Algoritmos - Aula 4

Fernando Raposo

Vamos ver

- Selection sort
- Merge sort

Selection Sort

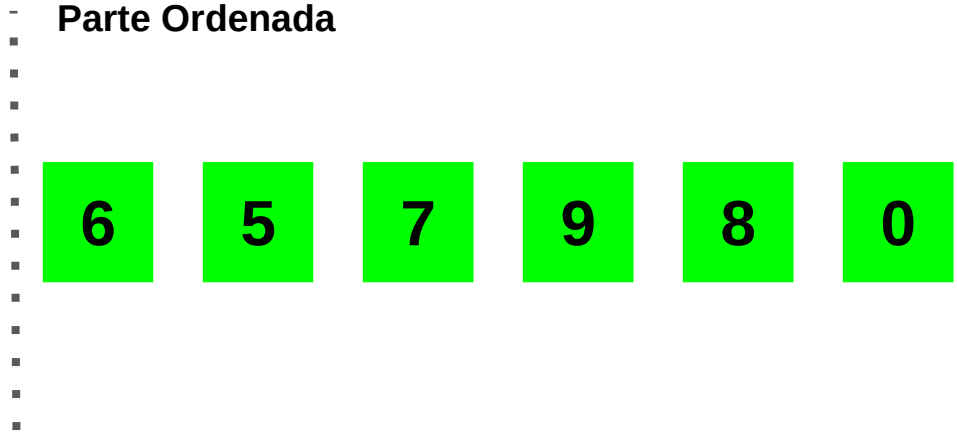
- Algoritmo de ordenação que ordena um array encontrando o menor elemento da parte não ordenada do array e colocando-o na parte ordenada;

Existem **dois sub-arrays** dentro do array:

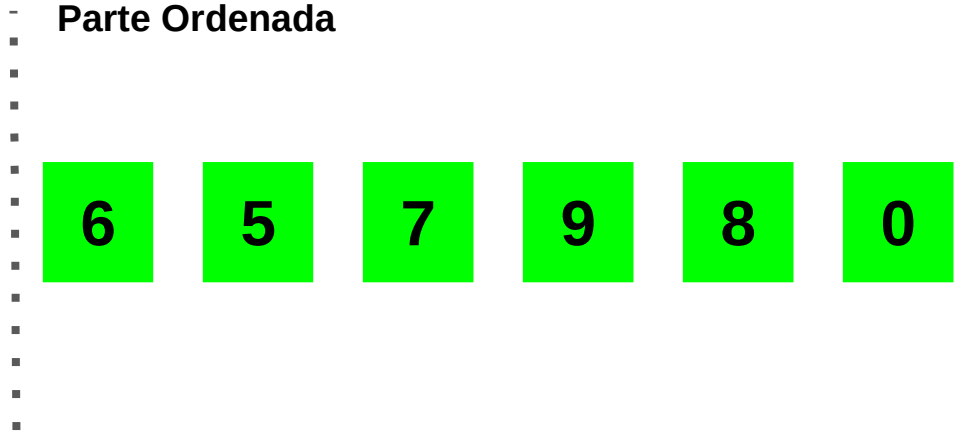
1. O array já ordenado;
2. A parte do array que não está ordenada ainda.

Selection Sort

Selection Sort

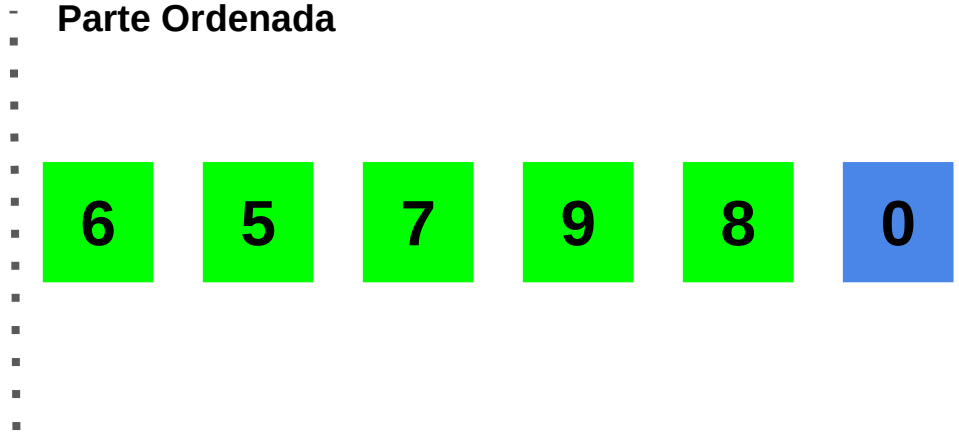


Selection Sort



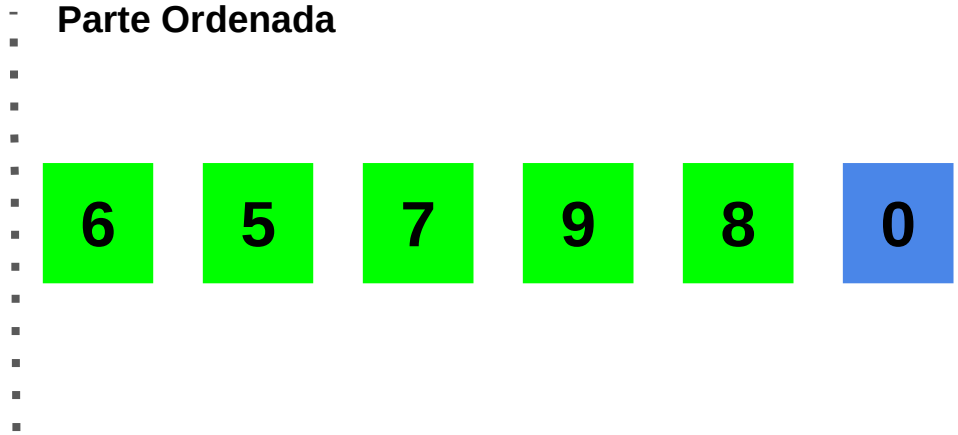
Ache o menor elemento entre as posições 0 e 5

