ALGORITMOS

2º Capítulo

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

- *Decrease and conquer.*
- *Algoritmos de ordenação.*

O que é Decrease and Conquer?

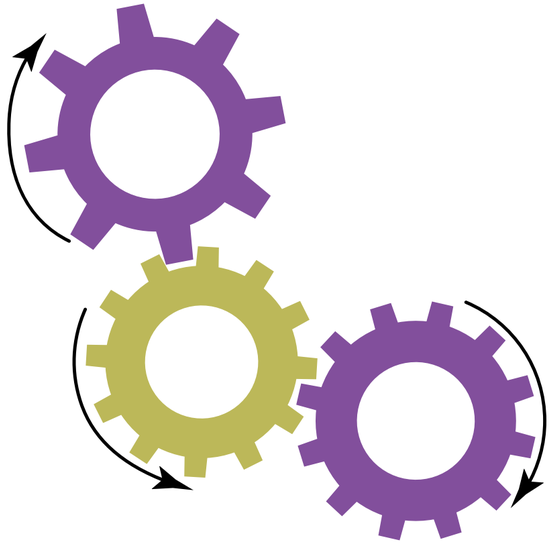
Decrease and Conquer

É uma técnica de algoritmos que tenta **resolver um problema reduzindo-o em um problema menor**, assim, ao solucionar o(s) problema(s) menor(es), também estará resolvendo o problema original.

Essa abordagem também é conhecida como **abordagem incremental ou indutiva.**

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

Processo geral



1. Diminua ou reduza a instância do problema para uma instância menor do mesmo problema;
2. Conquiste o problema resolvendo uma instância menor do problema;
3. Estenda a solução da instância menor para obter a solução do problema original.

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

Implementações

Ao utilizar a técnica é comum utilizar tanto abordagem **top-down (recursão)** ou **bottom-up (iteração).**

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

Padrões

Existem padrões característicos para se realizar esta técnica, ou seja, diminuir um problema e resolvê-lo:

- **Diminuir por uma constante (geralmente por um).**
  - **exemplo:** insertion sort, algorithms for generating permutations and subsets;
- **Diminuir por um fator constante (geralmente pela metade).**
  - **exemplo:** busca binária, exponenciação pela metade.
- **Diminuir por um tamanho variável.**
  - **exemplo:** algoritmo de Euclides, o algoritmo de seleção.

Exponenciação com recursão

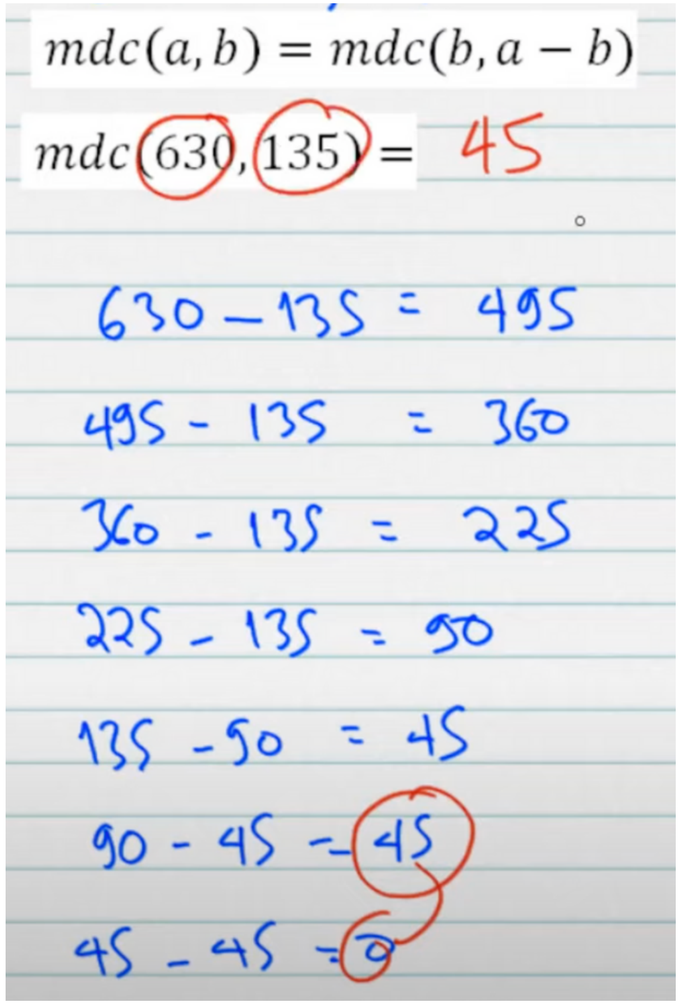
$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even and positive} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd} \\ 1 & \text{if } n = 0 \end{cases}$$

Exponenciação com recursão

colab:
https://colab.research.google.com/drive/1uoU5QuOzYgUxg1Pldy_u9_DP9xf16dQl?usp=sharing

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

Algoritmo de Euclides



https://youtu.be/rTZsui_ftAQ

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

Algoritmos de ordenação

Selection-sort

Ordenação por SELEÇÃO

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

Ideia principal: analisar um array e **selecionar o menor ou maior elemento**, trocando-o de index com o elemento não classificado mais à esquerda do array, repetindo esse processo até que todo o array seja classificado e, consequentemente, ordenado.

