

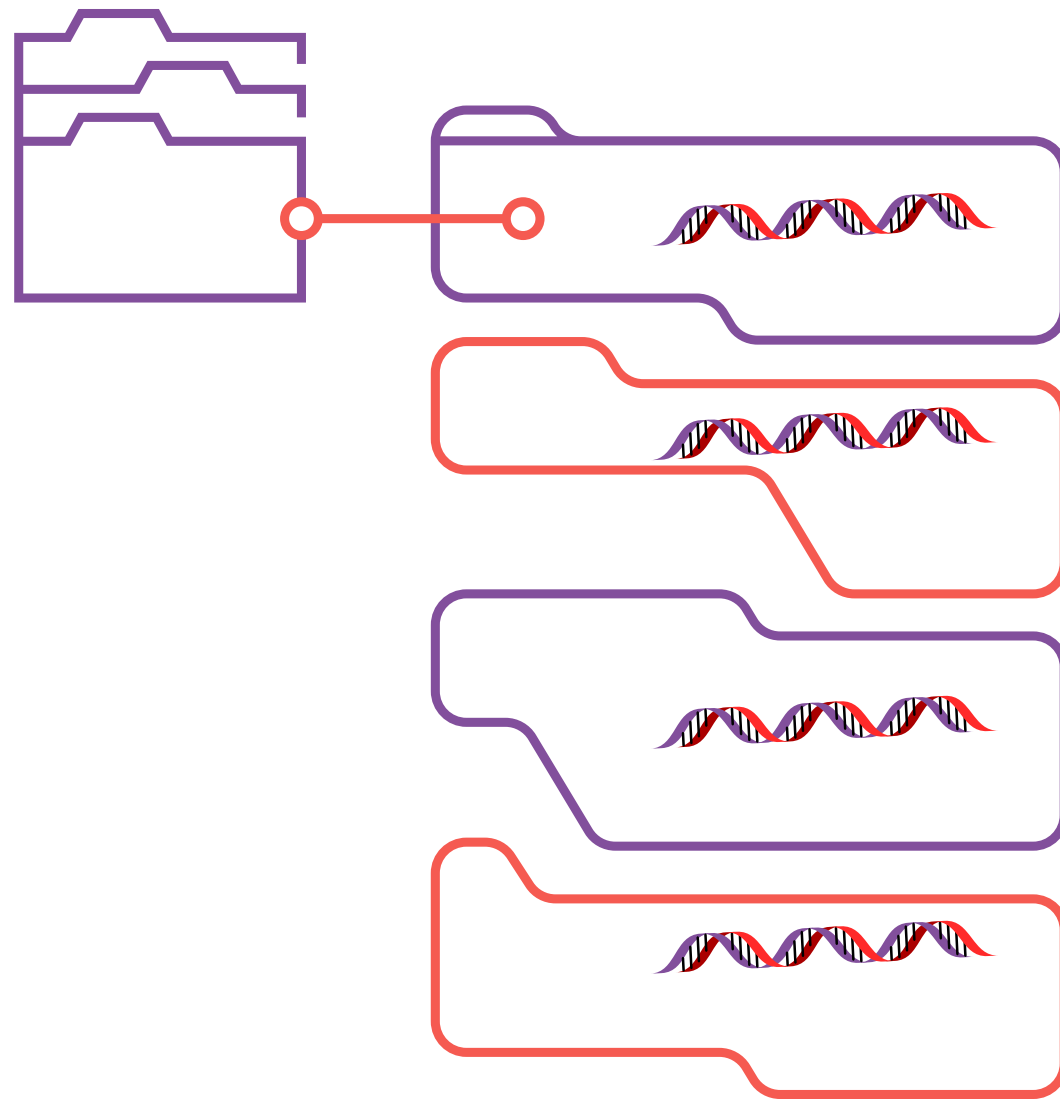
1º ENCONTRO

Algoritmos



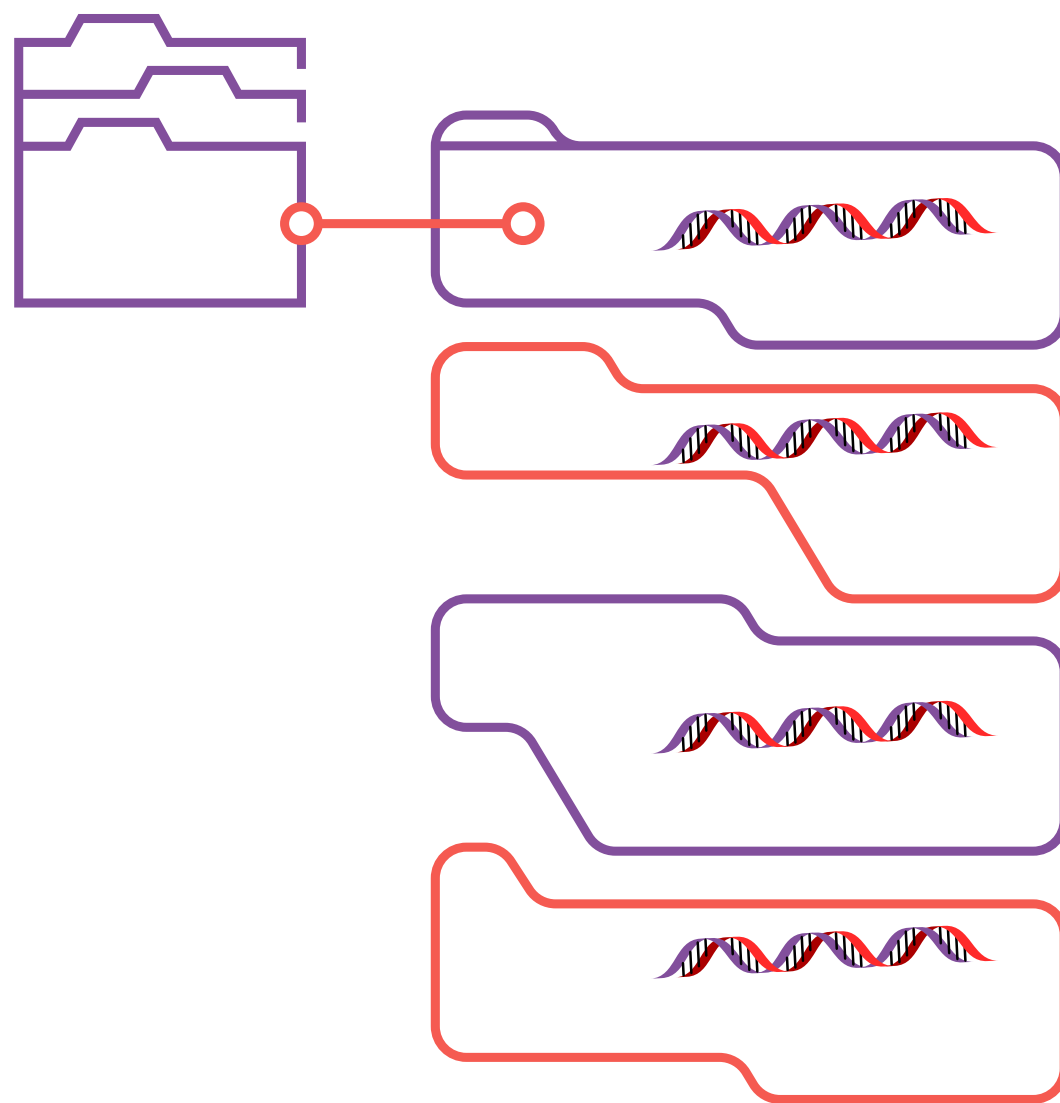
Mulheres em
Bioinformática
& Data Science LA
Promovendo a colaboração entre mulheres

O problema do acúmulo de dados biológicos



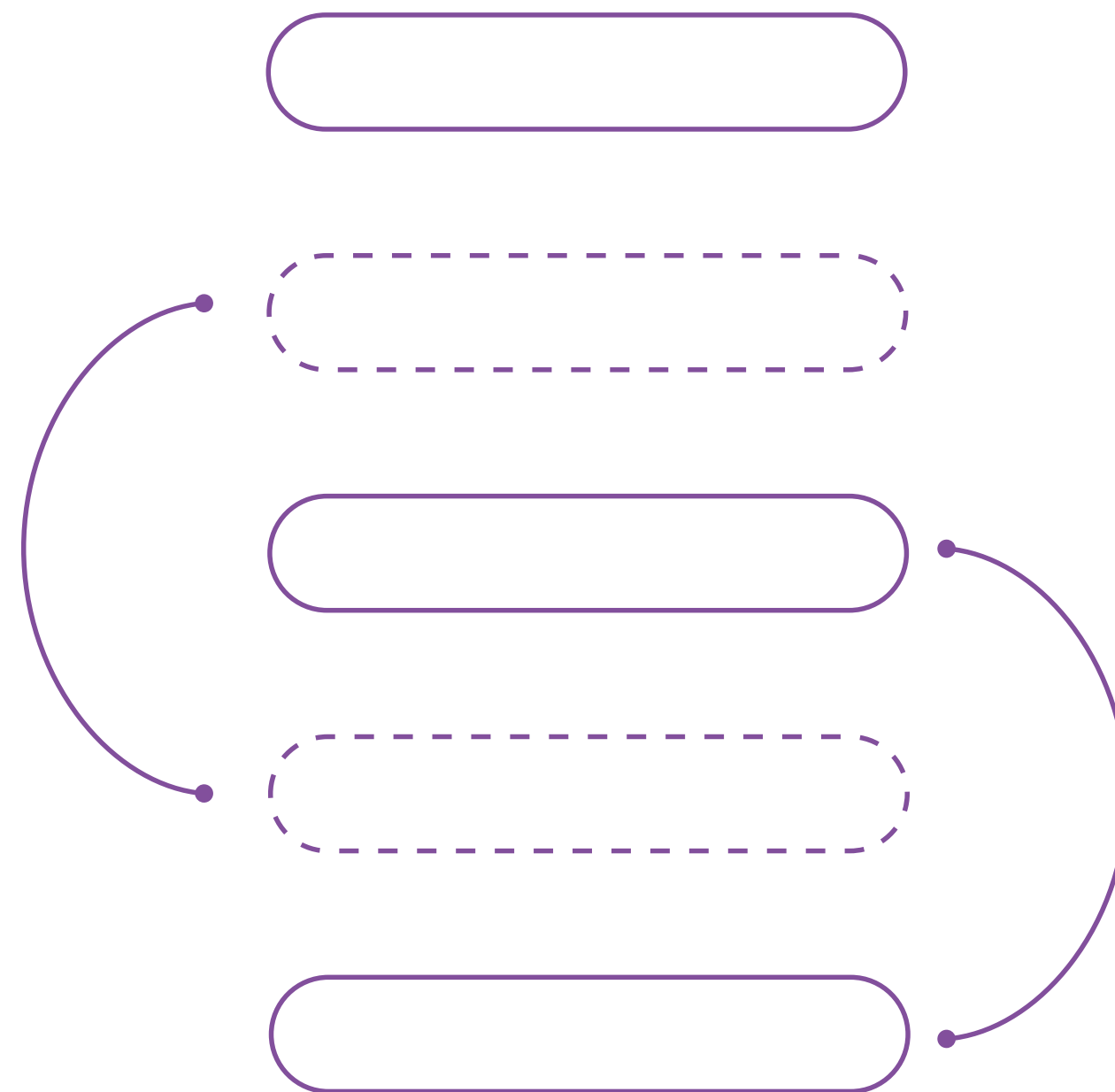
- Por que possuímos tantos dados biológicos atualmente?
- Onde esses dados estão armazenados? Qualquer pessoa pode armazenar?
- "Os biólogos precisam cada vez mais trabalhar com dados in silico". Por que?
- Por que é importante conhecer algoritmos?