Selection Sort



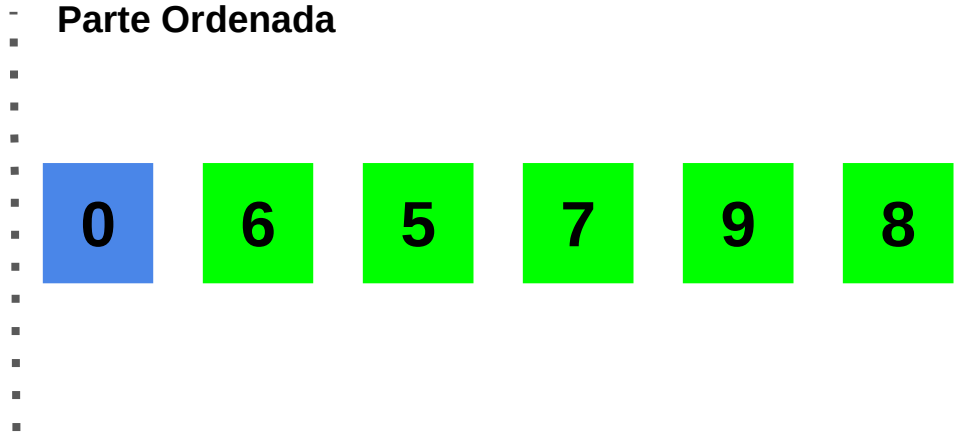
6 não, 5 não, 7 não, 9 não, 8 não, 0! Achado!

Selection Sort



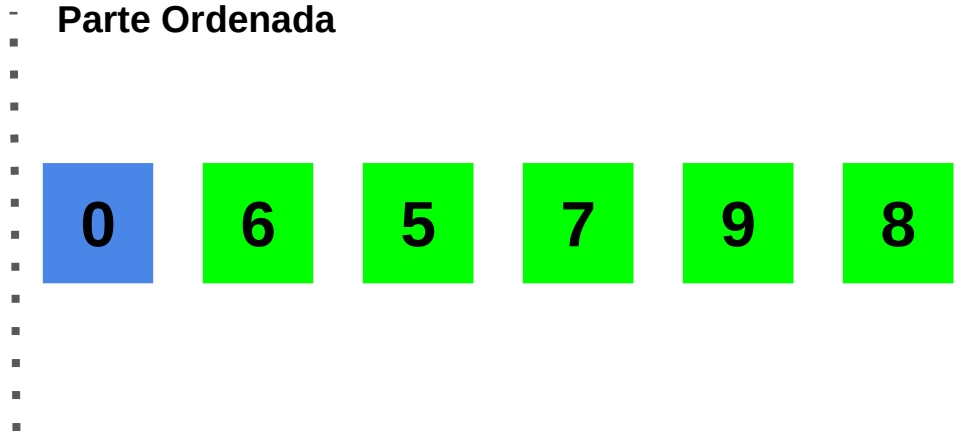
Coloque-o na frente!

Selection Sort



Coloque-o na frente!

Selection Sort



Mova a parte ordenada

Selection Sort

Feito!

Selection Sort

Ache o menor elemento entre as posições 1 e 5

Selection Sort

6 não, 5 Achado!, 7 não, 9 não, 8 não

Selection Sort

Coloque-o na frente + 1!

Selection Sort

Mova a parte ordenada

Selection Sort

Feito!

