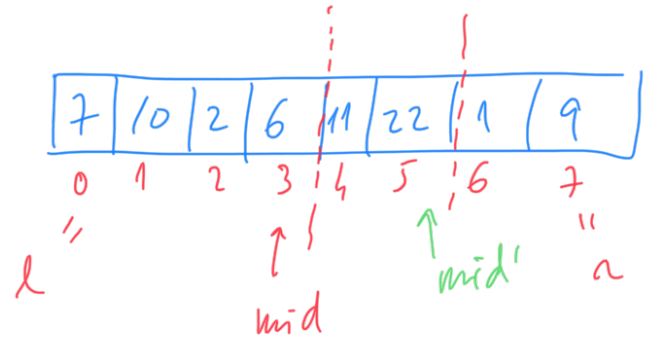
1.



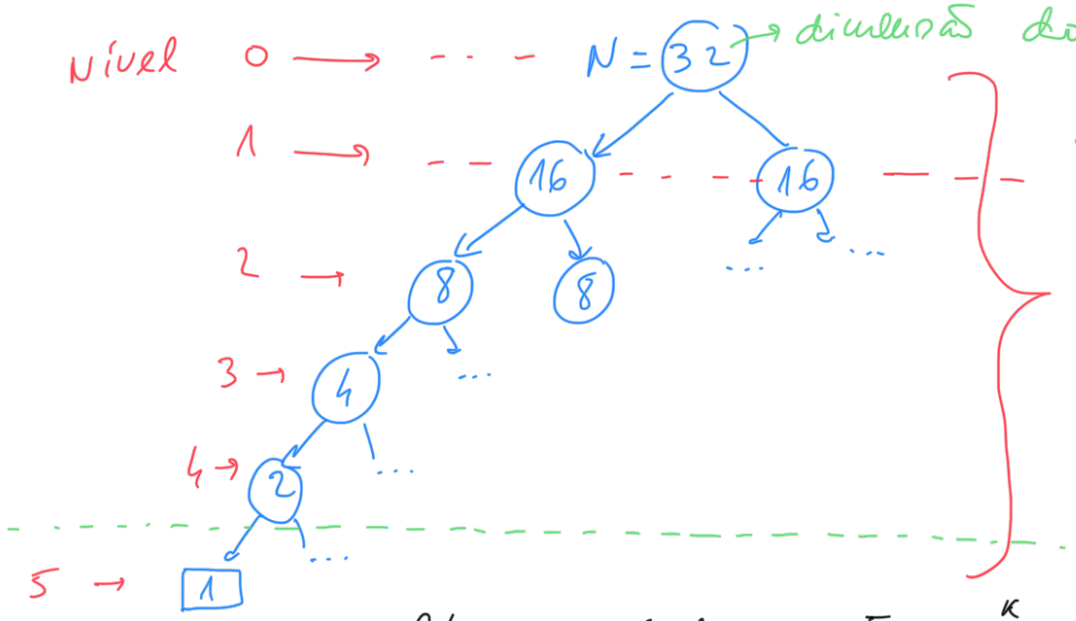
$$mid = \frac{r-l}{2} + \underbrace{l}_{1(r)} \leftarrow \text{referencial}$$

$$= \frac{2l - l + r}{2} = \frac{l+r}{2}$$

$$mid = \frac{0+7}{2} = 3 \qquad mid' = \frac{4+7}{2} = \frac{11}{2} = 5$$

MERGE SORT

Top-down merge sort usa, no máximo, $N \times \log_2 N$ comparações para ordenar um array de dimensão N.



n=5 altura; Nível k tem 2^k subarrays, cada um de dimensão 2^{n-k} → implica 2^{n-k} comparações para fazer o merge

ex: nível 1 tem 2^1 arrays de dim. $2^{5-1} = 2^4 = 16$

Custo por nível: 2^k arrays $\times\ 2^{n-k} = 2^n$

Como temos n níveis, o custo total é:

$$n \times 2^n = N \times \log_2 N$$

($n = \log_2 N$, $2^n = N$)

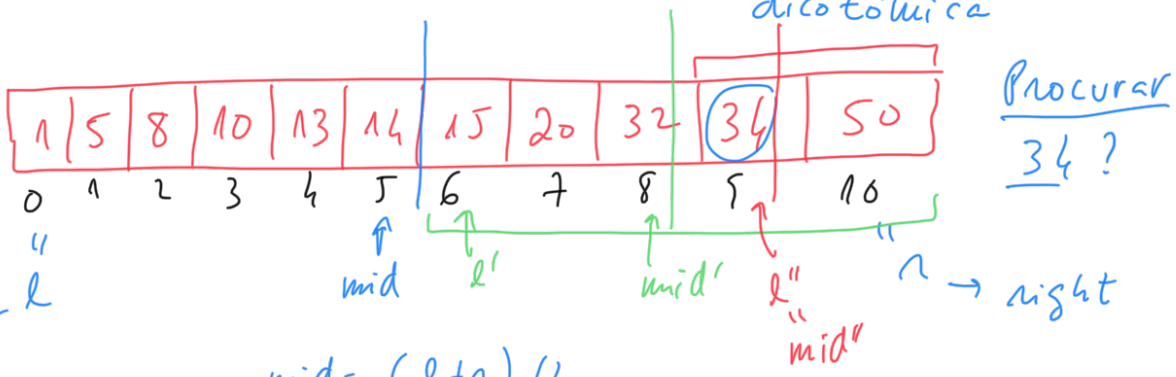
Complexidade temporal

Complexidade espacial: $O(N)$

PRÉ-CONDIÇÃO:

ARRAY TEM DE ESTAR ORDENADO

Binary Search - Pesquisa binária ou dicotómica



mid = (l + r) / 2

34 > array[mid] ? Sim → l' = mid + 1

mid' = (l' + r) / 2 = (6 + 10) / 2 = 8

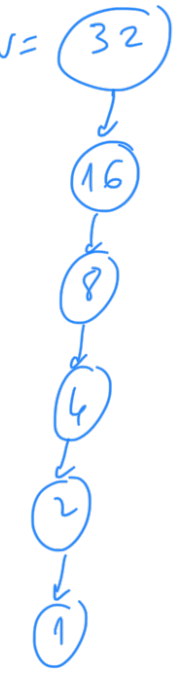
34 > array[mid'] ? Sim → l'' = mid' + 1

mid'' = (l'' + r) / 2 = (9 + 10) / 2 = 9

34 == array[mid''] ? Sim → Encontrou

Binary Search

dim. array $N =$



altura = profundidade do stack (se o algoritmo for recursivo)

$= n = \log_2 N$

Complexidade temporal:

$O(\log_2 N)$

T. P. C.

Lowth Bound → via moodle + GitHub

```
fun binarySearch (array: IntArray,
                  l: Int, r: Int, v: Int)
                  : Int {
```

↖ índice onde encontrou ou -1

```
    if (l <= r) {
        val mid = (l + r) / 2
        if (v == array[mid])
            return mid
        else if (v < array[mid])
            return binarySearch(array, l,
                                mid - 1, v)
        else
            return binarySearch(array,
                                mid + 1, r, v)
    }
}
```

procura à esquerda ←

procura à direita ←