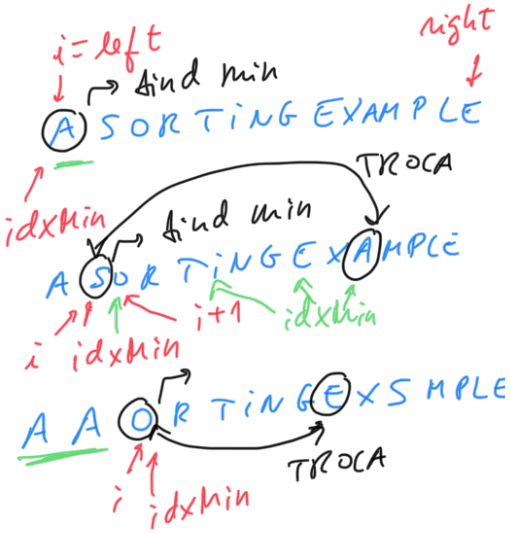
Selection Sort


```
fun selectionSort(a: CharArray, left: Int, right: Int)
{
    for(i in left until right)  // right exclusive
    {
        var idxMin = i  // Guarda o índice do min
        for(j in i+1..right)
        {
            // find min
            if (less(a[j],
                     a[idxMin]))
                idxMin = j
        }
        // TROCA a[idxMin]
        // com a[i]
        exch(a, i, idxMin)
    }
}
```



A A E R T I N G O X S M P L E

'e' ordenado

Complexidade Temporal do Selection Sort

$n = right - left + 1$

nº comparações:

n → cálculo do 1º mínimo
$+$
$n-1$ → " " 2º "
$+$
$n-2$ → " " 3º "
$+$
$\vdots$
$+$
1 = Série aritmética → $O(n^2)$

Soma dos n termos de uma série Aritmética é $\sum_{i=1}^{n} i = \frac{n \times (n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} = O(n^2)$

custo no melhor e no pior caso

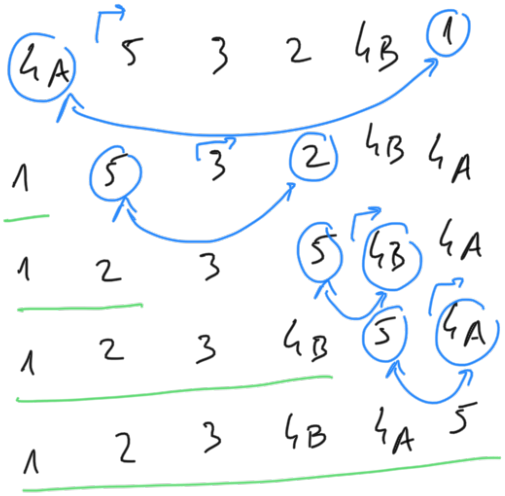
não considerando otimizações ← array já estar ordenado

ordenado decrescentemente

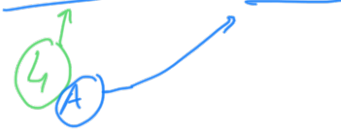
Selection Sort é estável?

Input: 4_A 5 3 2 4_B 1

Output:



→ O elemento "4" tem duas chaves: número e string



A sequência de input está ordenada por "string"

4_A ... 4_B ...

T.P.C. Fazer selection sort estável

↓ Não é estável pois altera a ordem segundo a chave de "string"

↳ Fazer deslocamentos como fez o insertion sort

Bubble Sort

```
fun bubbleSort(a: CharArray, left: Int, right: Int)
{
    for(i in left until right) // right exclusive
    {
        for(j in right downTo i+1 ) { // i+1 inclusive
            lessExch(a, j, j-1)
        }
    }
}
```

