# Algoritmos

## 1 – Introdução

Os algoritmos podem ser representados de três maneiras básicas: narração descritiva; fluxogramas e pseudocódigo.

Mas antes, é importante conhecer algumas noções informais sobre *o que é um algoritmo*.

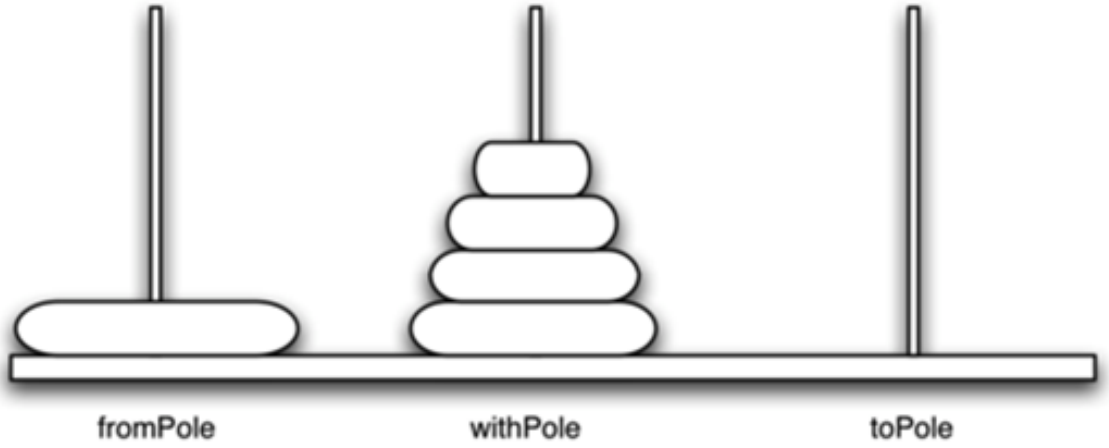
Considere três hastes que servem de suporte a discos de diferentes tamanhos, os menores sempre sobre os maiores. Pode-se mover cada disco para qualquer haste desde que um disco maior nunca fique sobre um disco menor. O objetivo é transferir os três discos de uma haste inicial para uma outra haste.

Eis então um belo exemplo do que chamamos de algoritmo para resolver esse desafio:

1 – Mova a torre de (altura – 1) para o pino intermediário usando o pino destino como intermediário.

2 – Mova o disco restante para o pino destino.

3 – Mova a torre de (altura - 1) do pino intermediário para o pino destino usando o pino origem como intermediário.




Como este algoritmo recorre a si mesmo (há uma chamada para dentro dele alterando parâmetros), chamamos tal algoritmos de **recursivo**.

Podemos representar algoritmos pela linguagem comum desde que os passos sejam claros e bem definidos.

Exemplo para um algoritmo associado à troca de uma lâmpada:

- Pegar uma escada;
- Posicionar a escada embaixo da lâmpada;
- Buscar uma lâmpada nova;

- Subir na escada;
- Retirar a lâmpada velha;
- Colocar a lâmpada nova.

