Paradigmas de Linguagens Computacionais

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

Merge Sort

```haskell
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))

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

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

mergeSort :: [Int] -> [Int]
mergeSort [] = []
mergeSort (x:[]) = [x]
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

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

Funções de divisão da estrutura de dados

```haskell
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))
```

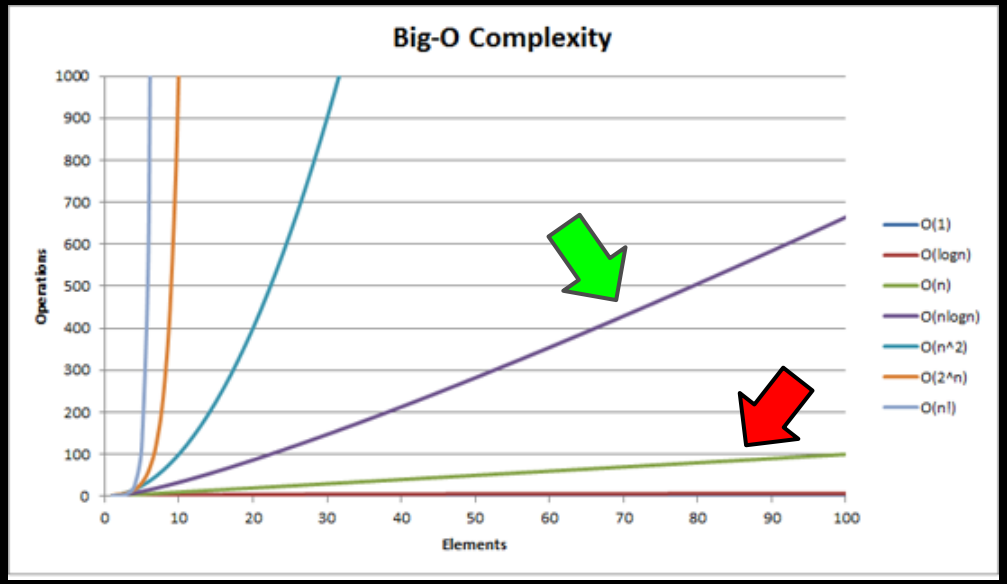
Função de ordenação

```haskell
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

```haskell
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 ?