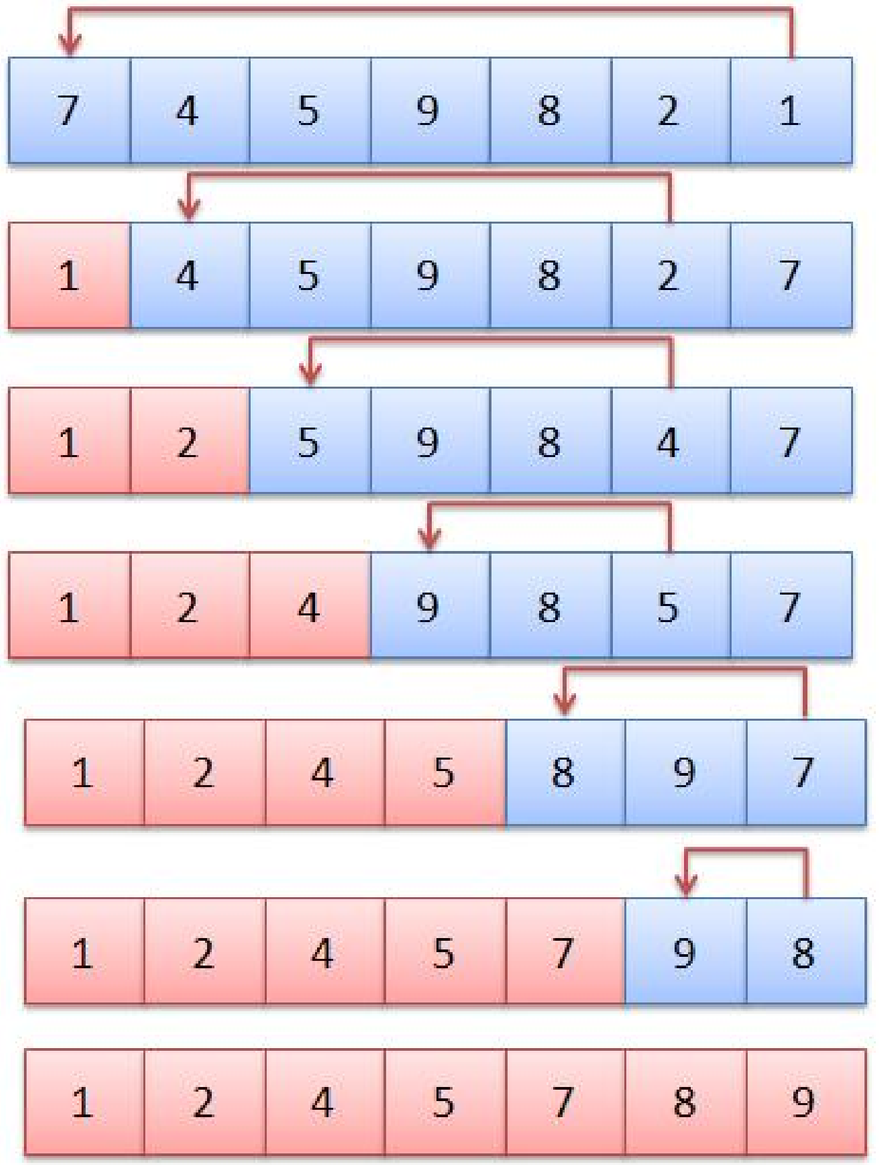
Desvantagens: é ineficiente em listas grandes e geralmente tem um desempenho pior do que a ordenação por inserção.

Vantagens: esse algoritmo é conhecido por sua simplicidade - eficiente em listas pequenas.

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

PROCESSO:

- Encontrar o menor (ou maior) elemento;
- Trocar ele de lugar com o elemento ainda não classificado **MAIS A ESQUERDA**;
- Repetir os itens acima até que **TODOS** os elementos sejam classificados.



Fonte imagem: https://stackoverflow.com/questions/36700830/selection-sort-algorithm

EXEMPLO:

Fazer um BLAST da sequência do receptor humano 5-hidroxitriptamina 2A, ordenando pelo escore usando os primeiros 14 valores para fazer o teste do algoritmo

https://blast.ncbi.nlm.nih.gov/Blast.cgi#sort_mark

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

>sp|P28223.2|5HT2A_HUMAN

MDILCEENTSLSSTTNSLMQLNDDTRLYSNDFNSGEANTSDAFNWTVDSENRTNLSCEGCLSPSCLSLLH
LQEKNWSALLTAVVIILTIAGNILVIMAVSLEKKLQNATNYFLMSLAIADMLLGFLVMPVSMLTILYGYR
WPLPSKLCAVWIYLDVLFSTASIMHLCAISLDRYVAIQNPIHHSRFNSRTKAFLKIIAVWTISVGISMPI
PVFGLQDDSKVFKEGSCLLADDNFVLIGSFVSFFIPLTIMVITYFLTIKSLQKEATLCVSDLGTRAKLAS
FSFLPQSSLSSEKLFQRSIHREPGSYTGRRTMQSISNEQKACKVLGIVFFLFVVMWCPFFITNIMAVICK
ESCNEDVIGALLNVFVWIGYLSSAVNPLVYTLFNKTYRSAFSRYIQCQYKENKKPLQLILVNTIPALAYK
SSQLQMGQKKNSKQDAKTTDNDCSMVALGKQHSEEASKDNSDGVNEKVSCV

COMPLEXIDADE DE TEMPO:

O algoritmo é baseado na seleção do mínimo valor em cada loop e a troca deste de lugar, para cada elemento no array são realizadas x comparações.

