ALGORITMO	CASO PESSIMO	CASO MEDIO	CASO OTTIMO **	IN PLACE
SELECTION SORT	n^2	n^2	n^2	V
INSERTION SORT	n^2	n^2	n	V
MERGE SORT	$n \log n$	$n \log n$	$n \log n$	X
QUICK SORT (PIVOT RANDOMIZZATO)	n^2	$n \log n$	$n \log n$	V *
BOGO SORT	∞	$n \cdot n!$	n	V

DIVIDE ET IMPERA: MERGE SORT, QUICK SORT

RANDOMIZZATI: QUICK SORT, BOGO SORT

* NON PROPRIAMENTE PER VIA DEI RECORD DI ATTIVAZIONE

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

$S(n) = \Theta(n)$

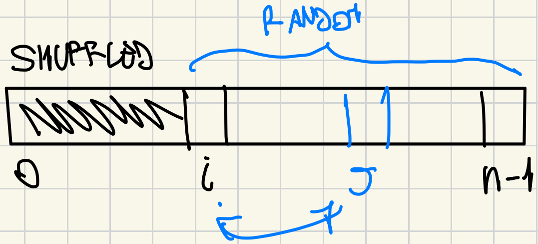
```
BOGOSORT(A)
    WHILE !SORTED(A)
        SHUFFLING(A)
```

FISHER-YATES '38

```
SHUFFLE(A)
    FOR (i = 0; i <= n-2; i++)
        J = RANDOM(i, n-1)
        SWAP(A[i], A[J])
```

$\Theta(n)$ TEMPO

$O(1)$ SPAZIO



C QSORT (GCC)

↳ MERGESORT

↳ HEAP SORT SE NON C'È SUFFICIENTE MEMORIA

JS ARRAY.SORT (WEBKIT)

↳ MERGESORT (CASI BASE $n=4$ CON INSERTION SORT)

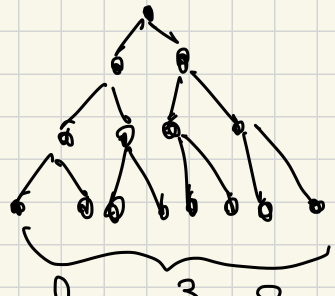
** NON MOLTO INTERESSANTE

$\forall$ ALGORITMO S DI ORDINAMENTO FORZARE UN CASO OTTIMO CON IF !SORTED(A) THEN S(A)