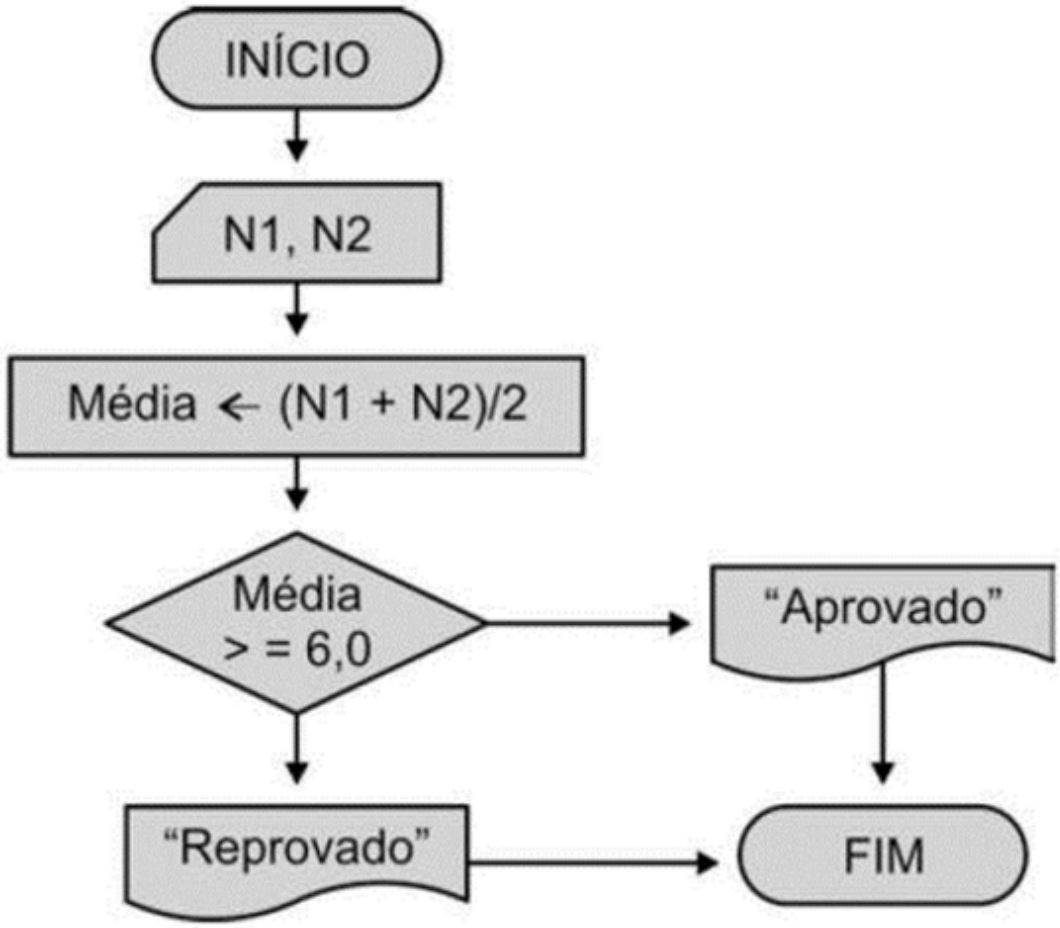
Podemos representar ainda um algoritmo por meio de uma “quase” linguagem de programação. Na pseudo-linguagem utilizamos os elementos típicos de uma linguagem de programação mas ela não é, em princípio, executada por um computador. Alguns esforços de tornar mais interessante uma pseudo-linguagem e facilitar o aprendizado do novato em programação têm sido empreendidos gerando ambientes interessantes.

Na listagem abaixo, representamos um algoritmo em pseudo-linguagem derivado do ambiente VISUALG (https://visualg3.com.br/, um ambiente de codificação e interpretação de comandos muito didático e interessante por meio do qual pode-se aprender bastante sobre sintaxe de uma linguagem, estruturas de controle e lógica de programação.

Leia com bastante atenção e tente descobrir o que faz o seguinte algoritmo:

```
1. início
2.    // declaração de variáveis
3.    real: N1, N2, N3, N4, // notas bimestrais
4.          MA; // média anual
5.    leia (N1, N2, N3, N4); // entrada de dados
6.    MA ← (N1 + N2 + N3 + N4) / 4; // processamento
7.    escreva (MA); // saída de dados
8.    se (MA >= 7)
9.       então
10.         escreva ("Aluno Aprovado!");
11.   fimse;
12. fim.
```

Outra forma de representação de um algoritmo é por meio de um fluxograma, espécie de diagrama em que a lógica de um algoritmo é representada por meio de um fluxo ligando elementos gráficos. Observe o exemplo abaixo:



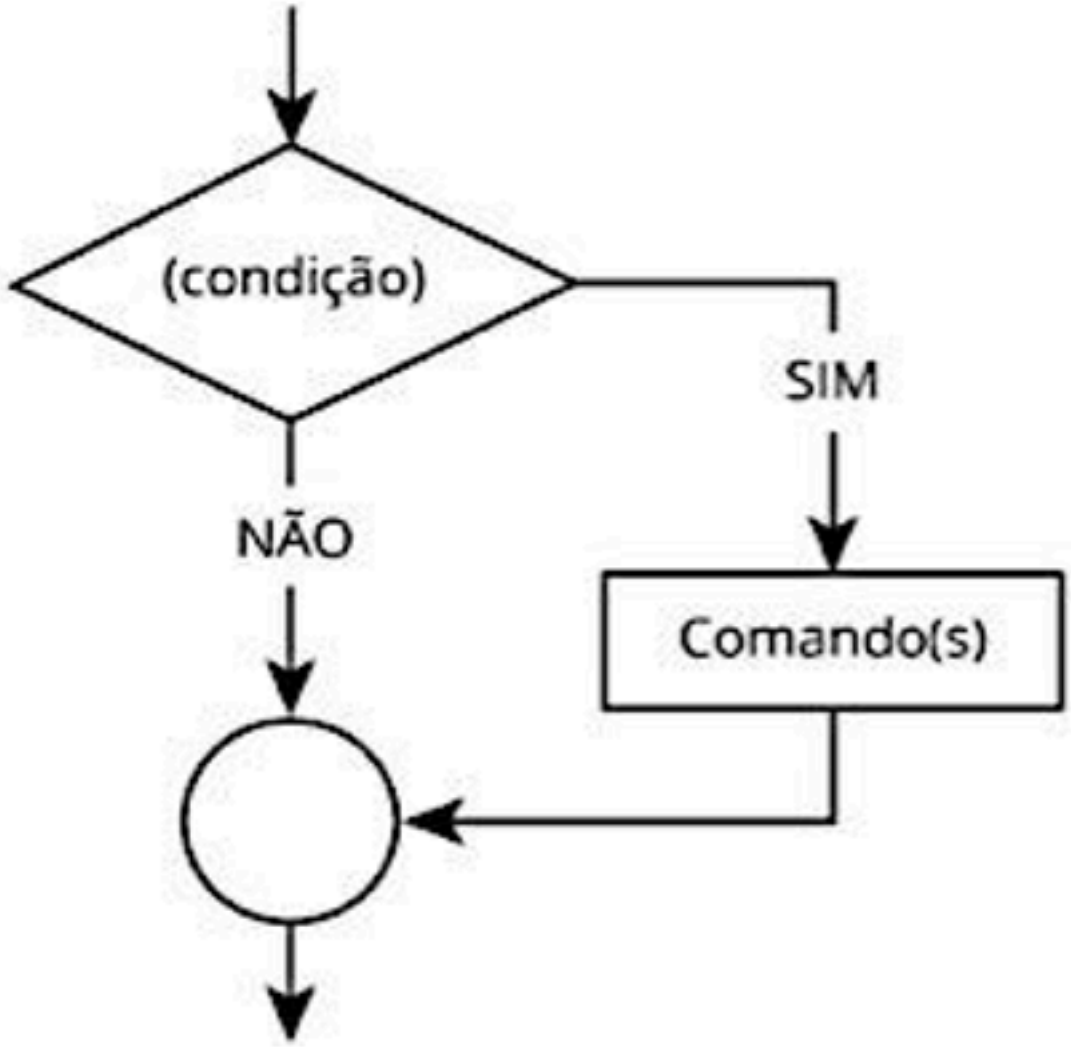

Durante muito tempo, a representação de algoritmos por meio de fluxogramas foi utilizada. Entretanto, novas ferramentas e pesquisas mostraram que fluxogramas ocultam representações que podem ser importantes (a estrutura de dados, por exemplo) e a ferramenta foi caindo em certo descrédito.

## 2 - Estruturas de controle

O fluxo de execução de um algoritmo está sujeito a desvios dependendo do atendimento ou não de certas condições. Chamamos os elementos responsáveis por essas mudanças e desvios de **estruturas de controle**.

2.1 - A condição **"se-(então)(-senão)"** é uma estrutura de seleção comum em muitas linguagens de programação.