Selection Sort

Ache o menor elemento entre as posições 2 e 5

Selection Sort

6 Achado!, 7 não, 9 não, 8 não

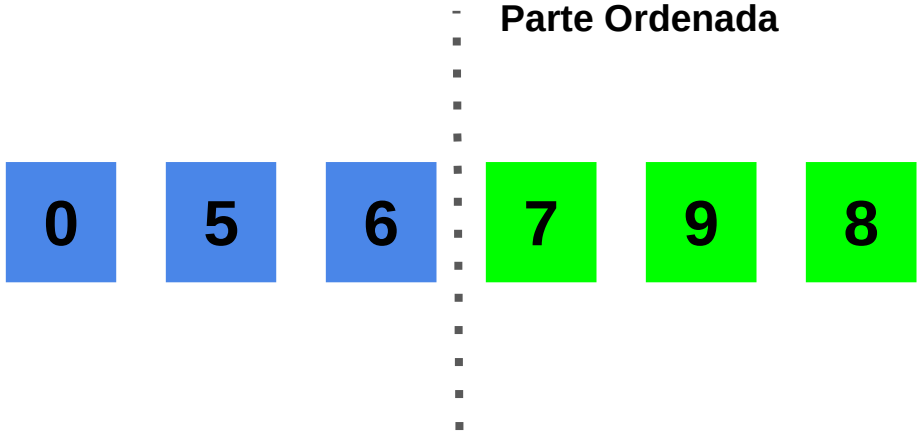
Selection Sort

Coloque-o na posição na frente + 2

Selection Sort

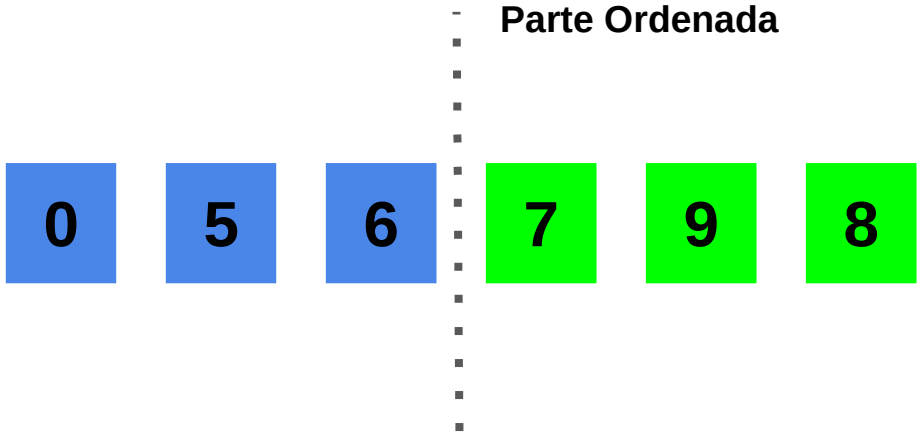
Mova a parte ordenada

Selection Sort



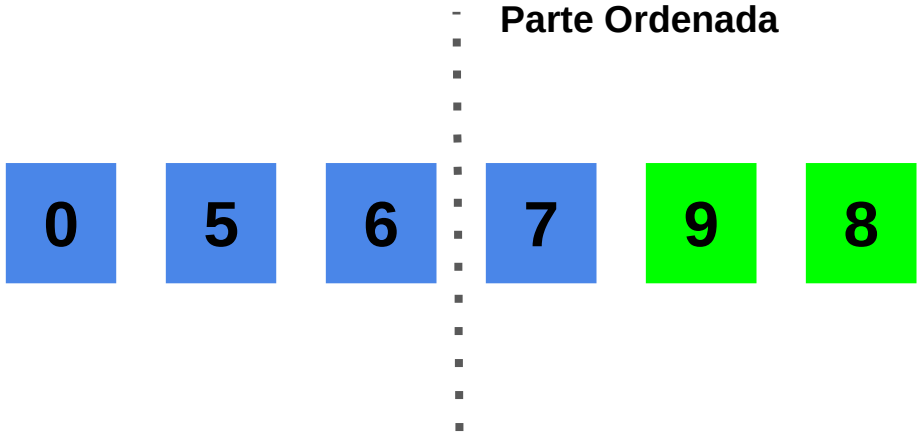
Feito!

Selection Sort



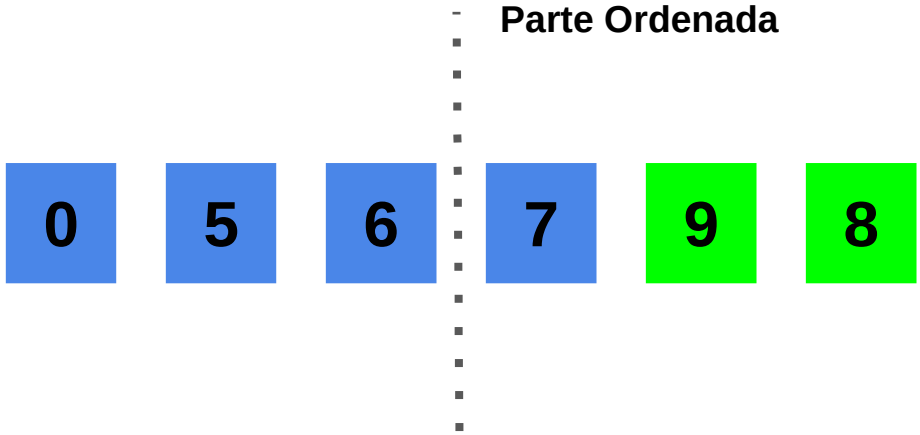
Ache o menor elemento entre as posições 3 e 5

