Métodos de Ordenação

Ordenação

Ordenação de vetores:

- entrada: vetor com os elementos a serem ordenados
- Saída: mesmo vetor com elementos na ordem especificada
- Ordenação:
  - Pode ser aplicada a qualquer dado com ordem bem definida
  - Vetor com dados complexos

Bubble Sort

Ordenação bolha:

- processo básico:
  - quando dois elementos estão fora de ordem, troque-os de posição até que o i-ésimo elemento de maior valor do vetor seja levado para as posições finais do vetor
- continue o processo até que todo o vetor esteja ordenado

Maior elemento

v0	4	2	5	1
v1	2	4	5	1
v2	2	4	5	1
v3	2	4	1	5
	0	1	2	3

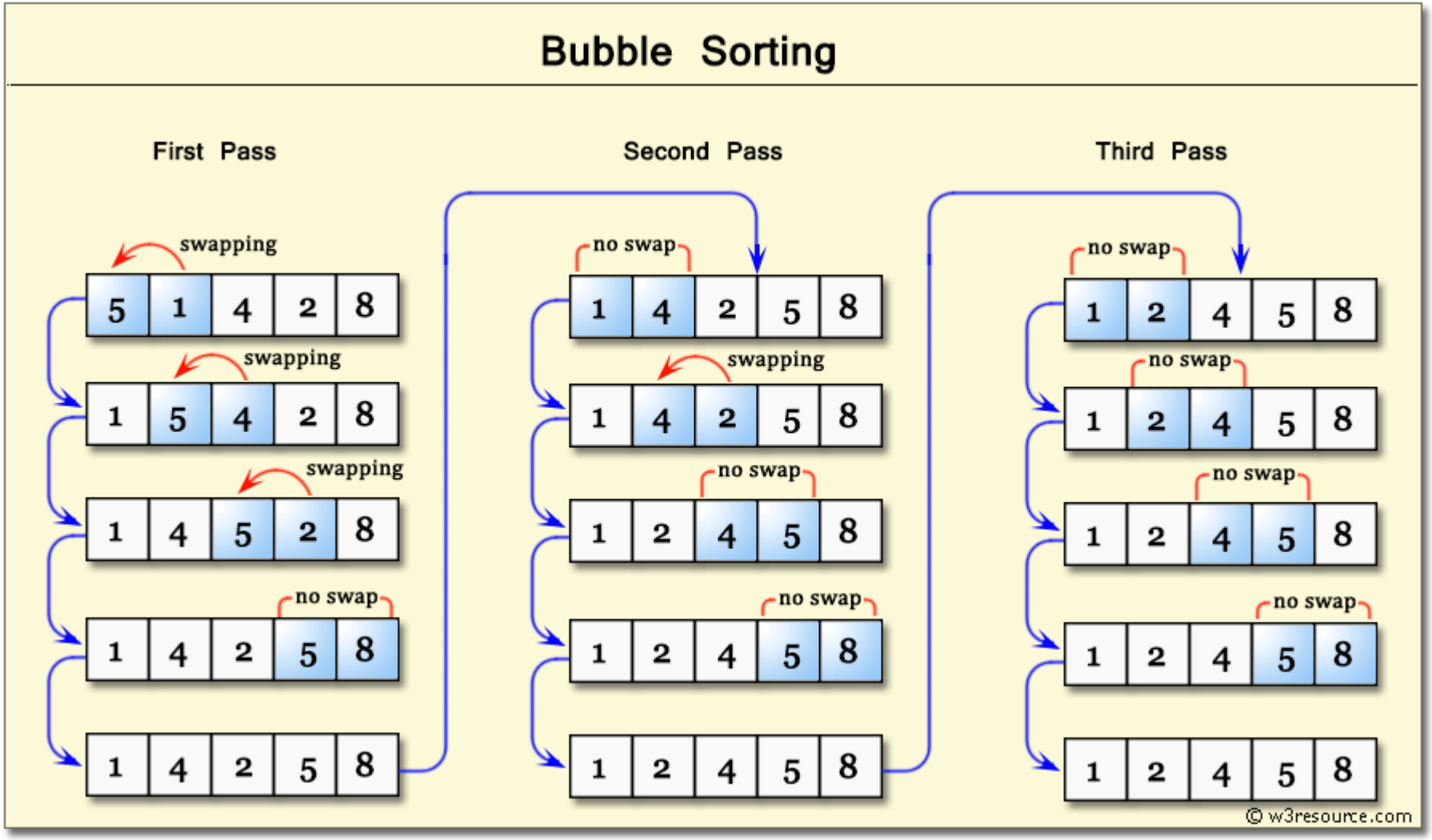
2º maior elemento

v4	2	4	1	5
v5	2	4	1	5
	2	1	4	5
	0	1	2	3

3º maior elemento

v6	2	1	4	5
	1	2	4	5
	0	1	2	3

Ordenação - bolha



Ordenação - Bolha

25 48 37 12 57 86 33 92	*25x48*
25 48 37 12 57 86 33 92	*48x37 troca*
25 37 48 12 57 86 33 92	*48x12 troca*
25 37 12 48 57 86 33 92	*48x57*
25 37 12 48 57 86 33 92	*57x86*
25 37 12 48 57 86 33 92	*86x33 troca*
25 37 12 48 57 33 86 92	*86x92*
25 37 12 48 57 33 86 92	final da primeira passada

o maior elemento, 92, já está na sua posição final

Ordenação - Bolha

```
25 37 12 48 57 33 86 92    25x37
25 37 12 48 57 33 86 92    37x12 troca
25 12 37 48 57 33 86 92    37x48
25 12 37 48 57 33 86 92    48x57
25 12 37 48 57 33 86 92    57x33 troca
25 12 37 48 33 57 86 92    57x86
25 12 37 48 33 57 86 92    final da segunda passada
```

o segundo maior elemento, 86, já está na sua posição final

