# Introduzione agli Algoritmi
# **Esame Scritto a canali unificati con spunti per la soluzione**

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
18 Febbraio 2025

**Esercizio 1:** Si determini l'equazione di ricorrenza che cattura il costo computazionale della seguente funzione Python e la si risolva con un metodo risolutivo a scelta tra quelli studiati a lezione:

```
def es1( n ):
    if n < 1:
        return 0
    i = 2*n
    s = 0
    while i >= 1:
        i = i//2
        s += 1
    return  s* es1(n//2)
```

**Scriviamo l'equazione di ricorrenza, nel caso base e nel caso generale.**

**Se $n \leq 1$, non si entra nell'if e quindi la funzione non esegue alcuna istruzione. Ne segue il caso base:**

$$T(0) = T(1) = \Theta(1).$$

**Se $n > 1$, la funzione inizializza $i = 2 * n$ e calcola il numero di bit necessari per rappresentare $2 * n$ dividendo $i$ per $2$ in un ciclo $while$. Questo ciclo esegue $\Theta(\log n)$ iterazioni (poiché ad ogni iterazione $i$ viene diviso per $2$).**
**La funzione poi restituisce il prodotto tra $s$ (che contiene il numero di iterazioni effettuate) e il risultato di $es1(n/2)$.**
**Possiamo quindi scrivere l'equazione di ricorrenza come:**

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(\log n)$$

**dove:**

- $T(n)$ **rappresenta il tempo di esecuzione per un input di dimensione $n$.**
- $T(n/2)$ **è il tempo di esecuzione per l'input ridotto a metà.**
- $\Theta(\log n)$ **è il costo della parte relativa al ciclo $while$ per calcolare $s$.**

**La soluzione, che non si può ottenere tramite il teorema principlale perché $f(n) = \Theta(logn)$ è più grande ma non polinomialmente di $n^{log_b a} = \Theta(1)$, è $T(n) = \Theta(log^2 n)$.**

**Esercizio 2 (10 punti):** Sia dato un array $A$ di interi. Si scriva un algoritmo che restituisca *True* se $A$ rappresenta uno heap minimo e *False* altrimenti. Ad esempio:

- per $A = [1, 3, 5, 7, 9, 11, 13]$ la risposta è $True$;

- per $A = [1, 3, 5, 7, 2, 11, 13]$ la risposta è $False$, in quanto il nodo con chiave $2$ è figlio del nodo con chiave $3$, ma $2 < 3$.

Dell'algoritmo proposto:

a) si dia la descrizione a parole,
b) si scriva lo pseudocodice,
c) si giustifichi formalmente il costo computazionale.

**Un heap è caratterizzato da due proprietà: essere un albero binario completo o quasi completo e rispettare l'ordinamento verticale. Si osservi che la prima proprietà è automaticamente rispettata per la struttura dati che abbiamo scelto, quindi rimane solo da controllare che per ogni nodo $i$ con indice valido (cioè con almeno un figlio), il valore del nodo sia minore o uguale ai valori dei suoi figli.**
**In un heap binario minimo rappresentato come array, per un nodo con indice $i$:**

- **Il figlio sinistro è in posizione $2i + 1$**
- **Il figlio destro è in posizione $2i + 2$**

**Ecco una semplice procedura iterativa in Python che risolve il problema:**

```
def es2(A):
    n = len(A)
    # Solo i nodi interni devono essere controllati
    for i in range(n // 2):
        left = 2 * i + 1
        right = 2 * i + 2
        # Confronto con il figlio sinistro
        if left < n and A[i] > A[left]:
```

```
            return False
        # Confronto con il figlio destro
        if right < n and A[i] > A[right]:
            return False
    return True
```

**Si osservi che i nodi nell'albero vengono considerati seguendo l'ordine di una visita per livelli.**

**Per quanto riguarda il costo computazionale, la funzione scorre solo i nodi interni (fino a $n/2$) quindi ha costo $O(n)$. Questo risultato va mostrato formalmente, mostrando i contributi di ciascuna istruzione e contando il numero di iterazioni effettuate dal ciclo.**

**Esercizio 3 (10 punti):** Siano dati in input un valore intero $M$ ed il puntatore $L$ alla testa di una lista concatenata i cui record sono costituiti da un campo *key* con valori interi e da un campo *next* contenente un puntatore al record successivo.

Si progetti un algoritmo **ricorsivo** che dia in output la somma di tutte le chiavi della lista concatenata che siano multipli di M, senza modificare la lista.

Ad esempio, se $M = 3$ e la lista puntata da $L$ è la seguente:

$$L \to 3 \to 1 \to 9 \to 2 \to 8 \to 6/$$

l'output dovrà essere 18, in quanto le chiavi che risultano essere multiple di 3 sono -nell'ordine- 3, 9 e 6.

Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi formalmente il costo computazionale dando la ricorrenza che lo caratterizza e poi la sua soluzione.

NOTA BENE: nello pseudocodice della funzione ricorsiva **non** si deve far uso di variabili globali.

**L'idea è semplicissima: per ogni record, se la sua chiave è divisibile per $M$ sarà sommata a ciò che viene restituito, altrimenti si restituisce il risultato della chiamata immutato. Ecco un possibile codice in Python:**

```
def somma(M, r):
    if r == None:
        return 0
if r.key% M == 0:
    return r.key + somma(M, r.next)
else:
    return somma(M, r.next )
```

**La relazione di ricorrenza che descrive il costo dell'algoritmo è quella che caratterizza la maggior parte degli algoritmi ricorsivi su liste puntate, cioè la seguente:**

$$T(n) = T(n-1) + O(1) \text{ se } n > 1, O(1) \text{ altrimenti}$$

**la cui soluzione è $\Theta(n)$ e va risolta formalmente con uno dei metodi risolutivi studiati.**