Exemplo:

[4,2,1,5]. Uma lista de **n=4**

- No 1º loop são feitas 3 comparações para saber qual o menor elemento **(n-1)**
- No 2º loop são feitas 2 comparações **(n-2)**
- No 3º loop é feita 1 comparação **(n-3)**
- No último loop não é feito comparação.

$$(n-1)+(n-2)+\ldots+1=\sum_{i=1}^{n-1} i$$

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

FÓRMULA PROGRESSÃO ARITMÉTICA:

$$\frac{n(a_1 + a_n)}{2}$$

SOMATÓRIO DO Nº DE COMPARAÇÕES:

$$(n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} i$$

RESULTADO:

$$\frac{(n-1)+1}{2}(n-1) = \frac{1}{2}n(n-1) = \frac{1}{2}(n^2 - n)$$

Logo, a complexidade é **O(n^2)** com relação ao nº de comparações.

Atenção: cada loop realizado faz um troca, são feitas **n-1** trocas (no pior caso).

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

COMPLEXIDADE DE TEMPO E ESPAÇO:

classe	Algoritmo de ordenação
estrutura de dados	Array, Listas ligadas
complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n^2)$
complexidade de espaços pior caso	$O(n)$ total, $O(1)$ auxiliar

Fonte imagem: https://pt.wikipedia.org/wiki/Selection_sort

COMPLEXIDADE DE TEMPO (PIOR CASO):

O grande: Pior caso

[3, 7, 4, 2, 9, 5]

```
1  def selection_sort(lista):
2      n = len(lista) # 1 passo
3      for j in range(n-1): # (n-1) passos
4          min_index = j # 1 passo
5          for i in range(j,n):
6              if lista[i] < lista[min_index]:
7                  min_index = i
8          if lista[j] > lista[min_index]: # 1 passo
9              aux = lista[j] # 1 passo
10             lista[j] = lista[min_index] # 1 passo
11             lista[min_index] = aux # 1 passo
```

Tem mais peso para o O grande

Big-O: Do pior caso → Entrar no if mais externo

Linha 2: 1 + Linha 3: $(n-1)*$ Linhas 4,8,9,10,11 $[5 + X] = \boxed{1 + 5(n-1) + X(n-1)}$

Linhas 5,6,7 (for mais interno)

Exemplo:

[3, 7, 4, 2, 9, 5] ⇒ Procurar pelo elemento menor → for mais interno

[2, 7, 4, 3, 9, 5]

Só olho para essa parte → for mais externo

Ordenação: for mais interno

5 (soma) $(n-1) + (n-2) + (n-3) + \ldots + 2$ ⇒ P.A. de razão -1

1º termo, último termo, segundo termo

$$\frac{(n+2)(n-1)}{2} = \frac{n^2+n-2}{2}$$

Último termo

$$1 + 5(n-1) + X(n-1) = 1 + 5(n-1) + \frac{n^2+n-2}{2} \Rightarrow \boxed{O(n^2)}$$