O problema do acúmulo de dados biológicos

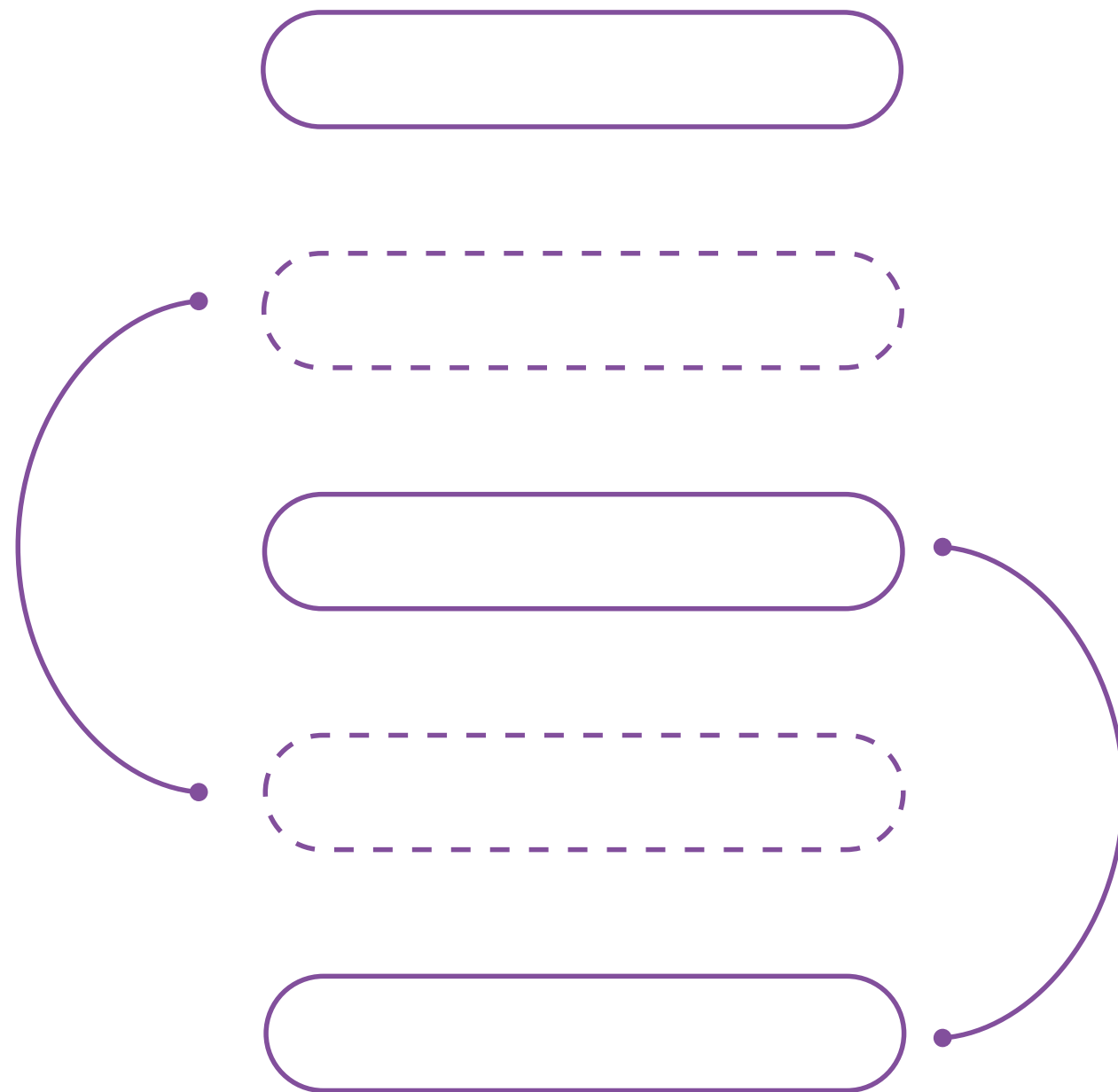


- Estocar
- Analisar
- Comparar