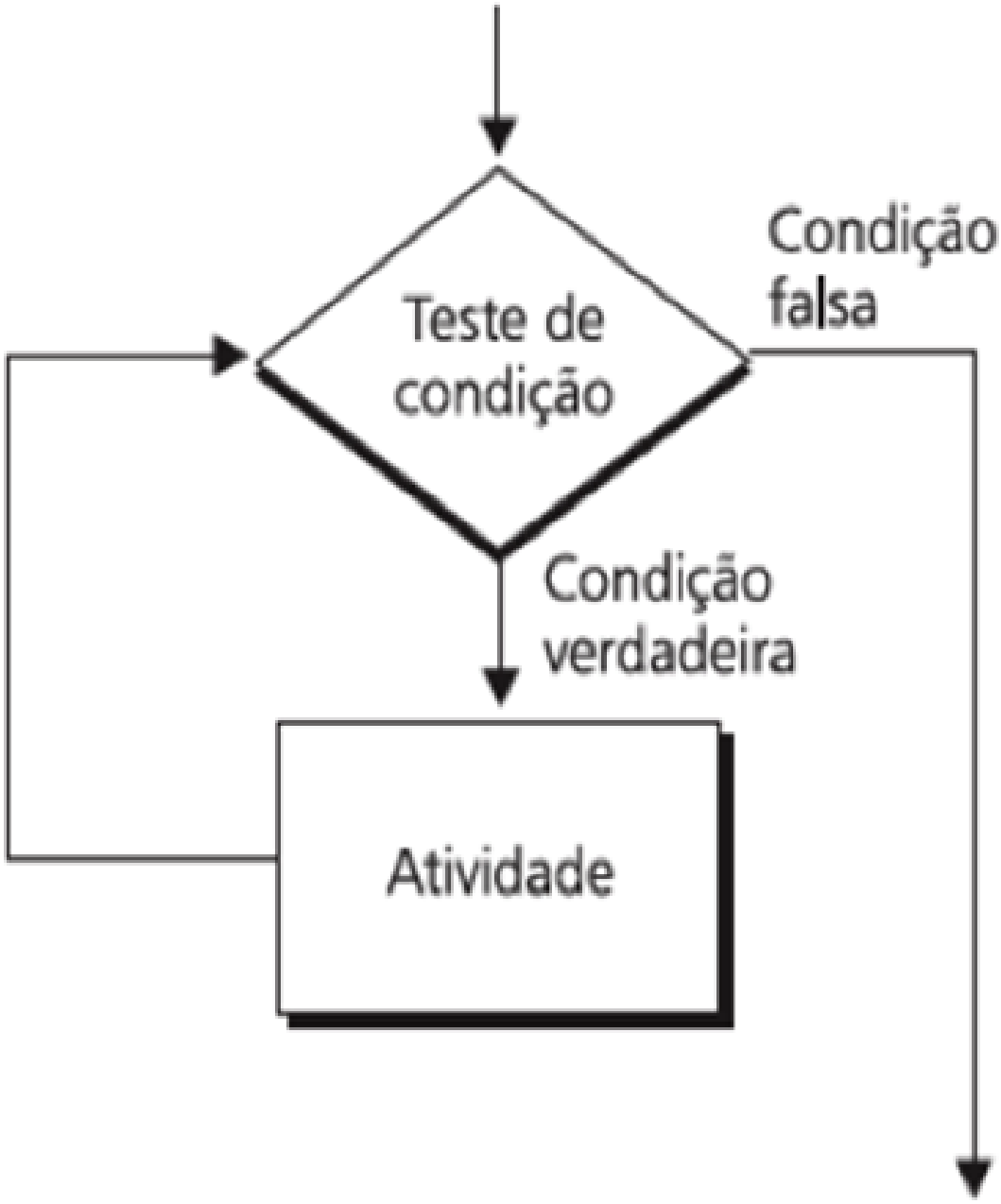
Utiliza expressões booleanas (do tipo lógica verdadeiro/falso) para desviar a execução do código para um fluxo ou outro, dependendo do valor avaliado ser verdadeiro ou falso.