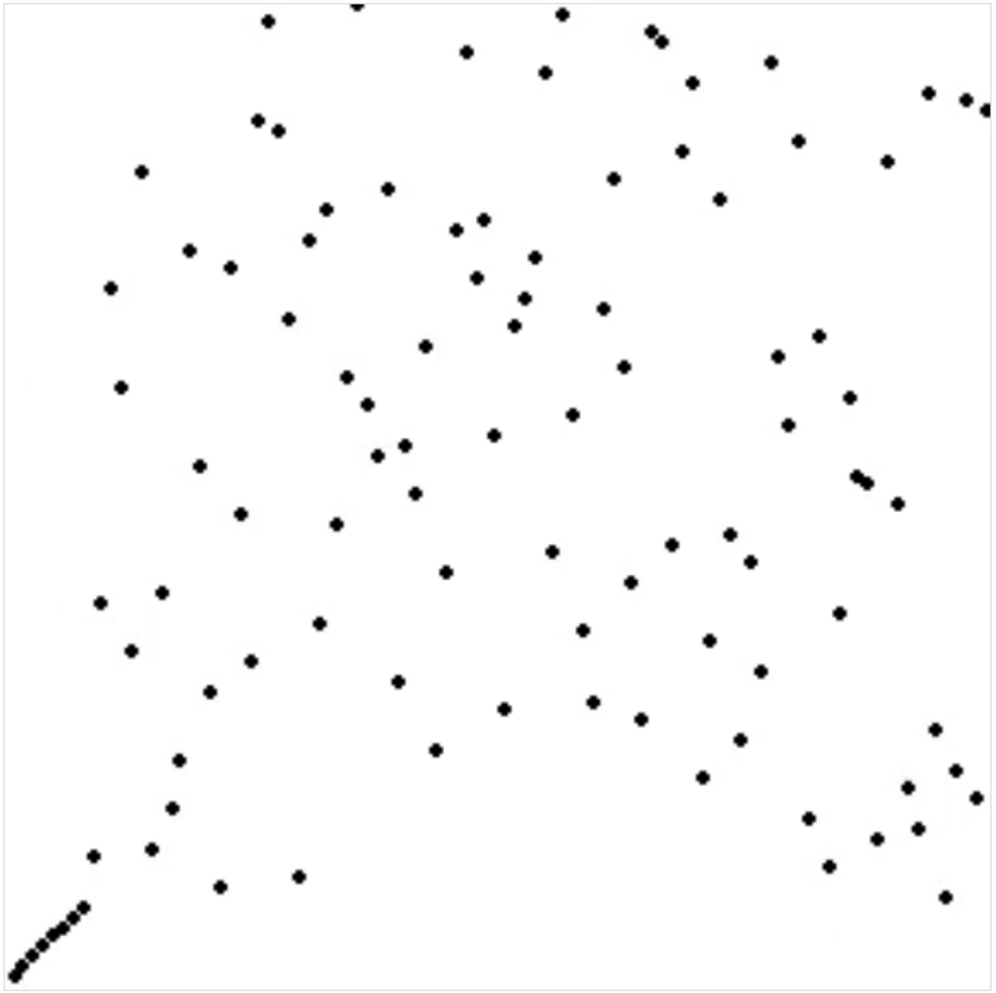
Grupo ②: Dependem de 'n'

$(1 - 5 - 1) + 5n + \frac{n}{2} + \frac{n^2}{2}$ → Termo mais importante para listas grandes

Grupo ①: Constantes

Grupo ③: Depende de n^2

8
5
2
6
9
3
1
4
0
7



1. https://en.wikipedia.org/wiki/Selection_sort#/media/File:Selection-Sort-Animation.gif
2. https://upload.wikimedia.org/wikipedia/commons/b/b0/Selection_sort_animation.gif

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

Bubble-sort

Ordenação por FLUTUAÇÃO

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

Ideia principal: A ideia é **percorrer o vector diversas vezes**, e a cada passagem fazer flutuar para o topo o maior elemento da sequência.

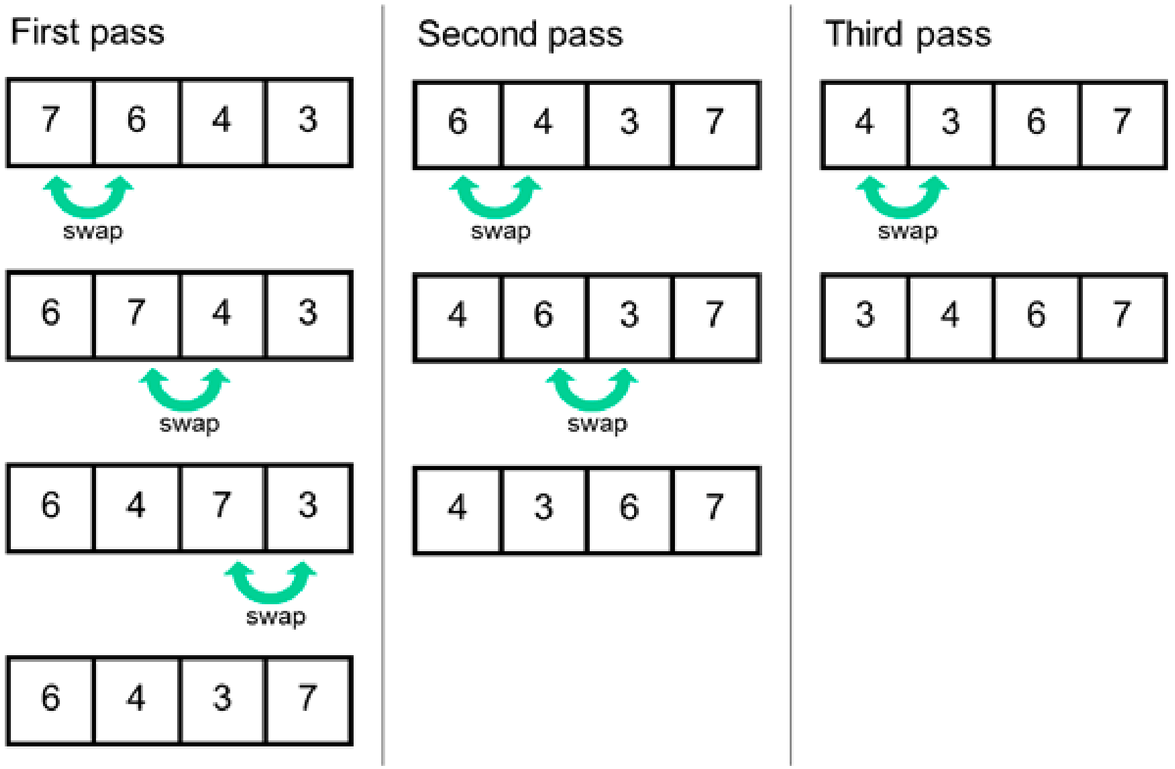
Desvantagens: é ineficiente em listas grandes, não é muito prático.