Selection Sort



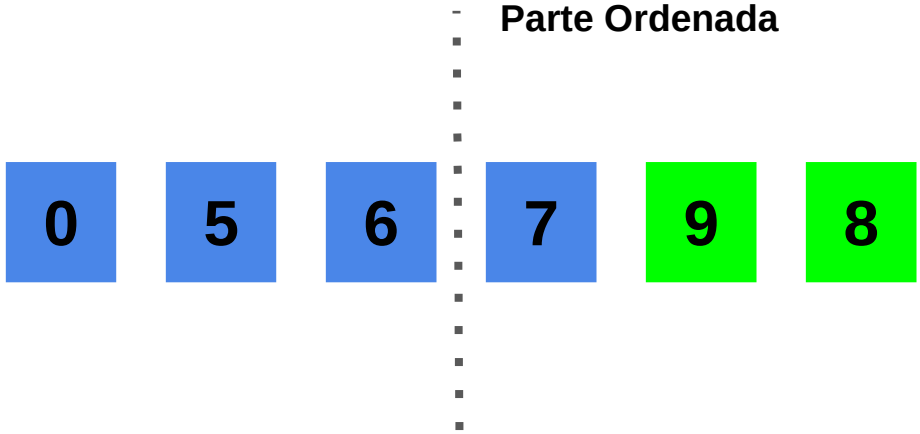
7 Achado!, 9 não, 8 não

Selection Sort



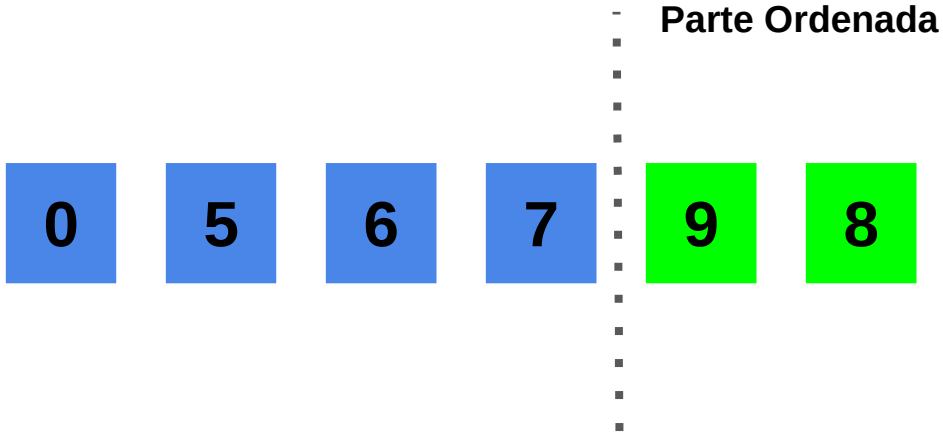
Coloque-o na posição frente +3

Selection Sort



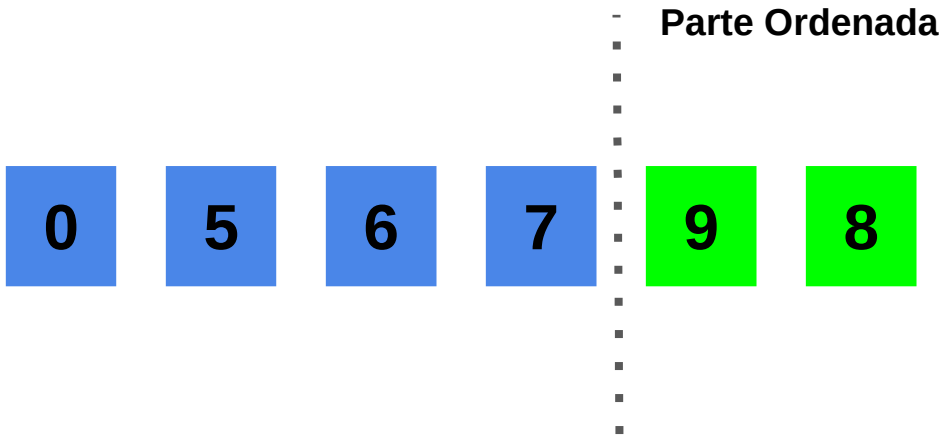
Mova a parte ordenada.

Selection Sort



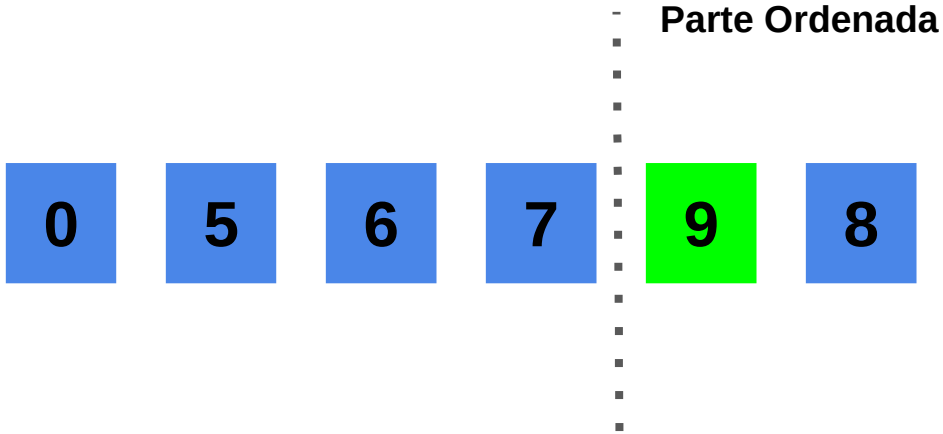
Feito!

Selection Sort



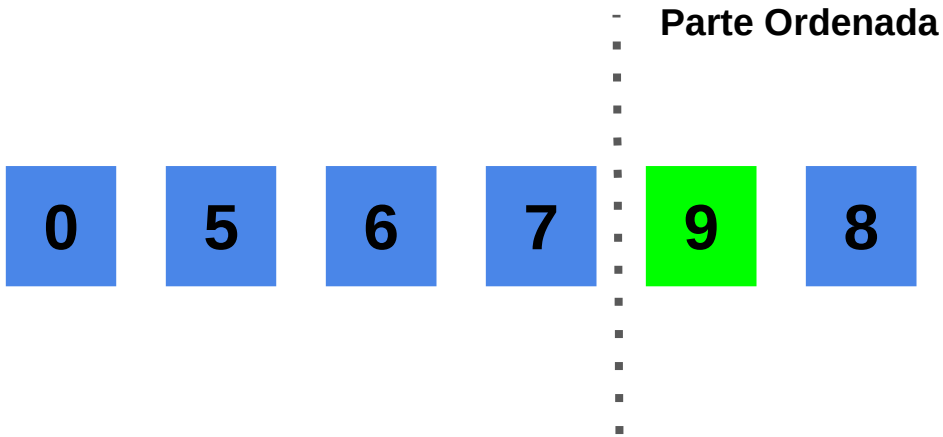
Ache o menor elemento entre as posições 4 e 5