$n \log n$ — n^2 PESSIMO → COMPLESSITÀ IN TEMPO PER L'ORDINAMENTO

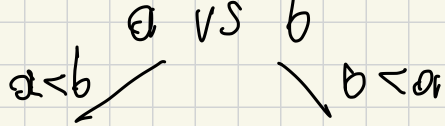
$\geq n$ — OGGI — MERGESORT, QUICKSORT (MEDIO) — SELECTION SORT, INSERTION SORT

$\Omega(n \log n)$

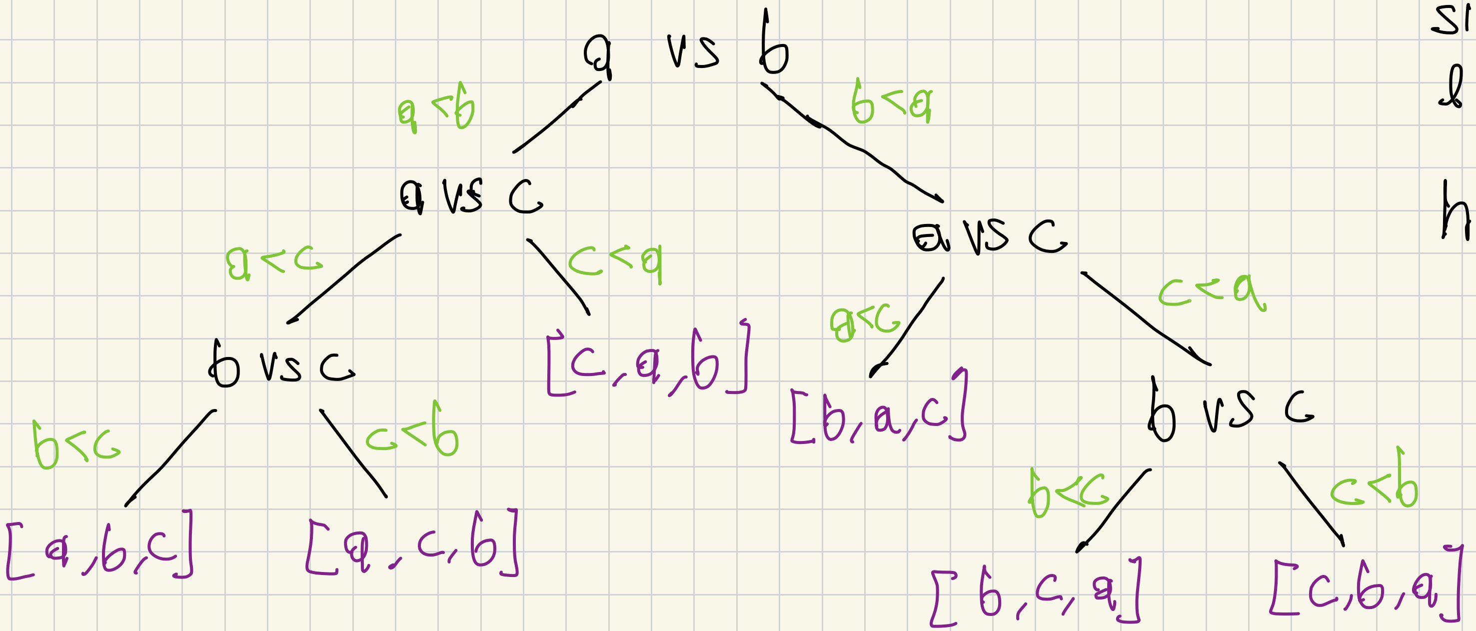


LIMITE INFERIORE PER IL PROBLEMA DELL'ORDINAMENTO

- OPERAZIONE ATOMICA È $<$
- CONFRONTO a CON b : $a < b$ O $b < a$
- ALBERO DI DECISIONE. OGNI NODO È DEL TIPO



- PER $n = 3$ ELEMENTI $A = [a, b, c]$



SIA

l = NUMERO DI FOGLIE ALBERO DI DECISIONE

h = ALTEZZA DELL'ALBERO

$l \geq n!$ PERCHÉ UN QUALSIASI ALG DI ORDINAMENTO DEVE ARRIVARE A UNA DELLE $n!$ PERMUTAZIONI

$l \leq 2^h$ CONSIDERANDO UN ALBERO BINARIO COMPLETO

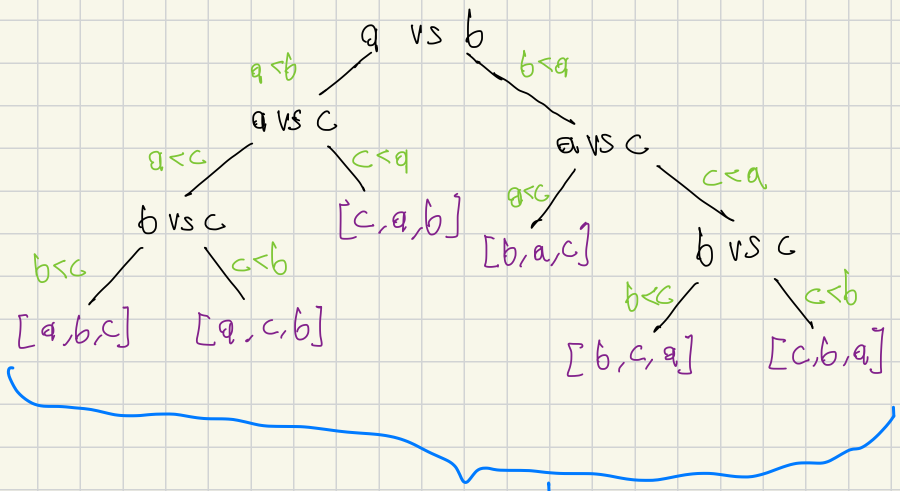
$$\underbrace{n! \le l \le 2^h}$$

$$2^h \ge n!$$

$$\log 2^h \ge \log n!$$

$$h \ge \log n!$$

STIRLING $\quad n! > \left(\frac{n}{e}\right)^n$



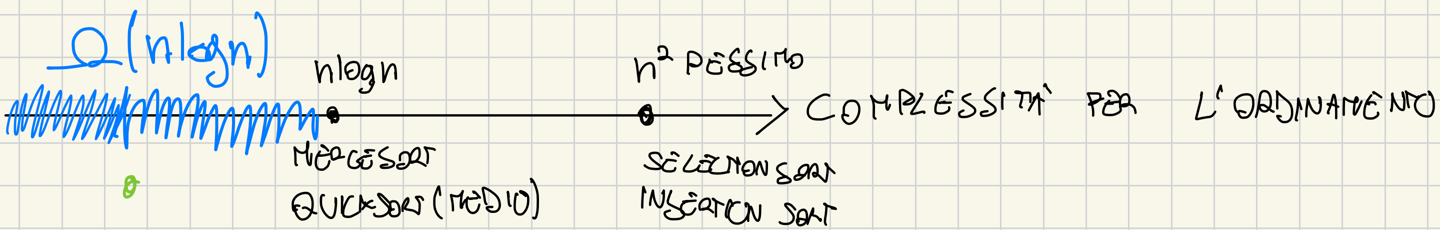
$$n! \le l \le 2^h$$

$$h = \Omega(n \log n)$$

$$h \ge \log\left(\frac{n}{e}\right)^n$$

$$h \ge n \log\left(\frac{n}{e}\right)$$