- Buscar
- Classificar
- Etc.

O que é um algoritmo?



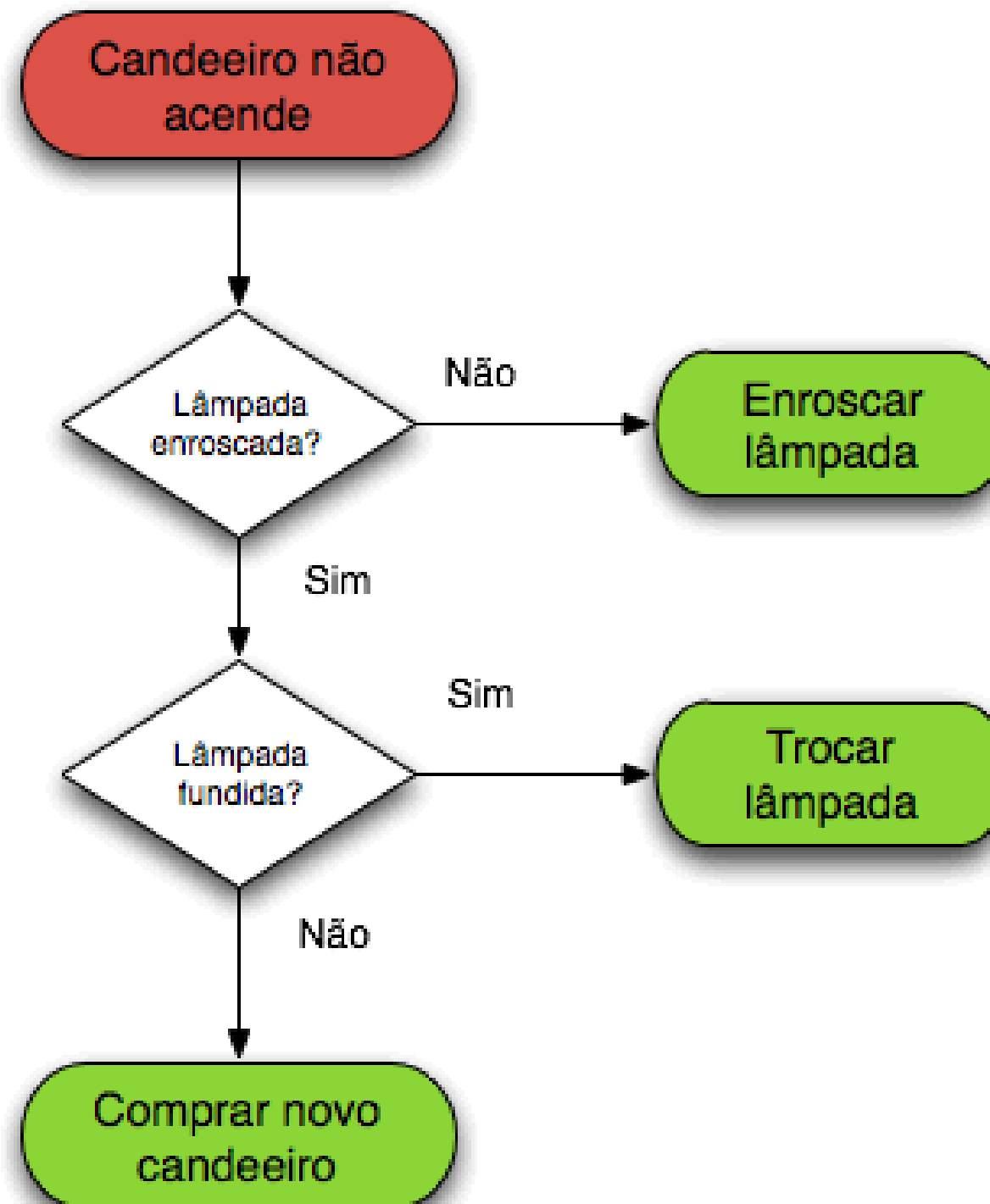
O que é um algoritmo?