Selection Sort



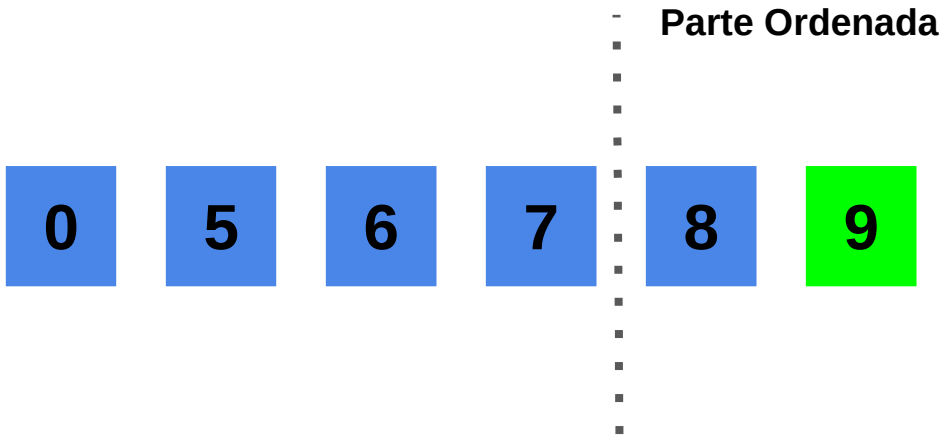
9 não, 8 Achado!

Selection Sort



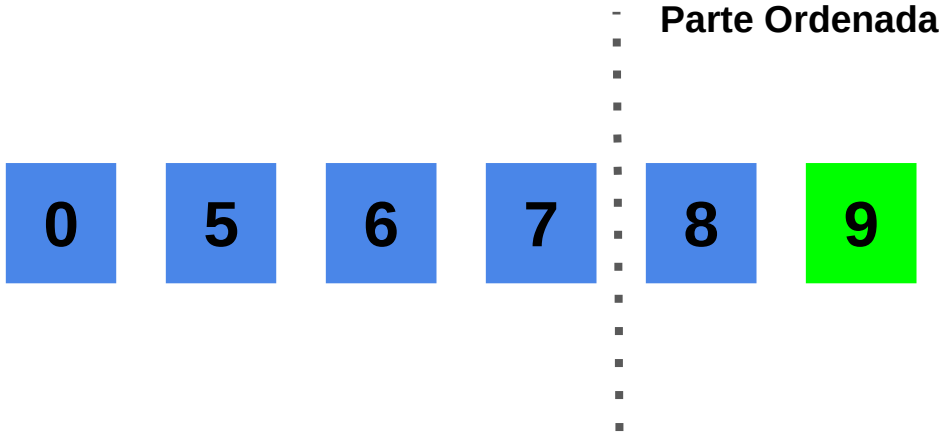
Coloque-o na posição frente + 4

Selection Sort



Feito!

Selection Sort



Mova a parte ordenada

Selection Sort

Feito!

Selection Sort

Ache o menor elemento entre as posições 5 e 5

Selection Sort

9 Achado!

Selection Sort

Coloque-o na posição frente + 5

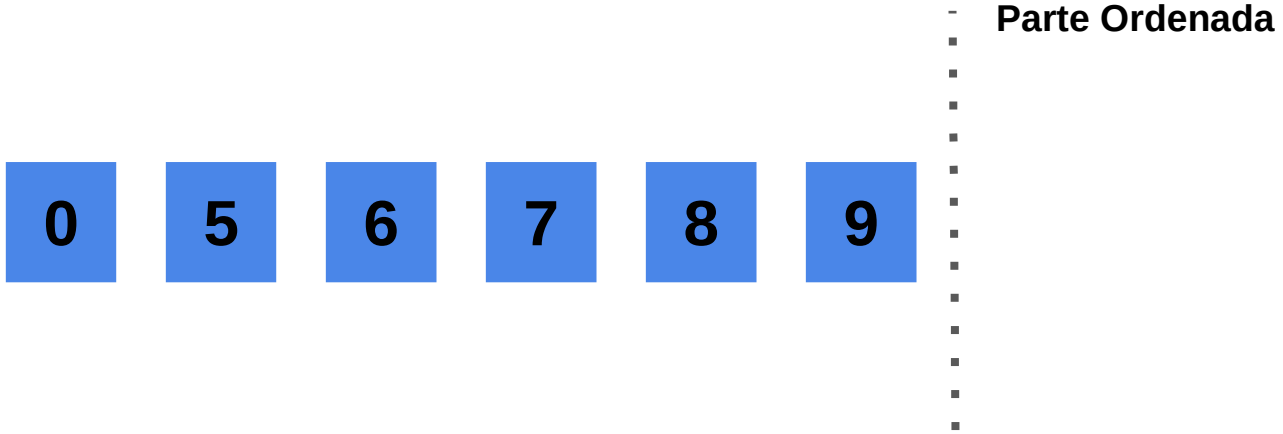
Selection Sort

Feito!

Selection Sort

Mova a parte ordenada

Selection Sort



Feito! e Ordenado!!!

