

Paradigmas de Linguagens Computacionais

Leonardo Espíndola (lre) e Filipe Nogueira Jordão (fnj2)

Merge Sort

```
1 right :: [Int] -> Int -> [Int]
2 right (x:xs) 1 = xs
3 right (x:xs) n = right xs (n-1)
4
5 left :: [Int] -> Int -> [Int]
6 left (x:xs) 1 = [x]
7 left (x:xs) n = x:(left xs (n-1))
8
9 size :: [Int] -> Int
10 size [] = 0
11 size (x:xs) = (size xs) + 1
12
13 merge :: [Int] -> [Int] -> [Int]
14 merge [] x = x
15 merge x [] = x
16 merge (x:xs) (y:ys)
17   | x <= y = x : merge xs (y:ys)
18   | otherwise = y : merge (x:xs) ys
19
20 mergeSort :: [Int] -> [Int]
21 mergeSort [] = []
22 mergeSort (x:[]) = [x]
23 mergeSort xs = merge (mergeSort (left xs ((size xs) `div` 2))) (mergeSort (right xs ((size xs) `div` 2)))
```

Função de cálculo do tamanho da estrutura de dados

```
size :: [Int] -> Int  
size [] = 0  
size (x:xs) = (size xs) + 1
```

Funções de divisão da estrutura de dados

```
right :: [Int] -> Int -> [Int]
right (x:xs) 1 = xs
right (x:xs) n = right xs (n-1)

left :: [Int] -> Int -> [Int]
left (x:xs) 1 = [x]
left (x:xs) n = x:(left xs (n-1))
```

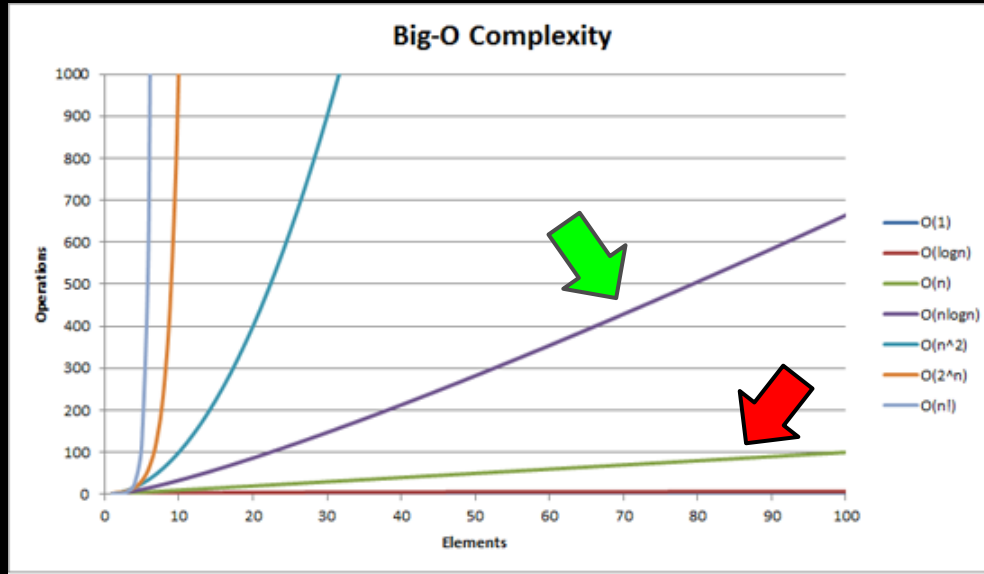
Função de ordenação

```
merge :: [Int] -> [Int] -> [Int]
merge [] x = x
merge x [] = x
merge (x:xs) (y:ys)
  | x <= y = x : merge xs (y:ys)
  | otherwise = y : merge (x:xs) ys
```

Função de chamada para a ordenação por merge

```
mergeSort :: [Int] -> [Int]
mergeSort [] = []
mergeSort (x:[]) = [x]
mergeSort xs = merge (mergeSort (left xs ((size xs) `div` 2))) (mergeSort (right xs ((size xs) `div` 2)))
```

Complexidade Merge Sort



Dúvidas ?