"Um algoritmo é uma sequência de passos ou instruções, em ordem e sem ambiguidade, que deve ser seguida para resolver um problema."

Já a programação é referente à execução dessas instruções – passo a passo – no computador"

O que é um algoritmo?



O que é um algoritmo?

Algoritmo 2 Pegar um onibus.

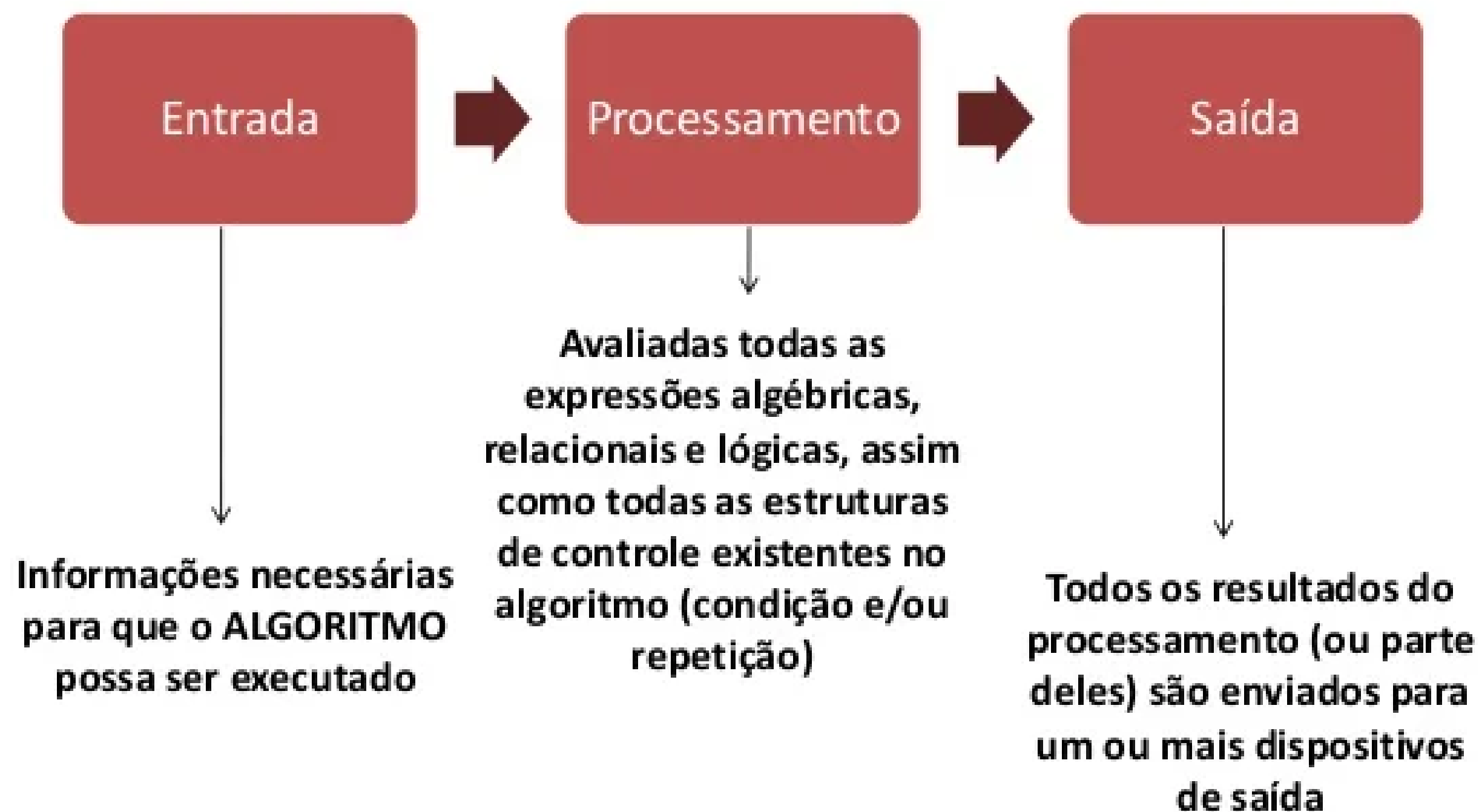
- 1: ir até a parada
 - 2: **enquanto** ônibus não chega **faça**
 - 3: esperar ônibus
 - 4: **fim-enquanto**
 - 5: subir no ônibus
 - 6: pegar passagem
 - 7: **se** não há passagem **então**
 - 8: pegar dinheiro
 - 9: **fim-se**
 - 10: pagar o cobrador
 - 11: troco \leftarrow dinheiro - passagem
 - 12: **enquanto** banco não está vazio **faça**
 - 13: ir para o próximo
 - 14: **fim-enquanto**
 - 15: sentar
 - 16: . . .
-

O que é um algoritmo?