Vantagens: é bem simples de ser entendido e, no seu melhor caso, apresenta complexidade menor do que no pior caso.

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

PROCESSO:

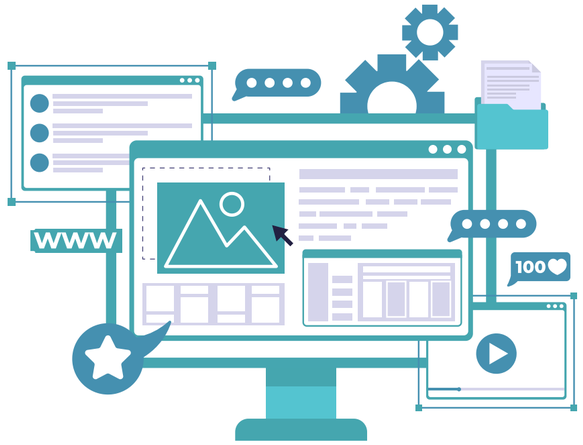
- Seleciona um elemento (da esquerda para a direita) e o compara com cada item do array até que ele **NÃO** seja maior que algum;
- Toda vez que ele for maior que algum elemento, há troca de posições entre estes.
- Esse processo é realizado até que o array seja ordenado.



https://medium.com/austins-software-engineering-journey/bubble-sort-b2f0d63e38f7

CÓDIGO

https://colab.research.google.com/drive/1TTsAnFzKTeN1l5DJqBibSBrBdxjgIImu?usp=sharing



Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

COMPLEXIDADE DE TEMPO:

No pior caso, é necessário percorrer todo o array diversas vezes, todavia, **no melhor caso será necessário percorrer o array apenas 1 vez.**

Exemplo do pior caso:

[4, 3, 2, 1]. Uma lista de **n=4**

- Para que o 4 chegue ao final da lista, são feitas 3 comparações **(n-1)**
- Para organizar o 3, são feitas 2 comparações **(n-2)**
- Para organizar o 2 é feita 1 comparação **(n-3)**
- Por fim, já teríamos o array ordenado

$$(n-1)+(n-2)+\ldots+1=\sum_{i=1}^{n-1} i$$

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

COMPLEXIDADE DE TEMPO E ESPAÇO:

classe	Algoritmo de ordenação
estrutura de dados	Array, Listas ligadas
complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n)$
complexidade de espaços pior caso	$O(1)$ auxiliar

Fonte imagem: https://pt.wikipedia.org/wiki/Bubble_sort

COMPLEXIDADE DE TEMPO (PIOR CASO):

BubbleSort:

```
1 def bubble_sort(lista):
2     n=len(lista) # 1 passo
3     for j in range(n-1): #(n-1) passos
4         for i in range(n-1): #(n-1) passos
5             if lista[i]>lista[i+1]: # x passos
6                 lista[i], lista[i+1]=lista[i+1], lista[i] # 1 passo
```

Big-O: Do pior caso
↳ De tempo

Linha 2: Linhas 3 e 4: Linhas 5 e 6:
1 + (n-1)(n-1) * (1+x):

Exemplo:

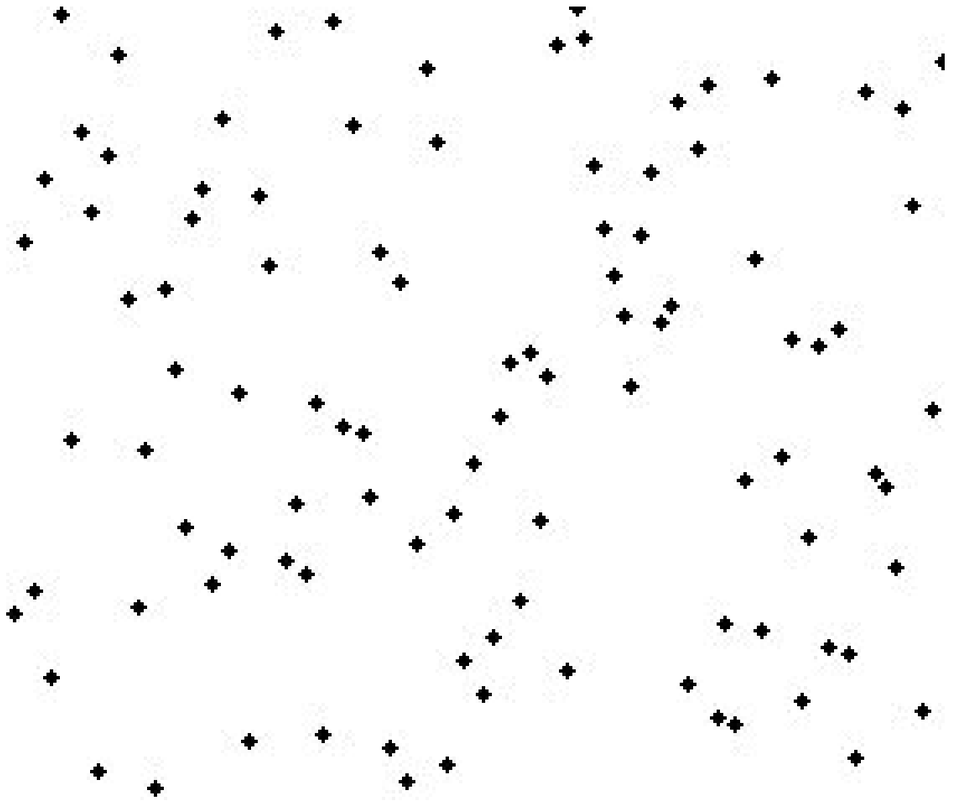
[3, 7, 4, 2, 9, 5]
[3, 4, 7, 2, 9, 5]
[3, 4, 2, 7, 9, 5]
[3, 4, 2, 7, 5, 9]
[3, 2, 4, 7, 5, 9]
[2, 3, 4, 7, 5, 9]
[2, 3, 4, 5, 7, 9]