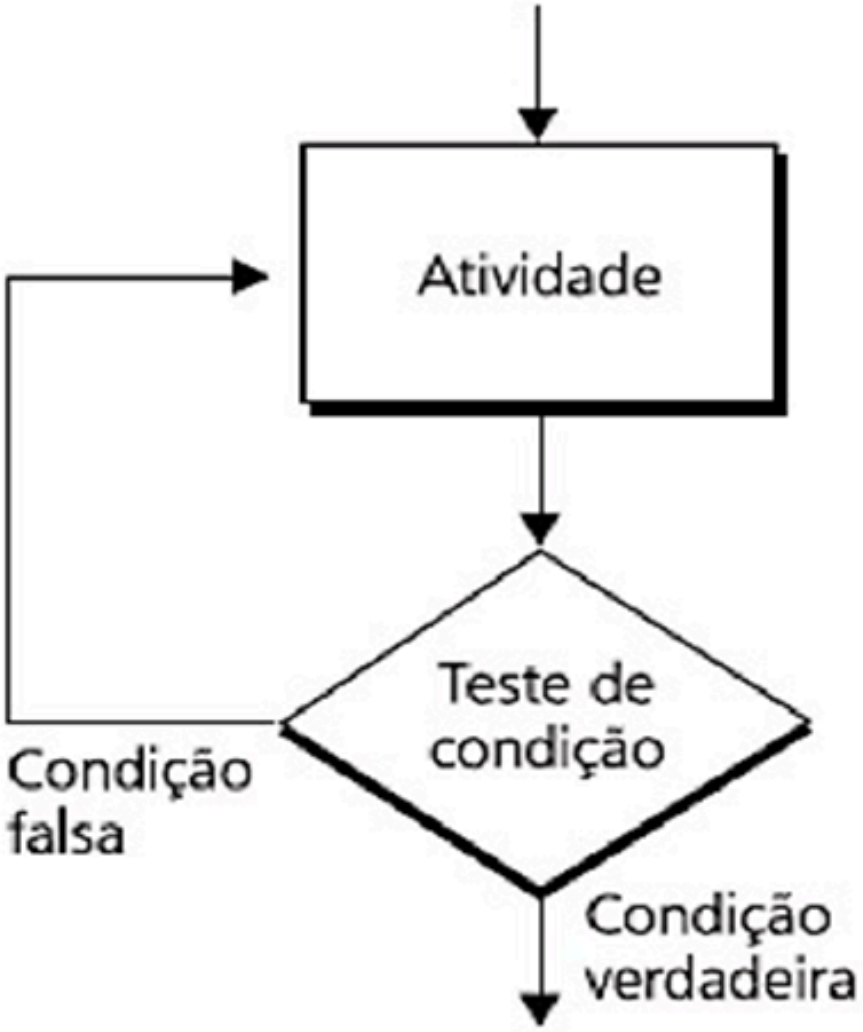

**SE** (condição) **Então**
{comando(s)}
**Fim-se**

2.2 - A estrutura **ENQUANTO** repete uma sequência de comandos **enquanto** uma determinada condição ou expressão lógica, for satisfeita.

A expressão é avaliada **antes** de cada repetição do laço interno. Quando seu resultado for VERDADEIRO, a <sequência-de-comandos> é executada. Quando a condição não for mais satisfeita, o fluxo desvia do laço de execução.




2.3 - A estrutura de repetição **REPITA...ATÉ** é utilizada quando um conjunto de comandos deve ser executado repetidamente até que uma condição (expressão lógica) seja verdadeira. Note a diferença para a estrutura ENQUANTO, aqui a condição é testada ao final do laço!