```
25 12 37 48 33 57 86 92    25x12 troca
12 25 37 48 33 57 86 92    25x37
12 25 37 48 33 57 86 92    37x48
12 25 37 48 33 57 86 92    48x33 troca
12 25 37 33 48 57 86 92    48x57
12 25 37 33 48 57 86 92    final da terceira passada
```

Idem para 57.

```
12 25 37 33 48 57 86 92    12x25
12 25 37 33 48 57 86 92    25x37
12 25 37 33 48 57 86 92    37x33 troca
12 25 33 37 48 57 86 92    37x48
12 25 33 37 48 57 86 92    final da quarta passada
```

Idem para 48.

```
12 25 33 37 48 57 86 92    12x25
12 25 33 37 48 57 86 92    25x33
12 25 33 37 48 57 86 92    33x37
12 25 33 37 48 57 86 92    final da quinta passada
```

Idem para 37.

```
12 25 33 37 48 57 86 92    12x25
12 25 33 37 48 57 86 92    25x33
12 25 33 37 48 57 86 92    final da sexta passada
```

Idem para 33.

```
12 25 33 37 48 57 86 92    12x25
12 25 33 37 48 57 86 92    final da sétima passada
```

Idem para 25 e, conseqüentemente, 12.

```
12 25 33 37 48 57 86 92    final da ordenação
```

Insertion Sort

A cada iteração, o algoritmo remove um elemento dos dados de entrada, encontra o local ao qual pertence na lista e o insere ali. Ele se repete até que nenhum elemento de entrada permaneça.

É de implementação simples.

Eficiente para conjunto de dados pequenos

Mais eficiente na prática do que a maioria dos outros algoritmosde complexidades quadráticas (O (N2)), como Selection Sort e o Bubble Sort.

Adaptável, pois varia de acordo com os dados de entrada.