"Computacionalmente falando, um problema é definido por uma relação de entrada/saída: recebemos uma entrada e queremos retornar como saída uma solução bem definida"

"O algoritmo deve ser descrito de acordo com a entidade que o executará: se for um computador, então o algoritmo deverá ser escrito em uma linguagem de programação."

Partes de um algoritmo



O Problema de ordenação

Example: Sorting Problem

INPUT: A sequence S of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.

OUTPUT: A permutation $\langle a'_1, a'_2, \dots, a'_n \rangle$ of S such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Exemplo:

Entrada -> $S = 5, 6, 1, 7, 3$

Processamento -> a lógica da ordenação

Saída -> uma permutação (ou seja, uma nova ordenação para a sequência) de modo que o primeiro elemento da nova sequência seja menor ou igual ao segundo e assim por diante. $S = 1, 3, 5, 6, 7$

"Dado um problema, podem haver muitos algoritmos que o resolvam corretamente, mas em geral nem todos serão igualmente eficientes."

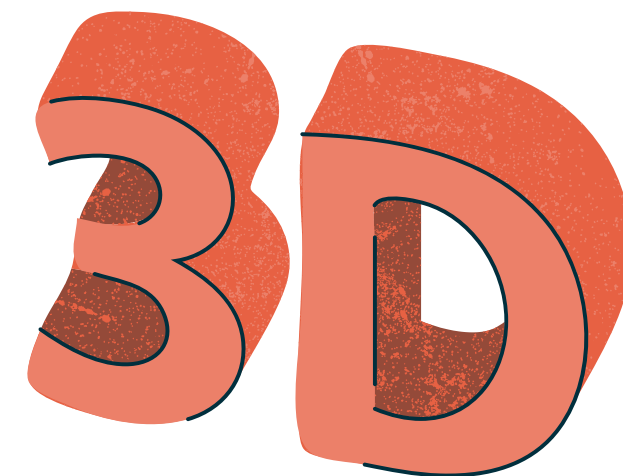
Por que?

Complexidade

- É a quantidade de recursos demandada por um algoritmo, independente do hardware e da linguagem de programação no qual está sendo executado.

Tipos de complexidades:

- De tempo;
- De espaço



Complexidade de TEMPO

- É calculado em função da quantidade de operações simples às quais é atribuído um custo **unitário** ou **constante** com respeito ao tamanho da entrada;

Custos de tempo insignificantes:

- Tempo de execução constante;
- Um fator constante somado com um polinômio de grau mais alto em n ;
- Um fator constante multiplicando um polinômio mais alto.



Complexidade de ESPAÇO

São as **estruturas de dados** que um algoritmo mantém na **memória** durante sua execução.

Não é o tamanho do programa que descreve um algoritmo.



Complexidade

"Muitas vezes, a complexidade do tempo é mais preocupante do que a complexidade do espaço."

Por que?



A eficiência de um algoritmo é uma função de seu tamanho de entrada.

Possível solução para o problema de ordenação:
gerar todas as permutações possíveis para a sequência.
E a partir daí verificar qual está ordenado.

- Cerca de quantos e quais passos levariam para se chegar na sequência ordenada?
- "Com este procedimento, é preciso ter sorte para encontrar a ordenação correta rapidamente". Por que?
- Você consegue ver que quanto maior a entrada, maior será a complexidade de tempo?

"Para avaliar o tempo de execução de um algoritmo independentemente do hardware específico no qual ele é executado, este é calculado em termos da quantidade de operações simples às quais é atribuído um custo unitário ou, no entanto, um custo constante com respeito ao tamanho da entrada"

BIG-O

Big-O é uma forma de classificar a complexidade do nosso algoritmo. Quanto tempo levará para executar aqueles passos? - É o termo dominante da função e mostra o comportamento do algoritmo!!! Ex.:

Big-O avalia apenas o comportamento de um algoritmo, como ele cresce em relação ao tamanho da entrada.

"O que conta é o fator de crescimento em relação ao tamanho da entrada, ou seja, a complexidade assintótica $T(n)$ à medida que o tamanho da entrada n cresce"

Com o big-O podemos avaliar se o nosso algoritmo segue um comportamento Constante, Linear, Exponencial, cúbico, etc.

Sempre é escolhido o pior caso - o limite ou o teto (é o máximo de recursos que pode usar)



PARA PENSAR:

Complexidades de TEMPO e de ESPAÇO podem ser do tipo constante, linear, exponencial, polinomial, etc. ou só os de TEMPO podem ter estas características?