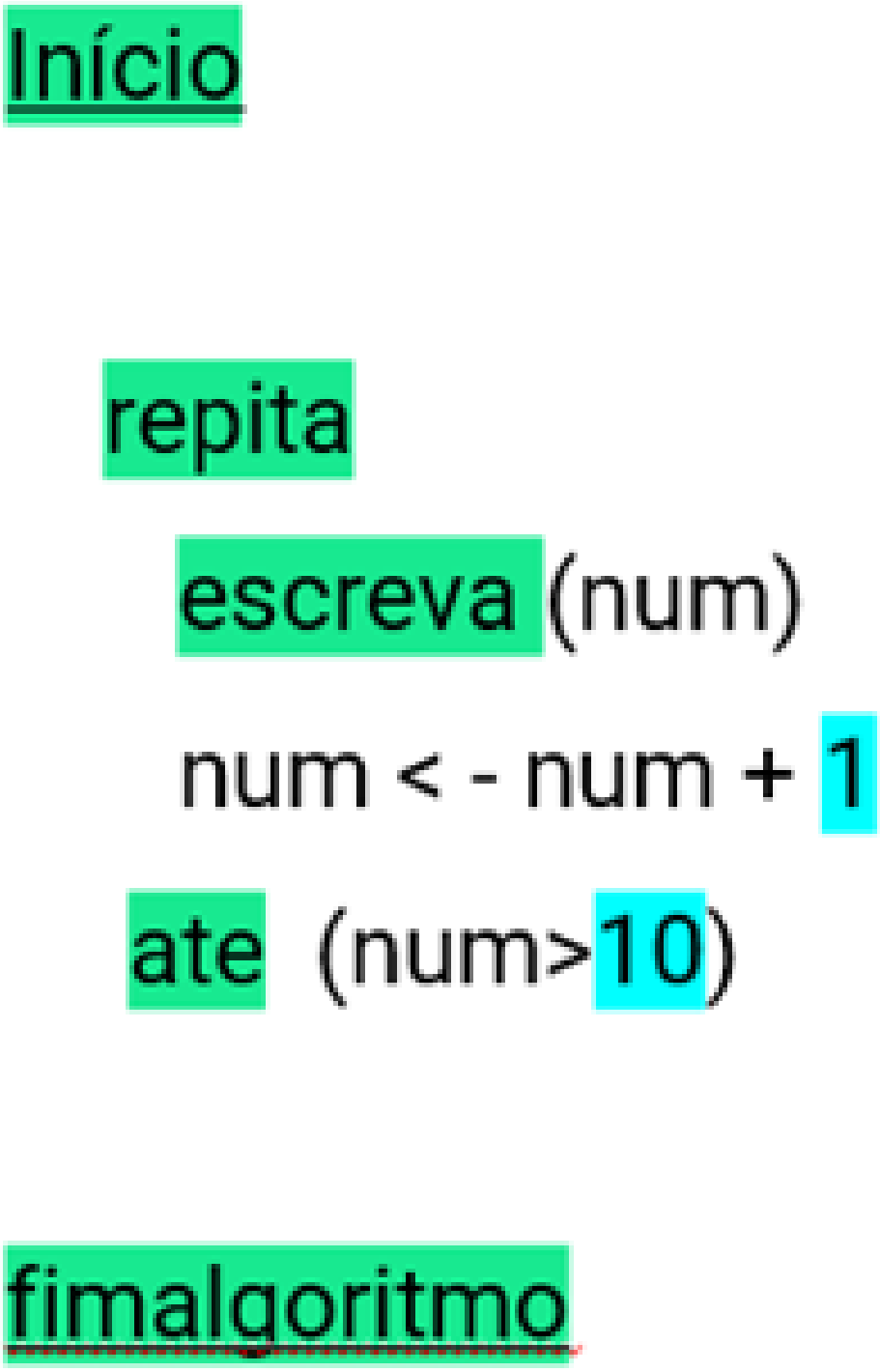

Isso significa, na prática, que ao adotar a estrutura REPITA sua escolha fará com que os comandos do laço sejam executados pelo menos uma vez !

### 3 – Exemplos

Observe os algoritmos abaixo atentamente. Este algoritmo escreve uma sequência de valores na tela de 1 até 10 usando a estrutura ENQUANTO.

```
Algoritmo "Números de 1 a 10 (com enquanto...faca)"
var i: inteiro
início
i < - 1
Enquanto i <= 10 faca
   escreva (i)
   i < - i + 1
fimenquanto
fimalgoritmo
```

Este algoritmo, por sua vez, escreve uma sequência de valores na tela de 1 até 10 usando a estrutura REPITA.




É interessante observar que estes algoritmos embora ligeiramente diferentes, geram o mesmo resultado. Pode-se usar outras estruturas de controle quando se sabe, *a priori*, o número de repetições como

**PARA** <variável contadora> **DE** <valor inicial> **ATE** <valor final> [**PASSO** <valor de incremento>] **FAÇA**

<instruções a serem executadas repetidamente até a <variável contadora> atingir o valor final>

**FIM**-PARA

Exemplo:

```
Algoritmo
    real MA // média anual de cada aluno
    Real ACM // acumulador
    real MAT // media anual da turma
    inteiro V // variável de controle
    ACM ← 0
    Para V de 1 até 50 passo 1
        Leia(MA)
        ACM ← ACM + MA // acumula as medias de cada aluno
    Fim-para
    MAT ← ACM/50
    Escreva("Media anual da turma =", MAT)
Fim-algoritmo
```

**4 – Exemplo de análise e comparação entre algoritmos: BUSCA de um elemento em uma lista.**

Vamos agora aprofundar um pouco.