Insertion Sort

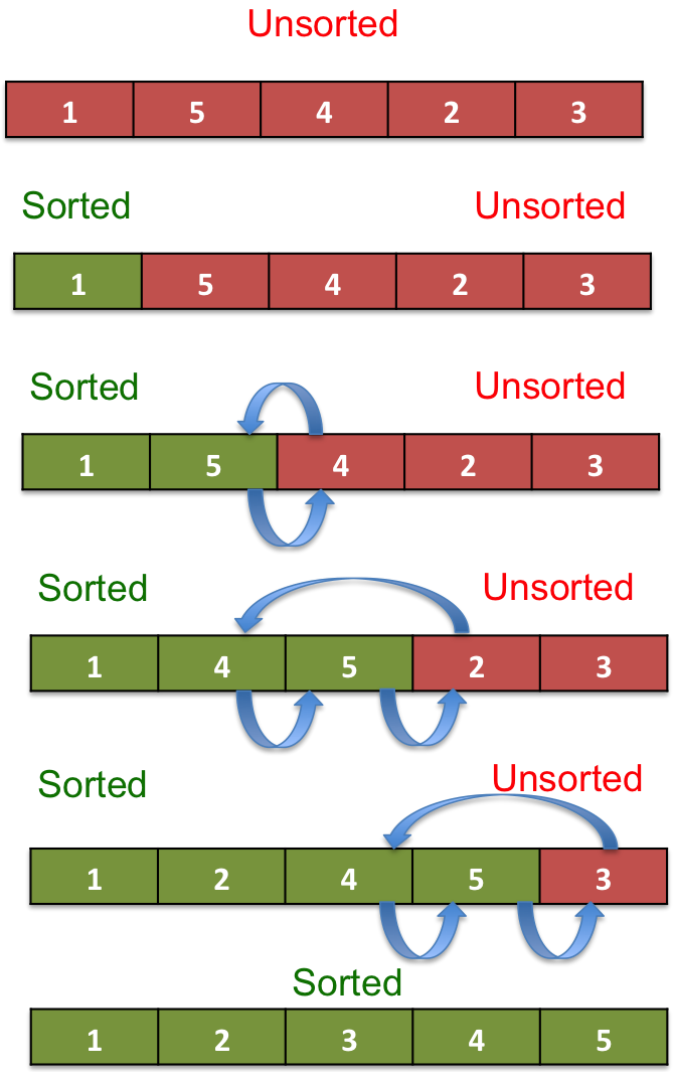
Mas e se eu já tenho um vetor e quero usar o Insertion Sort?

Então:

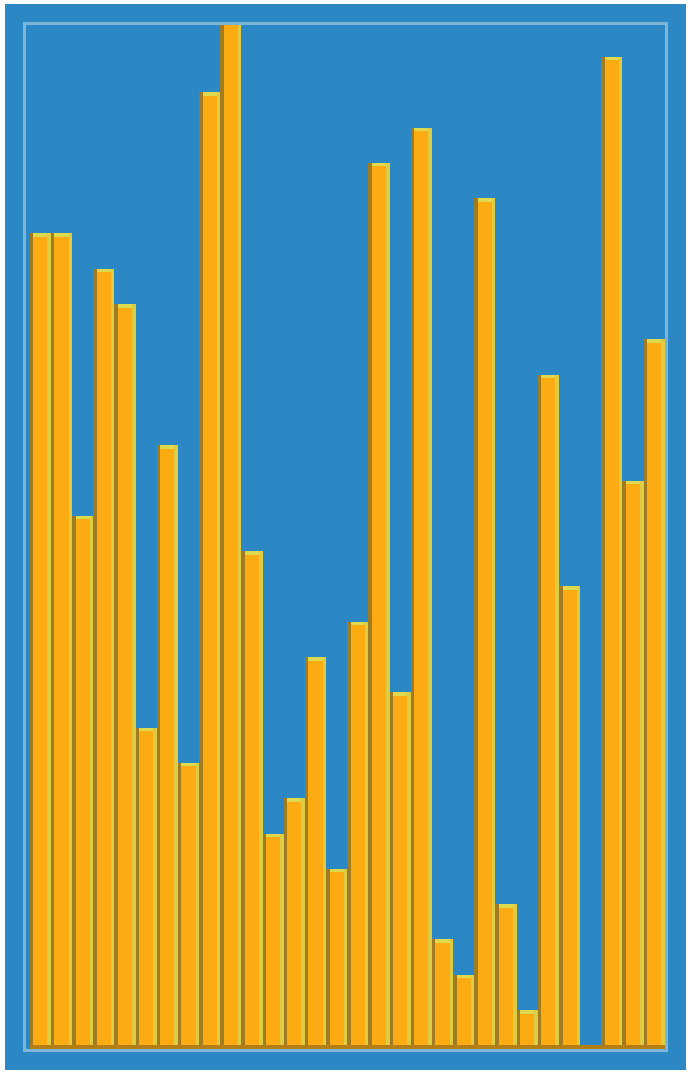
- Meu primeiro elemento será a parte ordenada
- O resto do vetor será a minha parte não-ordenada.
- Então vou comparando 1/1 dos elementos.

- https://www.youtube.com/watch?v=OGzPmgsI-pQ

- https://www.youtube.com/watch?v=8oJS1BMKE64



Insertion Sort



Insertion Sort

Varredura	X[0]	X[1]	X[2]	X[3]	X[4]
Vetor original	9	8	7	6	5
1	{8	9}	{7	6	5}
2	{7	8	9}	{6	5}
3	{6	7	8	9}	{5}
4	{5	6	7	8	9}

O que entrar no vetor, vai ser colocado no seu respectivo lugar

Selection Sort

A ordenação por seleção consiste em, cada etapa selecionar o maior (ou o menor) elemento e alocá-lo em sua posição correta dentro da futura lista ordenada.