Big-O = 1 + (n-1)(n-1)
Big-O = 1 + (n^2 - n - n + 1)
Big-O = x + n^2 - 2n + 1

Grupo ① = constantes
Grupo ② = Depende de "n"
Grupo ⑤ = Termo quadrático = Mais peso para big-O

Big-O = n^2 TEMPO

6 5 3 1 8 7 2 4



1. https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif
2. https://pt.wikipedia.org/wiki/Bubble_sort#/media/Ficheiro:Bubble_sort_animation.gif

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

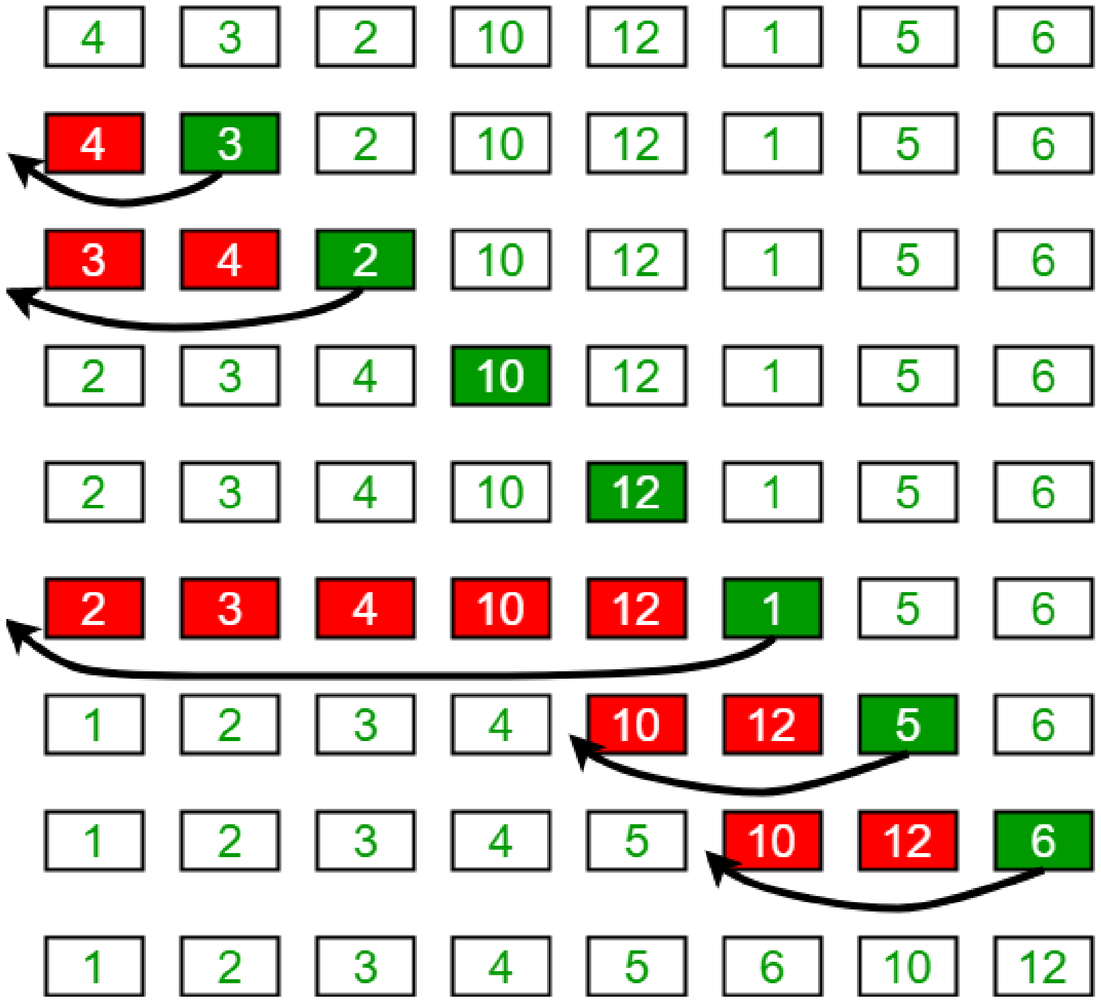
Insertion-sort

Ordenação por INSERÇÃO

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

PROCESSO:

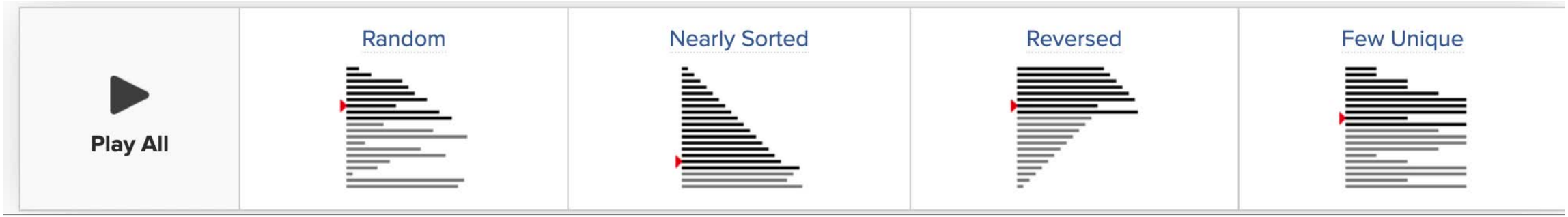
- Inicia-se no segundo elemento do array;
- Compara com todos os elementos à esquerda deste;
- Troca de index quando encontra um elemento **MAIOR** que ele;
- Os outros elementos se mexem para a direita.



VANTAGENS:

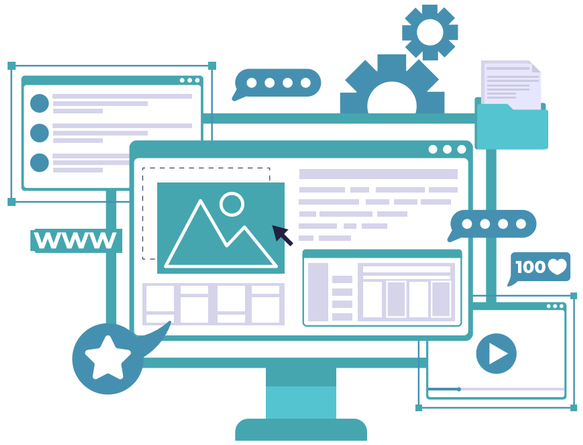
- Estável;
- Baixa sobrecarga;
- Adaptável: Complexidade de espaço O(n) quando quase classificado;
- Bom para resolver problemas pequenos ou quando os dados estão quase ordenados.

https://www.toptal.com/developers/sorting-algorithms/insertion-sort



CÓDIGO

https://colab.research.google.com/drive/1TTsAnFzKTeN1l5DJqBibSBrBdxjgIImu?usp=sharing



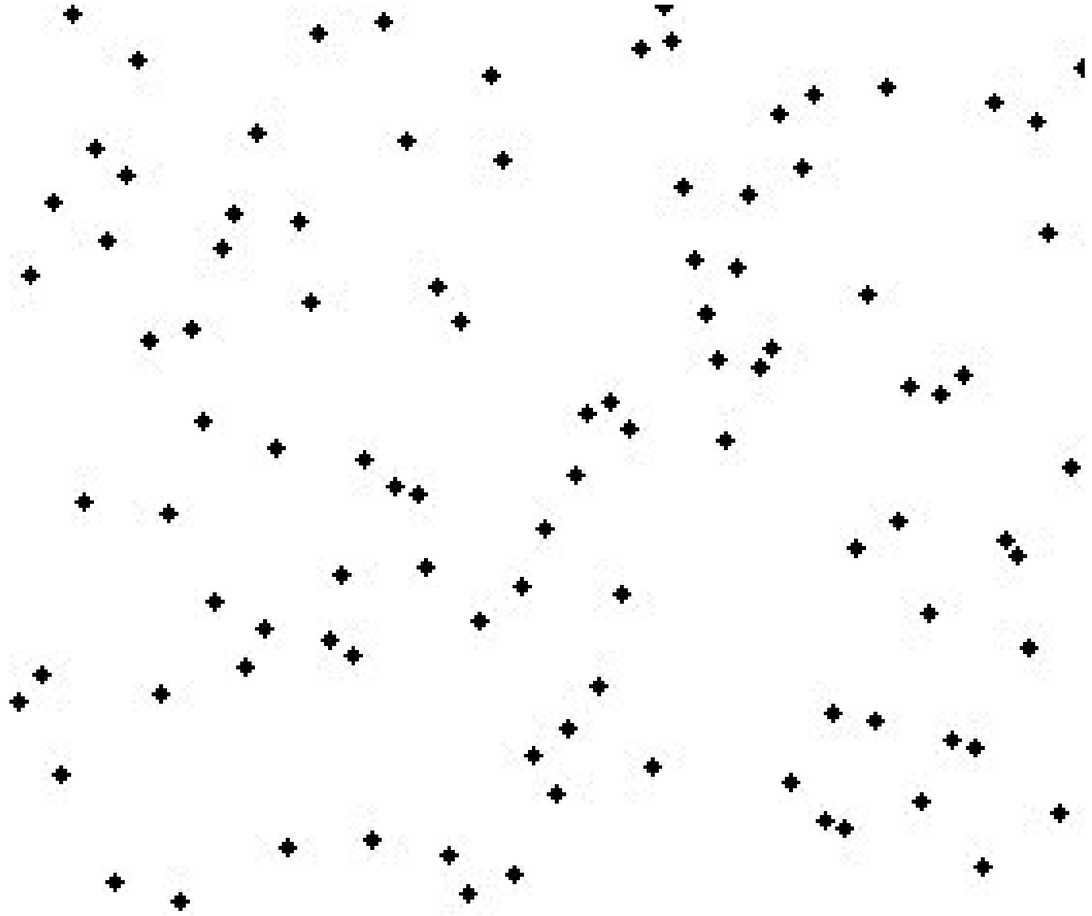
Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