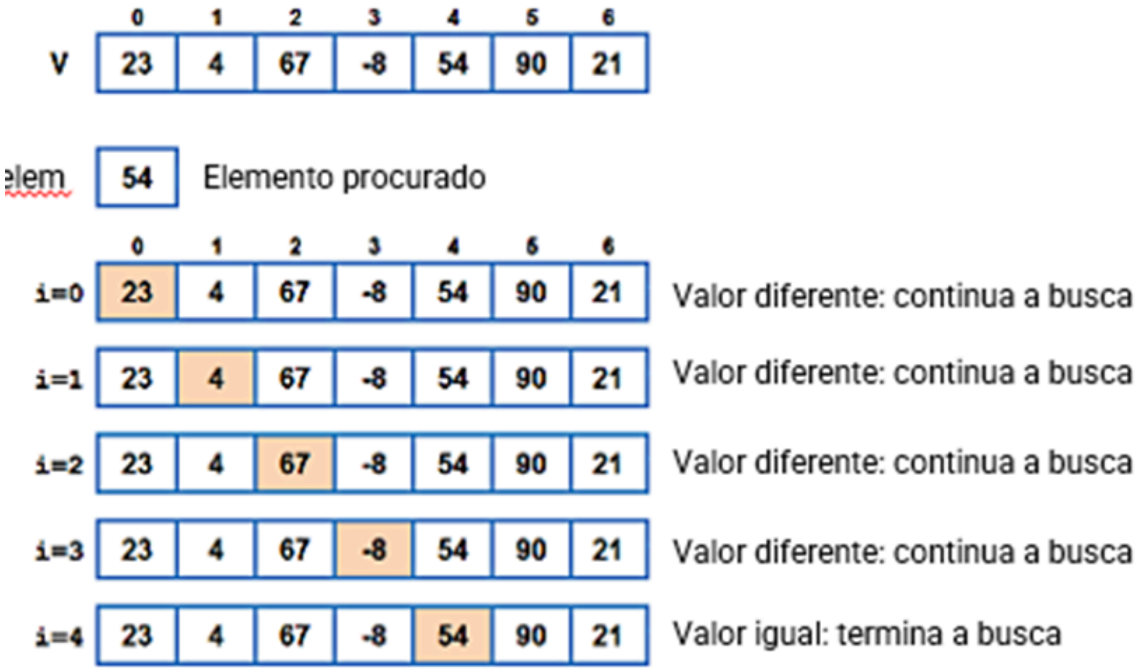
Em situações um pouco mais complexas, podemos ter métodos algorítmicos diferentes. O "preço" que se paga depende do tamanho da entrada de valores, do número de comparações realizadas, de situações específicas onde opera o algoritmo, etc. Vamos a um exemplo que é clássico e importante.

*Dada uma lista de valores, o problema consiste em saber se um determinado elemento pertence ou não pertence à lista.*

### 4.1 – **Busca linear**

O método da **busca linear** (ou sequencial) faz com que o elemento em cada posição da lista seja comparado ao elemento que se busca.

Esse processo é bastante simples e está ilustrado na figura abaixo em que se busca o número 54 na lista V.




Este algoritmo é chamado de **algoritmo de busca sequencial ou linear**.

Perceba que se o elemento não estiver na lista, todos os elementos da lista serão visitados. Se a lista for muito grande, isso pode representar um problema!

## 4.2 – **Busca binária**

Tudo começa com a lista ordenada, só é possível busca binária em listas ordenadas.

O valor a ser buscado é comparado com o valor do meio da lista. Se o valor a ser procurado é maior do que o valor do meio da lista, descartamos a metade inferior da lista, afinal ele não estará lá com certeza.

Concentramos os esforços então na lista que sobrou. Repetimos o processo analisando o elemento que está no meio da lista. Assim, de forma repetida e rápida vamos descartando os elementos e chegando cada vez mais perto da resposta, se o número está ou não na lista.

Repare que quando consultamos um dicionário, em geral fazemos uma espécie de busca binária. Abaixo o número a ser procurado é o 4 na lista V.


Ao comparar os dois métodos, busca linear e busca sequencial, é importante observar:

- Se a lista não está ordenada, a busca binária não se aplica.

- A busca sequencial é ineficiente para conjuntos grandes de valores de entrada.

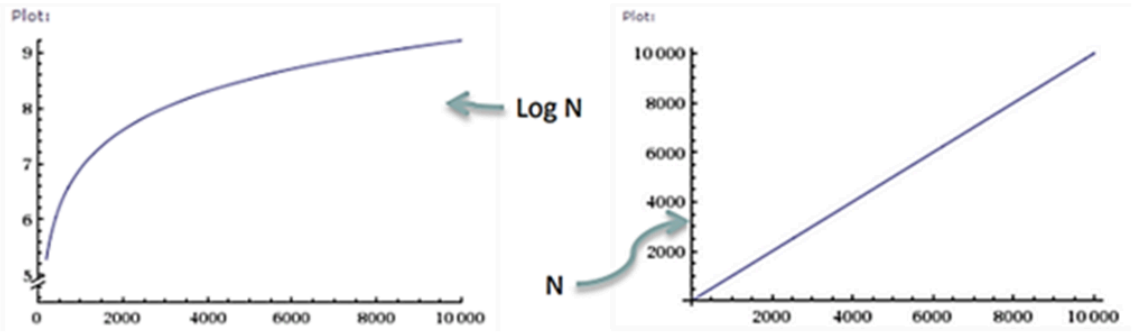
- Se o valor a ser buscado estiver logo no começo da lista, a busca sequencial é rápida, mas isso é raro acontecer, casos ótimos ou exceções não costumam ser muito considerados em análises de algoritmos.

