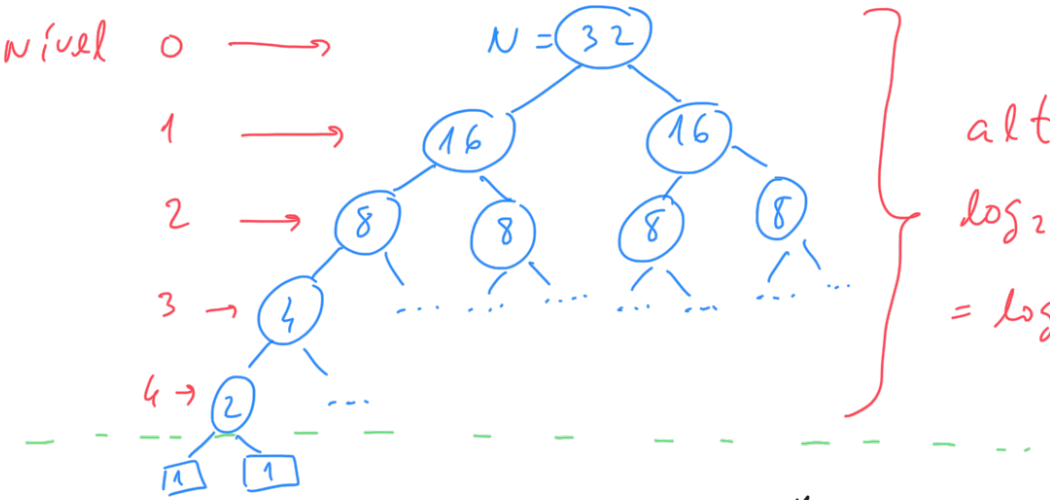
MERGE SORT

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



$\boxed{n=5}$ altura; nível k tem 2^k subarrays, cada um de dimensão 2^{n-k} → implica 2^{n-k} comparações

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:

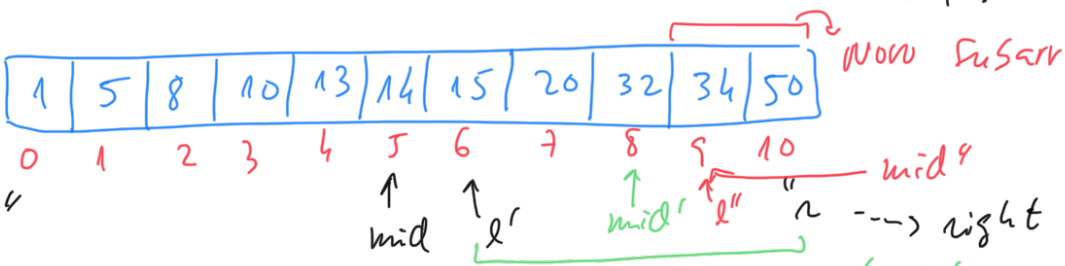
$$\underbrace{n}_{\log_2 N} \times \underbrace{2^n}_{N} = \underbrace{N \times \log_2 N}_{\text{Complexidade temporal}}$$

Complexidade espacial: $O(N)$

Binary search

ARRAY TEM DE ESTAR ORDENADO!

Procurar 34? (2.)



mid = (l + r) / 2 = 5

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

(3.)

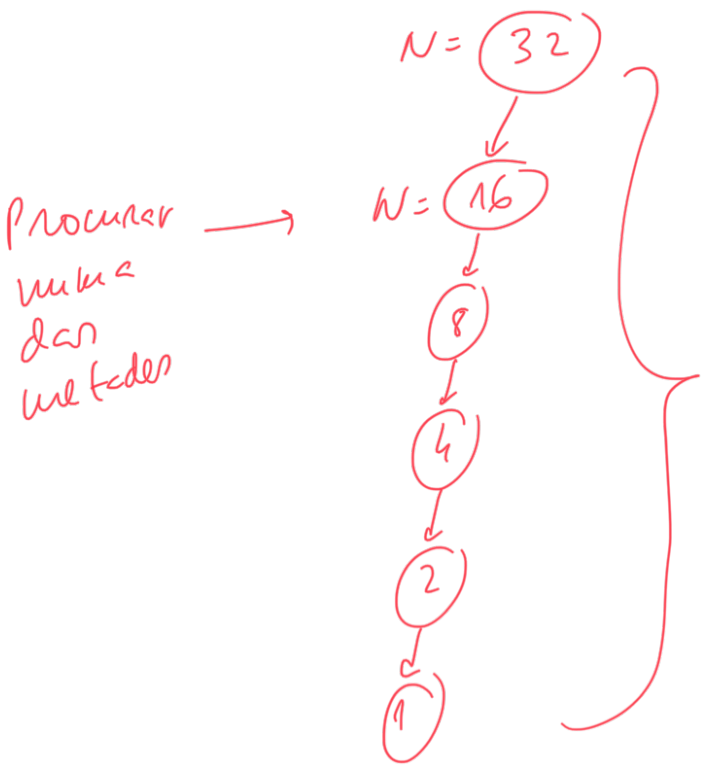
```
fun binarySearch ( array: IntArray , l: Int,
                   r: Int, v : Int ): Int
{
    if ( l <= r ) {
        val mid = ( l + r ) / 2
        if ( v == array[mid] )
            return mid
        else if ( v < array[mid] )
            return binarySearch (array,
                l, mid - 1,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

índice onde encontrou ou -1

procurar à esquerda

procurar à direita

Binary search



altura = nº chamadas recorrentes = $n = \log_2 N$

$= \log_2 2^5 = 5$

Complexidade temporal: $O(\log_2 N)$