COMPLEXIDADE DE TEMPO E ESPAÇO:

classe	Algoritmo de ordenação
estrutura de dados	Array, Listas ligadas
complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n)$
complexidade de espaços pior caso	$O(n)$ total, $O(1)$ auxiliar
estabilidade	estável

Fonte imagem: https://pt.wikipedia.org/wiki/Insertion_sort

6 5 3 1 8 7 2 4

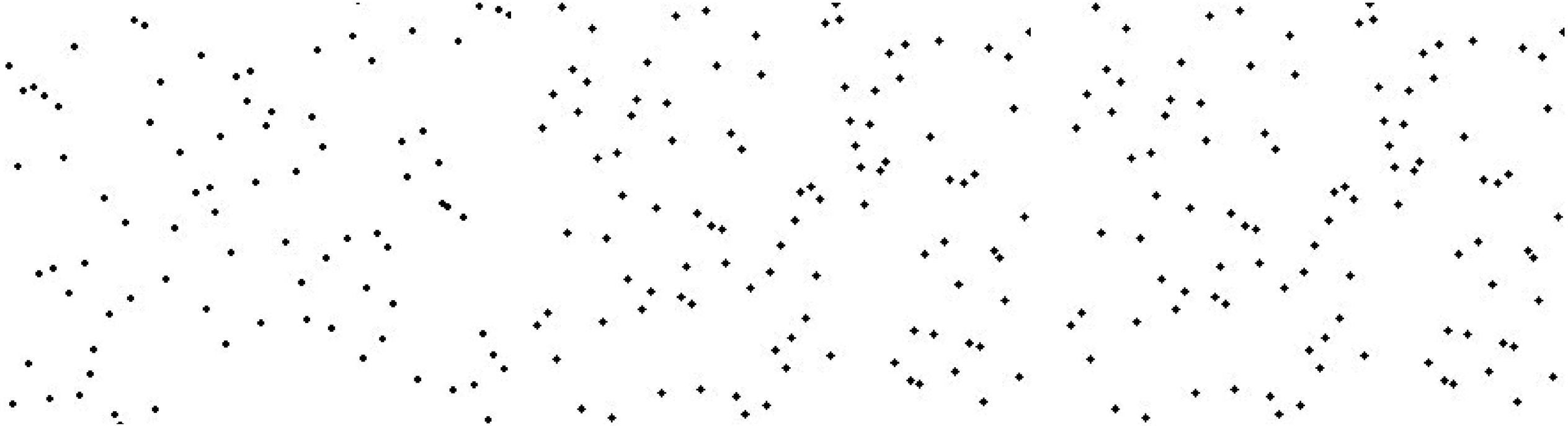


1. https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif
2. https://pt.wikipedia.org/wiki/Bubble_sort#/media/Ficheiro:Bubble_sort_animation.gif

Mulheres em Bioinformática & Data Science LA
Promovendo a colaboração entre mulheres

Comparações

Sorting Algorithm	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	**O(N)**	**$O(N^2)$**	**$O(N^2)$**	**O(1)**
Selection Sort	**$O(N^2)$**	**$O(N^2)$**	**$O(N^2)$**	**O(1)**
Insertion Sort	**O(N)**	**$O(N^2)$**	**$O(N^2)$**	**O(1)**



Selection sort

Bubble sort

Insertion sort

Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

Referências:

https://www.geeksforgeeks.org/decrease-and-conquer/
https://www.youtube.com/watch?v=wAyrtLAeWvI
https://www.youtube.com/playlist?list=PL5TJqBvpXQv4l7nH-08fMfyl7aDFNW_fC
https://youtu.be/ZT_dT8yn48s
https://youtu.be/GiNPe_678Ms
https://youtu.be/S5no2qT8_xg

Questões que surgiram no primeiro encontro:

1. É possível utilizar os algoritmos de ordenação apenas em inteiros? Não, é possível utilizar em floats, strings, etc também, desde que haja uma regra de ordenação de maior e menor.