- Olhando o comportamento matemático dos processos computacionais (complexidade assintótica) podemos enxergar os algoritmos por meio de gráficos de funções. No eixo X a quantidade n de elementos, no eixo Y o número de comparações.

- A busca sequencial tem *complexidade* ***n*** e a busca binária tem *complexidade* ***log n.***

- Como a função **log n cresce significativamente mais devagar do que a busca linear** em função do tamanho da entrada, dizemos que faz menos comparações, sendo mais eficiente do que a busca sequencial.

Abaixo você observa o gráfico à esquerda da busca binária e à direita sequencial (esta cresce mais rápido).

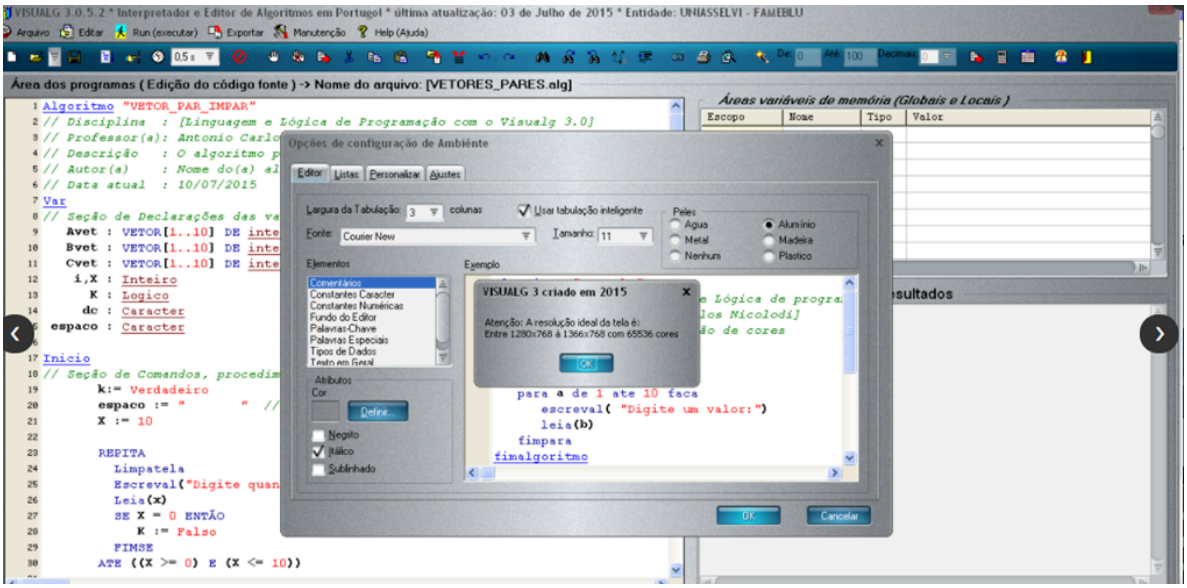


Importante reforçar que o algoritmo da busca binária só opera a partir de uma lista ordenada de valores ! Existem outros métodos de busca bastante interessantes, o intuito aqui foi ilustrar diferentes abordagens para a situação comum de busca de um elemento em uma lista.

## 5 – Dica: VisualG (https://visualg3.com.br/)

Conforme vimos, o VisualG permite criar, editar, interpretar e que também executa os algoritmos em portugol (estruturado português) como se fosse um "programa" normal de computador.

É um programa de livre uso e distribuição, GRÁTIS e DOMÍNIO PÚBLICO, usado para o ensino de lógica de programação em várias escolas e universidades no Brasil e no exterior.




Referência Bibliográfica

BROOKSHEAR, J.G. **Ciência da Computação: uma visão abrangente.** Porto Alegre: Bookman, 2013.

CÓRDOVA JUNIOR, R. S. et al. **Fundamentos computacionais.** Porto Alegre: SAGAH, 2018,

FORBELLONE, A.L.V. & EBERSPACHER, H. F. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados.** 3ª. Edição. São Paulo, SP: Prentice Hall, 2005.

## Ir para exercício