COMPLEXIDADE LINEAR:

```
1.50 def inverter_lista(lista):  
    tamanho = len(lista)  
    limite = tamanho//2  
    for i in range(limite):  
        aux = lista[i]  
        lista[i] = lista[tamanho-i]  
        lista[tamanho-i] = aux  
  
    # 4 + N complexidade de espaço  
    # 2 + 2*N - complexidade de tempo = O(n)
```

Exemplo:

4 linhas + N --> Complexidade de espaço

2 linhas + 4 linhas / 2 (tamanho//2) --> Complexidade de tempo

Características:

- Termo dominante: o termo que multiplica ao N;
- Um único algoritmo iterativo --> Linear $O(n)$

COMPLEXIDADE EXPONENCIAL

```
def tem_duplicados(lista):  
    for i in range(len(lista)-1):  
        for j in range(i+1, len(lista)):  
            if lista[i] == lista[j]:  
                return True  
    return False  
  
# complexidade de tempo:  $N-1 + N-2 + N-3 + \dots + 1 = N*(N-1)/2$   
#  $(N^2 - N)/2 + 1 = O(n^2)$ 
```

Características:

- Termo dominante: o termo de maior polinômio;
- Um algoritmo iterativo dentro de outro iterativo (for - for) --> exponencial/polinomial $O(n^2)$

1. Constante:

São funções que independente da quantidade de valores de entrada passados, a performance do tempo de resposta dela se manterá a mesma - TRATÁVEIS!

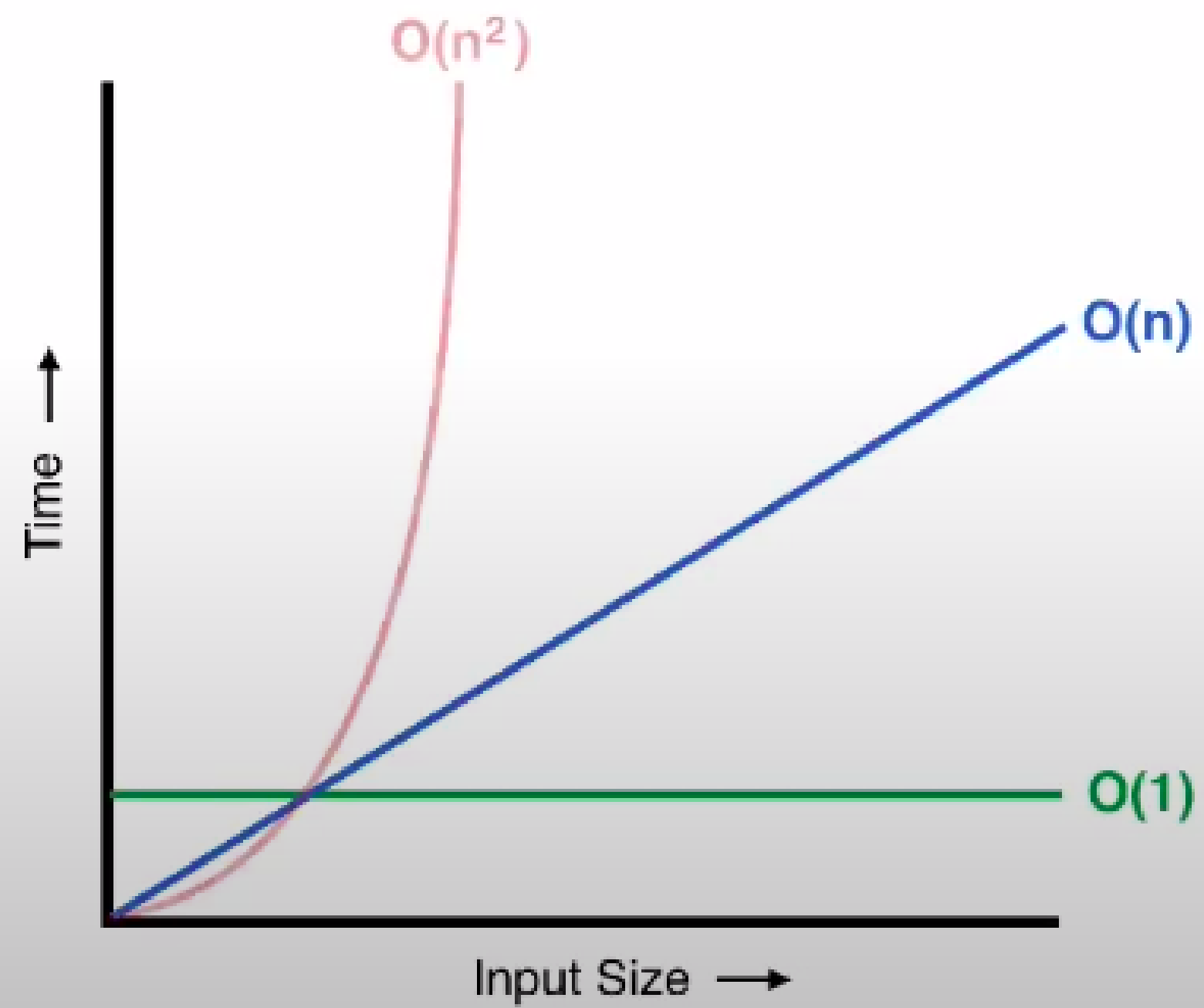
2. Linear:

"Algoritmo que varre uma entrada de tamanho n um número constante de vezes e, em seguida, executa um número constante de algumas outras operações, leva tempo $O(n)$ e é dito ter complexidade de tempo linear. Um algoritmo que leva tempo linear apenas no pior caso também é dito em $O(n)$, porque a notação big-O representa um limite superior." - TRATÁVEIS!