Análise de Complexidade Selection Sort

- No melhor caso: O(n)
- No pior caso e caso médio: $O(n^2)$

Merge Sort

- Nosso primeiro algoritmo **Divide-and-Conquer**;
- Divide o array de entrada em duas partes, chama a si mesmo para as duas partes e depois junta as duas metades já ordenadas;

Merge Sort

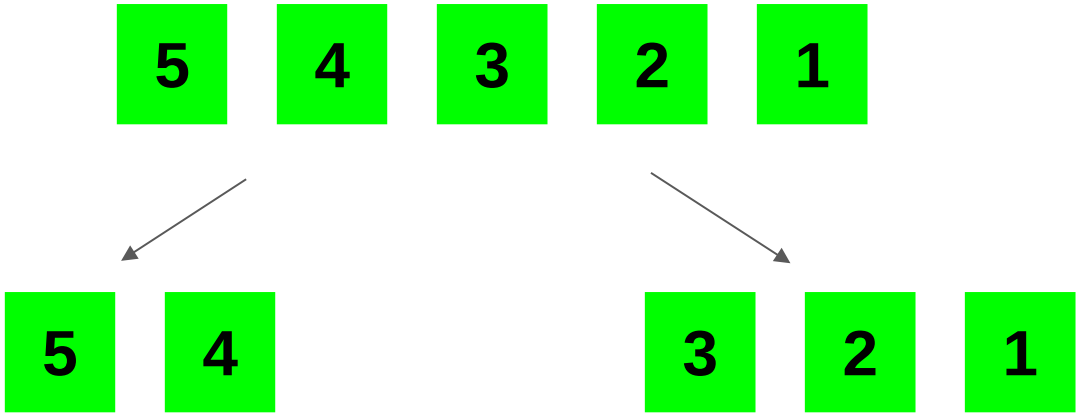
mergeSort(array):

```
    Se len(array) > 1:
        mid = len(array)//2
        E = array[:mid]
        D = array[mid:]
        mergeSort(E)
        mergeSort(D)
        merge(array, E, D)
```

Merge Sort

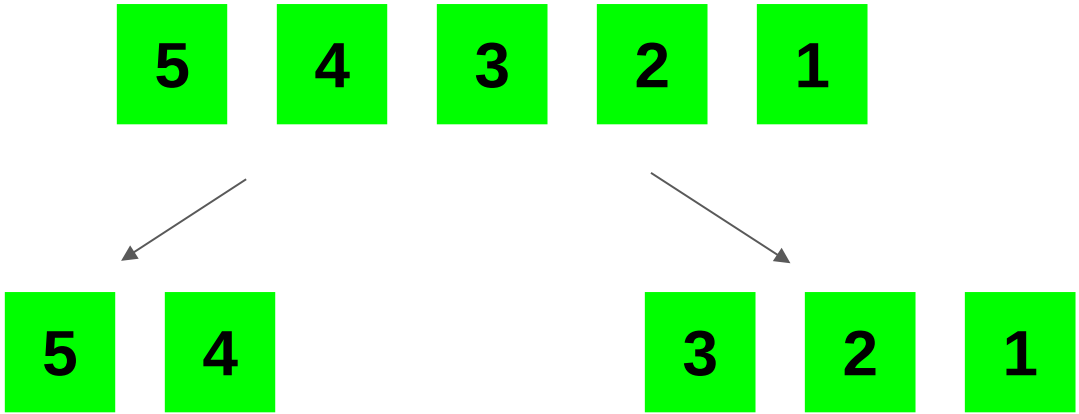
Repartir “ao meio” em dois sub-arrays E e D

Merge Sort



Feito!

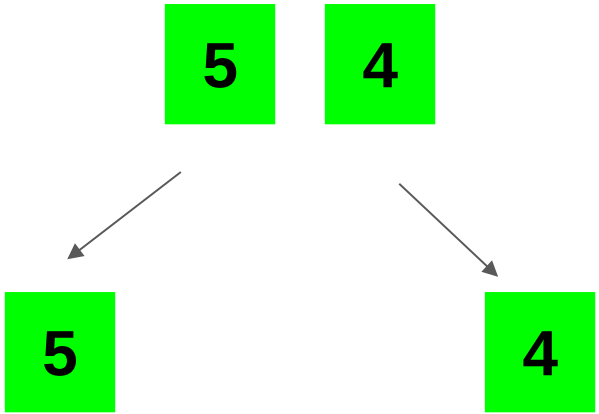
Merge Sort



Aplicar as chamadas recursivas

`mergeSort(E)`

Merge Sort



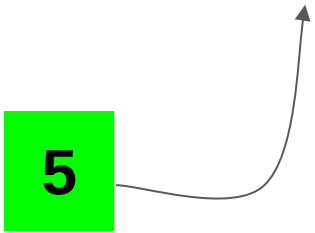
Aplicar as chamadas recursivas

`mergeSort(E)`

Merge Sort

Opa! Chegamos no caso base! `len(array) > 1`

Merge Sort

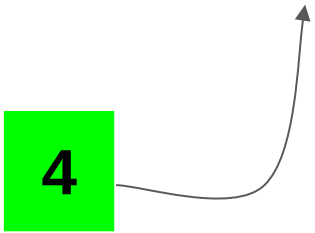


A função vai retornar...