Durante a aplicação do método de seleção a lista com n registros fica decomposta em duas sub listas, uma contendo os itens já ordenados e a outra com os restantes ainda não ordenados. No início a sub lista ordenada é vazia e a outra contém todos os demais. No final do processo a sub lista ordenada apresentará (n-1) itens e a outra apenas 1.

Selection Sort

As etapas(ou varreduras) consistem em **buscar** o maior elemento (ou o menor) da lista não ordenada e colocá-lo na lista ordenada. Veja no exemplo abaixo o resultado das etapas da ordenação de um vetor de inteiros:

Etapa	X[0]	X[1]	X[2]	X[3]	X[4]
Vetor original	5	9	1	4	3
1	{5	3	1	4}	{9}
2	{4	3	1}	{5	9}
3	{1	3}	{ 4	5	9}
4	{1}	{3	4	5	9}

https://www.youtube.com/watch?v=xWBP4lzkoyM

https://www.youtube.com/watch?v=92BfuxHn2XE

SHELL SORT

Tem complexidade O(N2).

É um aprimoramento do Insertion Sort.

O Insertion Sort compara registros adjascentes. Qual o problema?

Se o menor elemento estiver no final? Ele tem que percorrer todo o vetor, comparando 1 por 1. Já o ShellSort contorna este problema permitindo trocas de registros distantes, ou seja, ele compara o primeiro elemento com o quarto, o segundo com o quinto, o terceiro com o sexto. E assim por diante.

SHELL SORT

O elemento na posição *x* é comparado (e trocado) com o elemento na posição *x+h (h é quantas casas saltou).*

Quando *h = 1*, o algoritmo é equivalente ao algoritmo de inserção.

Vantagens:

- Shellsort é uma ótima opção para arquivos de tamanho moderado
- Sua implementação é simples e requer uma quantidade de código pequena

Desvantagens:

- O tempo de execução do algoritmo é sensível à ordem inicial do arquivo

SHELL SORT

Shell Sort:

1º passo: todos os elementos que estiverem em intervalos de 4 posições entre si na sequência corrente são agrupados e ordenados separadamente. Este processo é chamado de ordenação de distância quatro (h=4).

2º passo: os elementos são agrupados em grupos cujo intervalo é de duas posições, sendo então ordenados novamente. Este processo é chamado ordenação de distância 2 (h=2).

3º passo: todos os elementos são ordenados através de uma ordenação simples de distância 1 (h=1).

SHELL SORT

Exemplo: Shellsort

E	X	E	M	P	L	O
E	X	E	M	P	L	O
E	L	E	M	P	X	O
E	L	E	M	P	X	O

h = 4

E	L	E	M	P	X	O
E	L	E	M	P	X	O
E	L	E	M	P	X	O
E	L	E	M	P	X	O
E	L	E	M	P	X	O
E	L	E	M	O	X	P

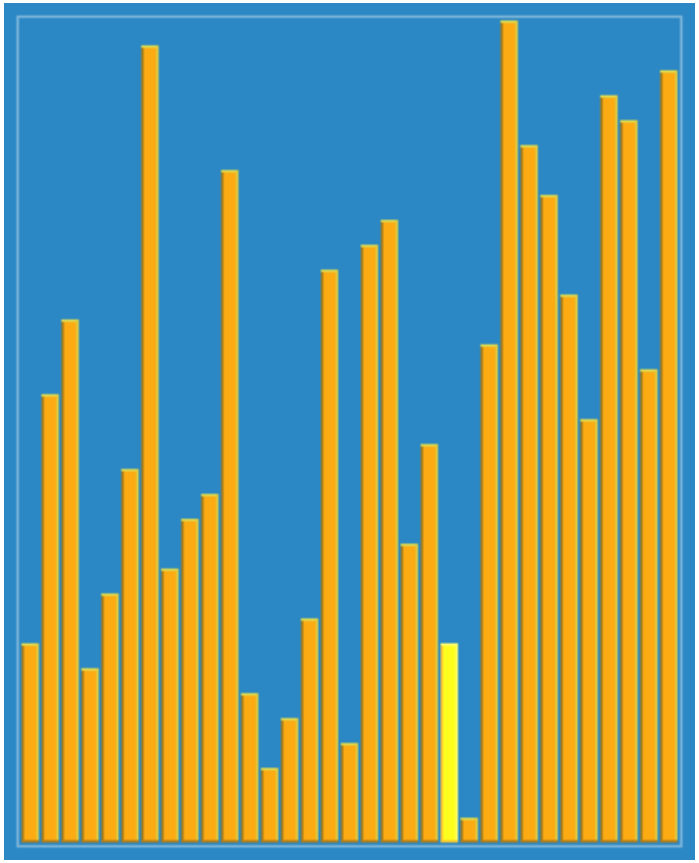
h = 2

h = 1 (inserção)

