T(2^{k-1}) \end{cases}$$

$2^k/2 = 2^{k-1}$

SOMMA_LISTA_RICORSIVA

$$\begin{cases} T(0) = O(1) \\ T(n) = O(n) + T(n-1) \end{cases}$$

$$T(2^k) = O(1) + T(2^k/2) = O(1) + T(2^{k-1}) =$$

$$= O(1) + O(1) + T(2^{k-2})$$

$$= O(1) + O(1) + O(1) + T(2^{k-3})$$

$8 = 2^3$

$\frac{8}{2} = 4 = 2^{3-1}$

$2^{k-k} = 2^0 = 1$

$$= \underbrace{O(1) + O(1) + \cdots + O(1)}_{k} + T(2^{k-k})$$

$$= O(1) + O(1) + \cdots + O(1) + T(1) \quad (T(1) = O(1))$$

$$= \underbrace{O(1 + 1 + \cdots + 1)}_{k \text{ volte}} = O(k)$$

$T(2^k) = O(k)$ $\quad 2^k = n \rightsquigarrow k = \log_2 n$

$T(n) = O(\log_2 n)$ COMPLESSITÀ RICERCA BINARIA

$T(n) = O(n)$ COMPLESSITÀ RICERCA LINEARE

$\log(2n) = \log n + 1$