Complexidade temporal: $O(n^2)$

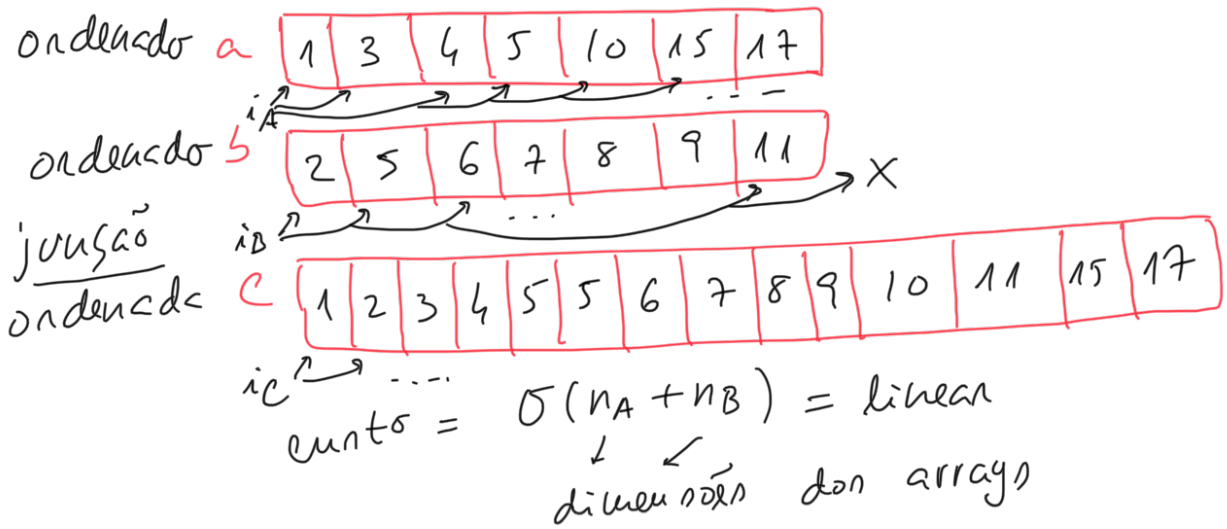
Otimização: Quando faz a 1ª passagem, conta o nº lessExch
⇒ Se = zero, já está ordenado
Usar um boolean

T.P.C. ⟶
O(n) no caso do array estar ordenado
Melhor caso

Merge sort

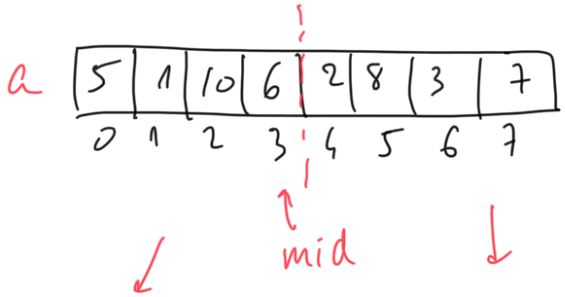
Usa a técnica "Divide to Conquer"
↙
"Dividir para conquistar"

Merge - Junção → aplica-se apenas se ambos os arrays estiverem ordenados



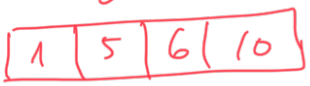
custo = $O(n_A + n_B)$ = linear
↓ ↙
dimensões dos arrays

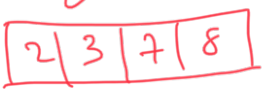
Merge sort



ordenar recursivamente a metade esquerda

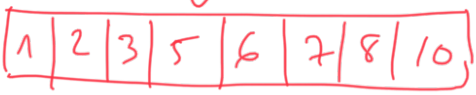
ordenar recursivamente a metade direita





Para ordenar, faz o mesmo, divide em 2 metades e ordena-as recursivamente

merge




```
mergesort(a, l, r):
  if (l < r){
    mid = (l+r)/2
    mergesort(a, l, mid)
    mergesort(a, mid+1, r)
    merge(a, l, mid, r)
  }
```

6.