Merge Sort

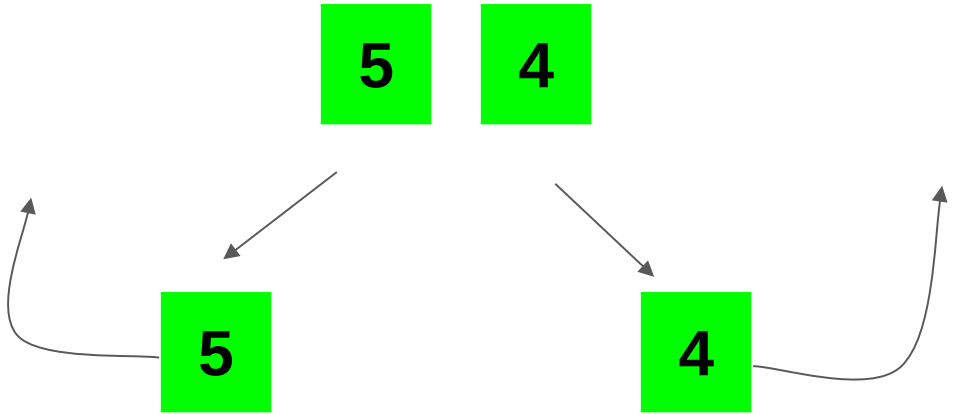
Opa! Chegamos no caso base! `len(array) > 1`

Merge Sort



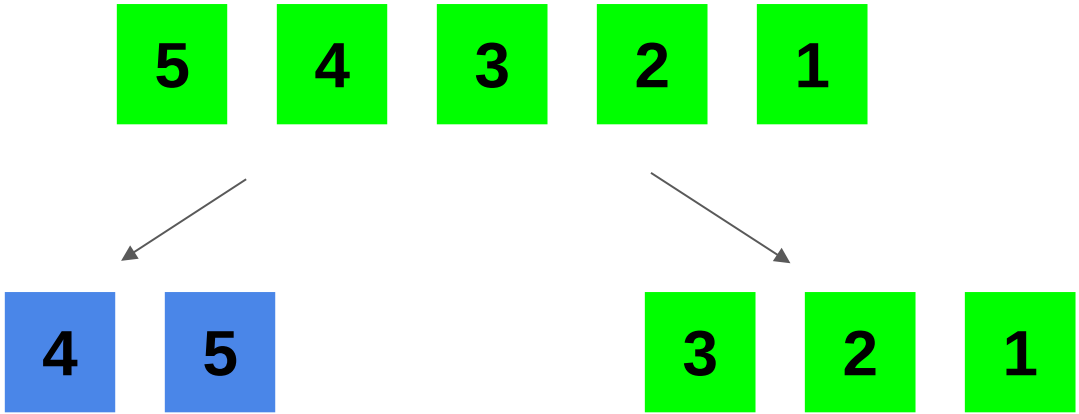
A função vai retornar...

Merge Sort



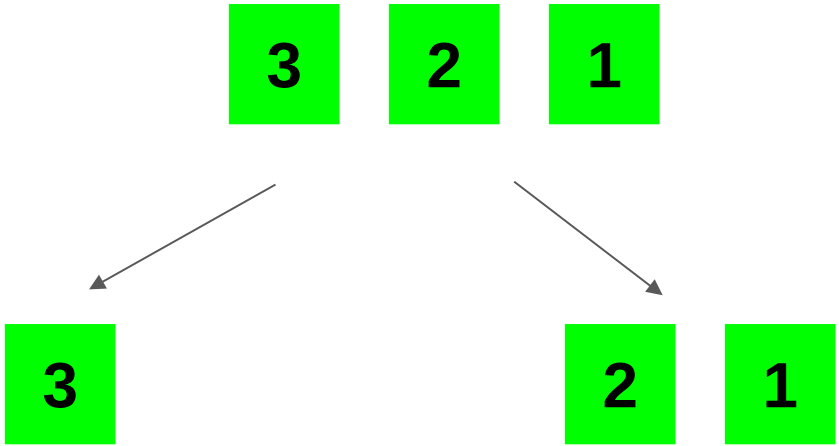
Hora de fazer o Merge (array, E, D)

Merge Sort



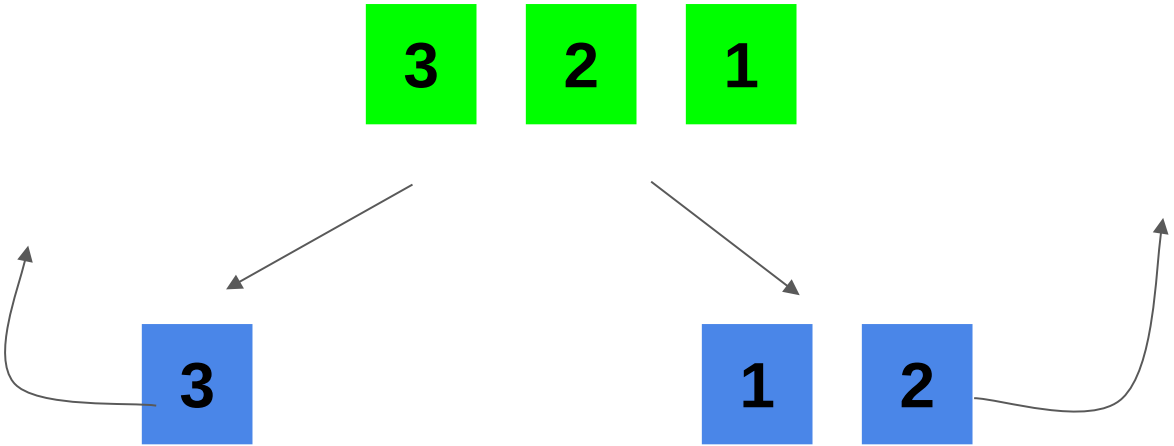
Feito!, agora aplicar as chamadas recursivas
`mergeSort(D)`