3. Exponenciais

Nesse tipo de algoritmo além de interagirmos com cada elemento de entrada, nós iremos realizar uma sub interação - INTRATÁVEIS!

Big O Notation

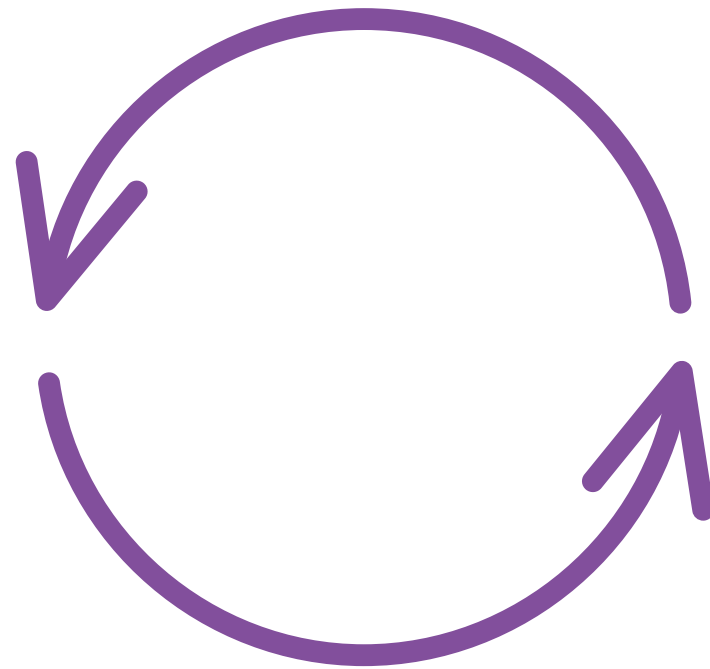




Discussão geral

Algoritmos iterativos

Um algoritmo iterativo é um algoritmo que repete uma mesma sequência de ações várias vezes; o número de vezes não precisa ser conhecido a priori, mas tem que ser finito.



- Quando usar for e quando usar while?

for x while

INSERTION-SORT(S, n)

 for $i = 1$ to $n - 1$ do

$j \leftarrow i$

 while ($j > 0$ and $S[j - 1] > S[j]$)

 swap $S[j]$ and $S[j - 1]$

$j \leftarrow j - 1$

 end while

 end for

exemplo

```
INSERTION-SORT(S,n)
  for i=1 to n-1 do
    j ← i
    while (j > 0 and S[j-1] > S[j])
      swap S[j] and S[j-1]
      j ← j-1
    end while
  end for
```

$S = [8, 5, 1, 9]$

For: $O(n^2)$ porque todos os elementos de S devem ser lidos, e de fato, o comando for é executado $Y(n)$ vezes: uma por cada posição da matriz, da segunda à última.

"complexidade de tempo de INSERTION-SORT não pode, no entanto, ser provada como ótima, pois o limite inferior para o problema de ordenação não é n^2 , mas sim $n \log_2 n$ (resultado não comprovado aqui). Para alcançar a complexidade de tempo $O(n \log_2 n)$ precisamos de um paradigma ainda mais poderoso que apresentaremos na próxima seção"



Discussão geral

Algoritmos recursivos

Um algoritmo recursivo é um algoritmo que, entre seus comandos, chama-se recursivamente em instâncias menores: divide o problema principal em subproblemas, resolve-os recursivamente e combina suas soluções para construir a solução do problema original.

```
MERGE-SORT(S,p,r)
  if  $p < r$  then
     $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
    MERGE-SORT(S,p,q)
    MERGE-SORT(S,q + 1,r)
    MERGE(S,p,q,r)
  end if
```

As três leis da recursividade

1. Um algoritmo recursivo deve possuir um **caso base** (*base case*).
2. Um algoritmo recursivo deve modificar o seu estado e se aproximar do caso base.
3. Um algoritmo recursivo deve chamar a si mesmo, recursivamente.

```
function multiplica(num1, num2)
{
    //Multiplicação por 0 é igual a 0
    if (num1 == 0 || num2 == 0) {
        return 0;
    }
    //Caso base, onde a recursão para
    else if (num2 == 1){
        return num1;
    }
    //Multiplicando através da soma com recursividade
    else {
        return (num1 + multiplica(num1, num2 - 1));
    }
}

var result = multiplica(5,4);
document.write(result);
```



Algoritmos iterativos x Algoritmos recursivos



Discussão geral



Vamos compartilhar conteúdos?

Referências:

<https://drive.google.com/file/d/1BuaH8ZQzEwRJET2-hfa9E1CM-sUfDp5i/view>

Sobre complexidade e big -O:

<https://www.youtube.com/watch?v=KVIGx-9CuO4>

<https://www.youtube.com/watch?v=UQzCFkRbIrE>