$$h \ge n(\log n - \log e) = \Omega(n \log n)$$



SECONDO SEMESTRE: ALGORITMI CHE BATTONO QUESTO LIMITE INFERIORE PERCHÉ USANO OPERAZIONI DIVERSE DAI CONFRONTI

PROGETTARE UN ALGORITMO DI TIPO DIVIDE ET IMPERA (COPIE) CHE DATO UN ARRAY A ORDINATO DETERMINA SE A CONTIENE VALORI DUPLICATI

A = [1, 3, 5, 7, 10]

COPIE(A) = FALSE

A = [1, 2, 2, 5, 6, 7]

COPIE(A) = TRUE

CASO BASE: 0/1 ELEMENTO, BANALMENTE NO DUPLICATI

COPIE([1, 2, 5, 8, 8, 9])

COPIE([1, 2, 5])   COPIE([8, 8, 9])

RICOMBINO I RISULTATI RESTITUENDO TRUE SSE

1. SOTTOPROBLEMA SX HA DUPLICATI

OPPURE 2. SOTTOPROBLEMA DX HA DUPLICATI

OPPURE 3. GLI ELEMENTI A CAVALLO SONO UGUALI

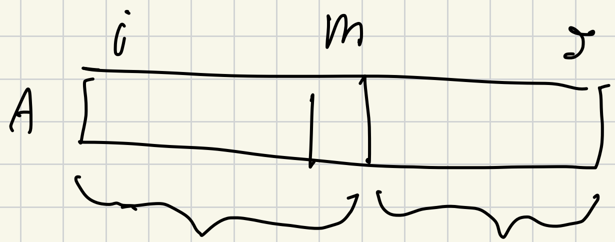
~~FALSE~~ TRUE

COPIE([1, 2, 5]) FALSE   COPIE([5, 6, 7]) FALSE

```
COPIE(A, i, J)
    IF (i ≥ J)
        RETURN FALSE

    m = ⌊(i + J)/2⌋

    RETURN ( A[m] == A[m+1] )
              || COPIE(A, i, m)
              || COPIE(A, m+1, J)
```



$T(n) = 2\,T\left(\frac{n}{2}\right) + \Theta(1) \Rightarrow T(n) = \Theta(n)$ TEMPO

SPAZIO $\Theta(\log n)$ PER LA PILA COI RECORD DI ATTIVAZIONE