E	L	E	M	O	X	P
E	L	E	M	O	X	P
E	L	E	M	O	X	P
E	E	L	M	O	X	P
E	E	L	M	O	X	P
E	L	E	M	O	X	P
E	L	E	M	O	X	P
E	L	E	M	O	P	X

Basicamente o algoritmo passa várias vezes pela lista dividindo o grupo maior em grupos menores. Nos grupos menores é aplicado o Insertion Sort.

SHELL SORT



https://www.youtube.com/watch?v=qzXAVXddcPU

QUICKSORT

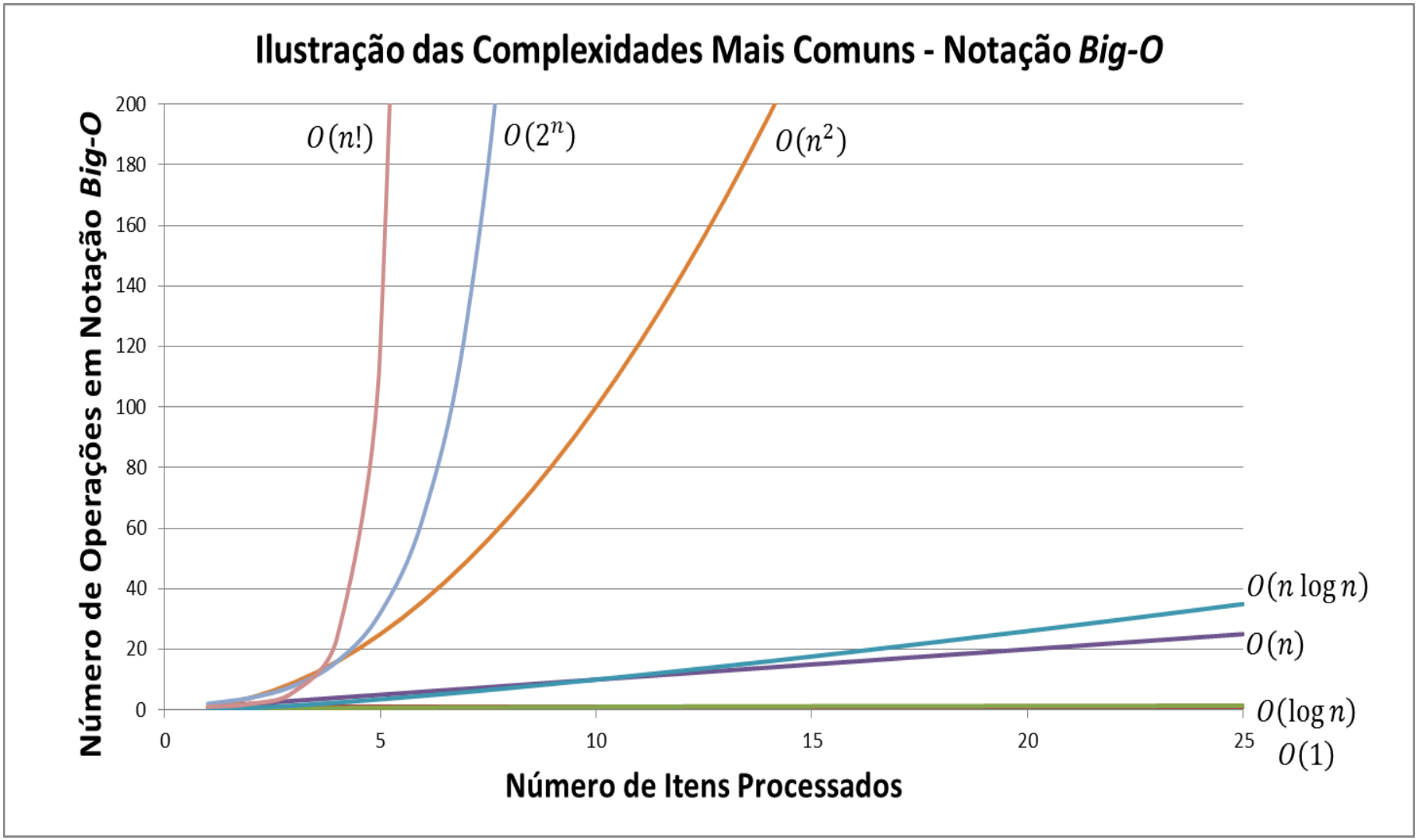


Ilustração das Complexidades Mais Comuns - Notação Big-O
Número de Operações em Notação Big-O
200
180
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
Número de Itens Processados
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)