Merge Sort



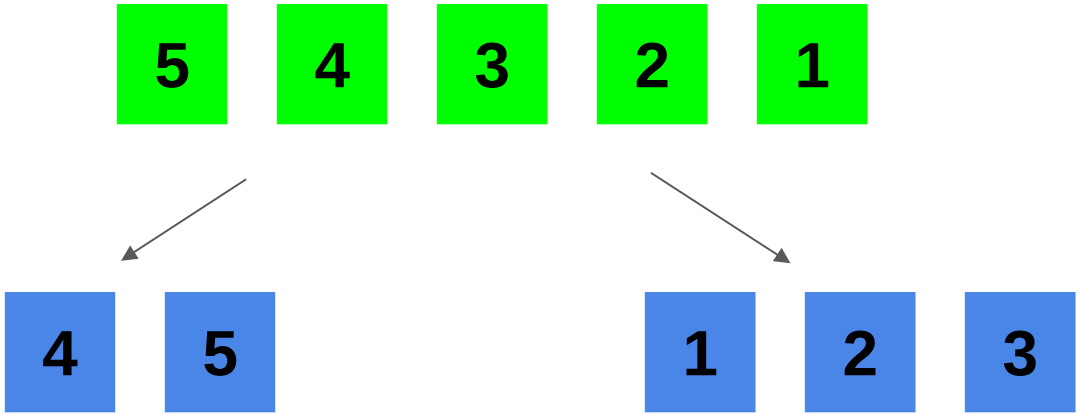
Seguindo o mesmo padrão anterior...

Merge Sort



Seguindo o mesmo padrão anterior...

Merge Sort



Ficamos desta forma e chamamos o Merge(array, E, D) final.

Merge Sort

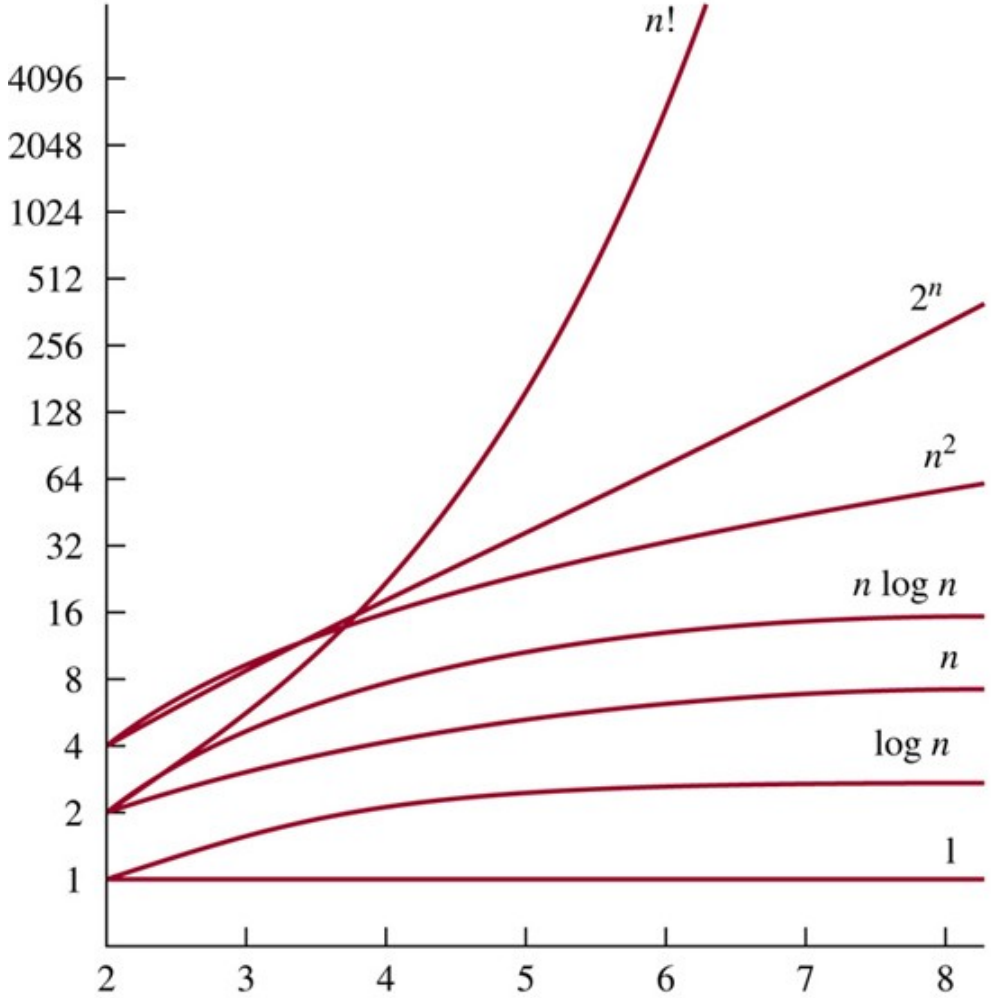
Ordenado!

Merge Sort

- Complexidade:
  - Merge Sort tem a mesma complexidade para o caso médio, o melhor caso e o pior caso: **O(nLogn)**

Merge Sort

- **Eixo Horizontal:**
  - **Tamanho do problema**
- **Eixo Vertical:**
  - **Número de Instruções**




Thank You!