QUICKSORT

Em algumas raras instâcias, o Quicksort pode ser tão lento quanto os algoritmos elementares; mas em geral é muito mais rápido. Mais precisamente, o algoritmo tem complexidade O(N logN) no caso médio e O(N2) no pior caso.

O núcleo do algoritmo QuickSort é rearranjar um vetor v[n] de modo que todos os elementos pequenos fiquem na parte esquerda do vetor e todos os elementos grandes fiquem na parte direita.

QUICKSORT

1. Iniciar com uma lista L de n itens
2. Escolher um item pivô v, de L
3. Particionar L em duas listas não ordenadas, L1 e L2

   L1: conterá todas as chaves **menores** que v

   L2: conterá todas as chaves **maiores** que v

   Itens com a mesma chave que v podem fazer parte de L1 ou L2

   O pivô v não faz parte de nenhuma das duas listas

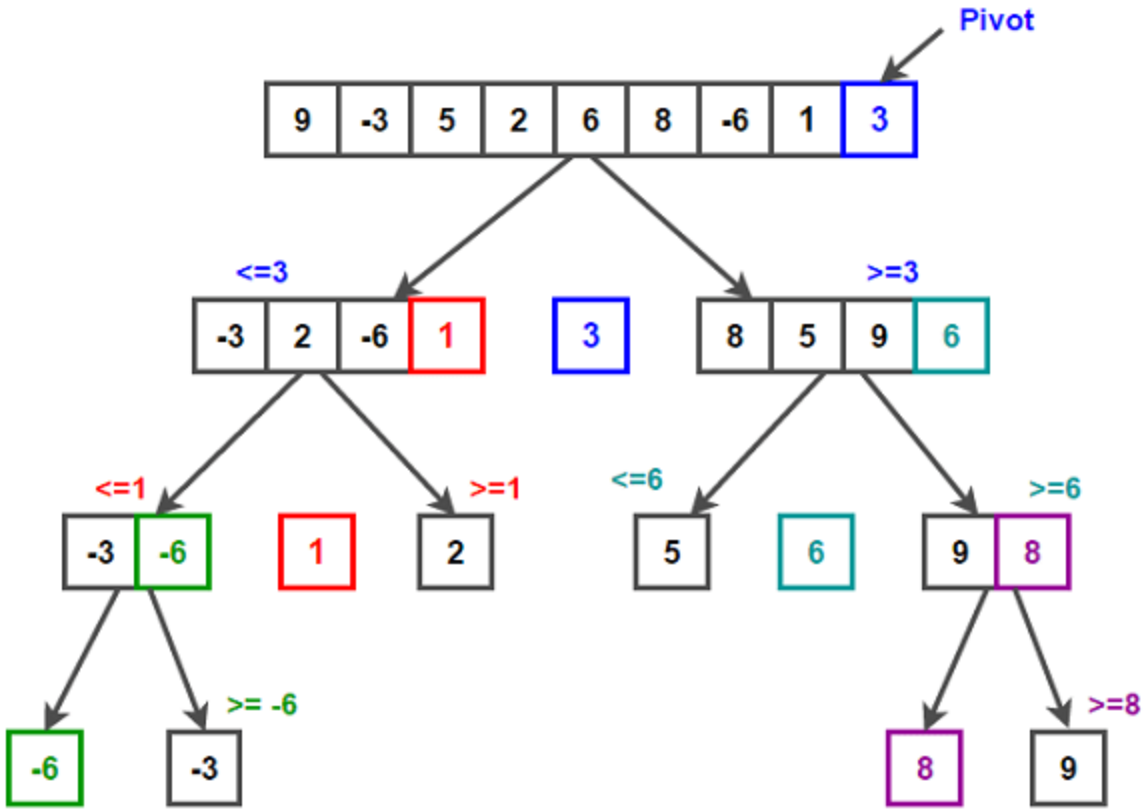
4. Ordenar:

   L1 recursivamente, obtendo a lista ordenada S1

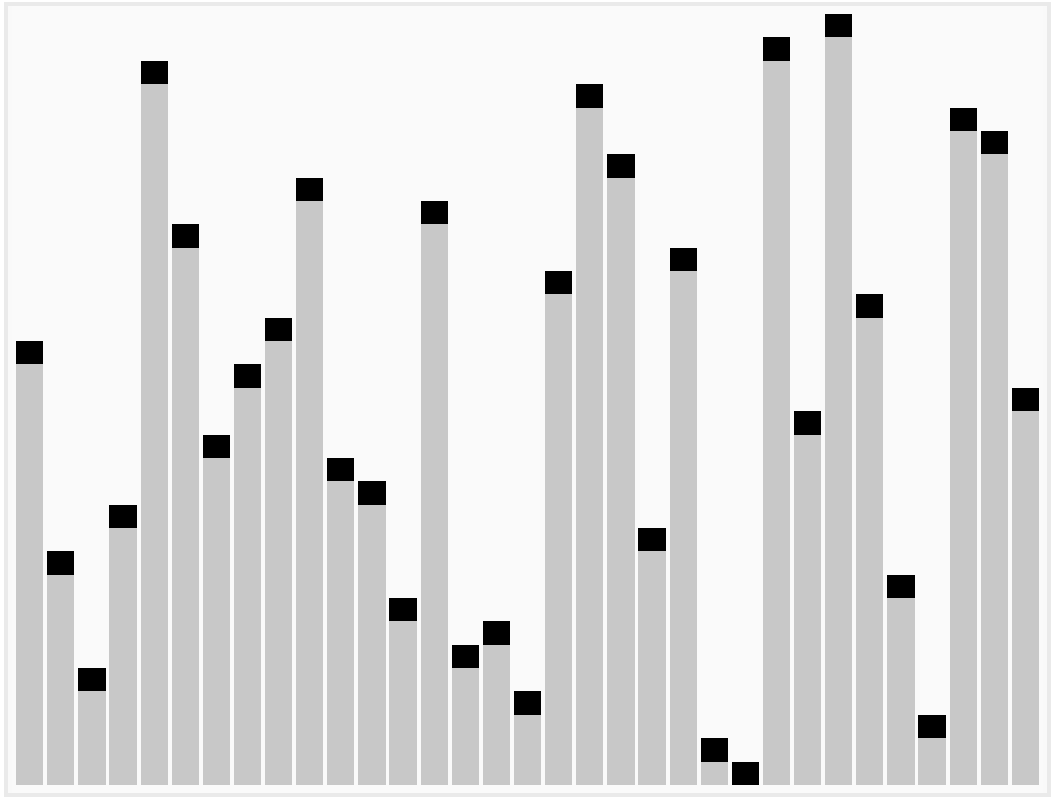
   L2 recursivamente, obtendo a lista ordenada S2

5. Concatenar S1, v, S2 produzindo a lista ordenada S

QUICKSORT



QUICKSORT



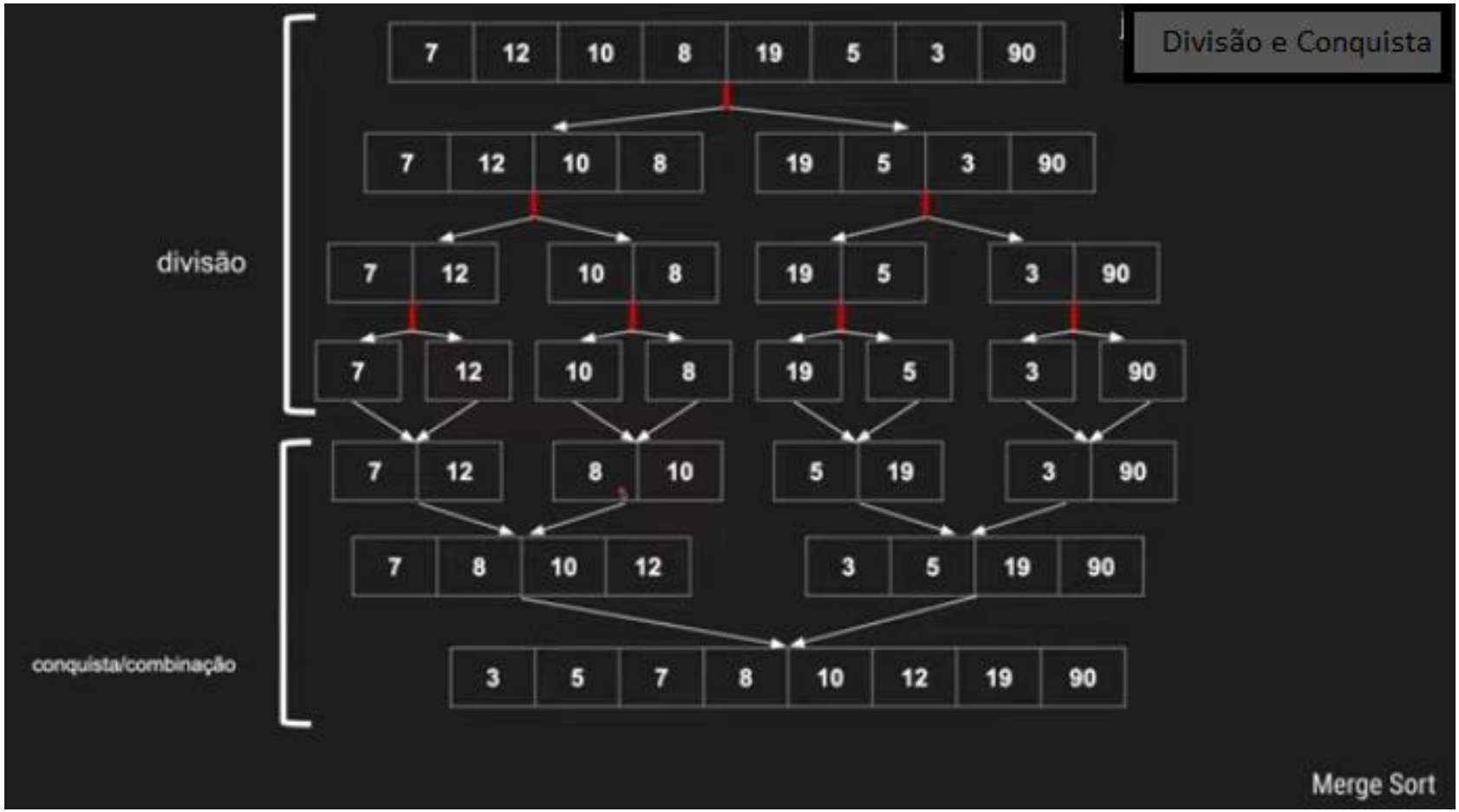
MERGESORT

O Merge Sort é um algoritmo de divisão-e-conquista semelhante ao QuickSort.

No caso do Merge Sort, uma característica importante é que sua eficiência é N logN para o melhor, pior e para o caso médio.

Tanto o QuickSort quanto o MergeSort aplicam várias vezes o particionamento para ordenar um array, porém o MergeSort **não** usa o pivô, mas sim o **divide o vetor em partes iguais** (mesmo sendo um número ímpar de elementos).

MERGESORT


Divisão e Conquista
divisão
conquista/combinação
7 12 10 8 19 5 3 90
7 12 10 8 19 5 3 90
7 12 10 8 19 5 3 90
7 12 10 8 19 5 3 90
7 12 8 10 5 19 3 90
7 8 10 12 3 5 19 90
3 5 7 8 10 12 19 90
Merge Sort

MERGESORT vs QUICKSORT

Exercício de Fixação 1 – Bubble Sort

Ordene o vetor v=20,12,28,05,10,18,04,14,02 usando o método de bolha. Mostre o vetor a cada passo do loop.

Exercício de Fixação 2 – Insertion Sort

Ordene o vetor v=20,12,28,05,10,18,04,14,02 usando o método de inserção. Mostre o vetor a cada passo do loop.

Exercício de Fixação 3 – Selection Sort

Ordene o vetor v=20,12,28,05,10,18,04,14,02 usando o método de seleção. Mostre o vetor a cada passo do loop.

Exercício de Fixação 4 – ShellSort

Ordene o vetor v=20,12,28,05,10,18,04,14,02 usando o método de ShellSort. Mostre o vetor a cada passo do loop.

Exercício de Fixação 5 – Quicksort

Ordene o vetor v=20,12,28,05,10,18,04,14,02 usando o método de Quicksort. Mostre o vetor a cada passo do loop.

Exercício de Fixação 6 – MergeSort

Ordene o vetor v=20,12,28,05,10,18,04,14,02 usando o método de MergeSort. Mostre o vetor a cada passo do loop.