



# **Pesquisa Sequencial e Binária**

Prof. Wylliams Barbosa Santos  
wylliamss@gmail.com  
Introdução à Programação

Crédito de Conteúdo: Professora Ceça Moraes

# Agenda

- Pesquisa Sequencial
- Noções de complexidade
- Pesquisa Binária

# Contexto

- Diferentes estratégias para pesquisa (busca) de um elemento específico em um conjunto de dados.
  - **Lista, array, coleção**
- Operação **importante**, encontrada com muita frequência em diversas aplicações
- Dois métodos mais conhecidos:
  - **Busca Seqüencial linear**
  - **Busca Binária**

- Forma **mais simples** de realizar pesquisas.
- **Metodologia:**
  - **Pesquisa bem sucedida:**

É efetuada a verificação de cada elemento do conjunto, sequencialmente, até que o elemento desejado seja encontrado;
  - **Pesquisa mal sucedida:**

Todos os elementos do conjunto tenham sido verificados sem que o elemento procurado tenha sido encontrado;

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
14	21	5	45	12	3	86	98	46	53	24	2	1	15	90	47

- **Questão 1:**
  - O elemento 90 está presente no vetor?
- **Questão 2:**
  - Quantas comparações são necessárias para achar o elemento 90?

# Características

- Algoritmo extremamente **simples**
- Pode ser muito **ineficiente** quando o conjunto de dados se torna muito **grande**
  - **Alto** número de comparações

# COMPLEXIDADE DE ALGORITMOS

# Complexidade

- A Complexidade de um Algoritmo consiste na “**quantidade de trabalho**” necessária para a sua execução, expressa em função das operações fundamentais.
  - Operações críticas variam de acordo com o **algoritmo**, e em função do **volume de dados**
  - Por exemplo, na pesquisa sequencial é fundamental as **comparações** entre elementos



# Complexidade

- Um algoritmo serve para resolver um determinado problema, e os problemas têm sempre uma **entrada de dados (E)**
- O tamanho de **E (N elementos)** afeta diretamente o **tempo de resposta** do algoritmo
- Dependendo do problema, já existem alguns **algoritmos prontos**, ou que podem ser adaptados
  - **Decisão: qual algoritmo escolher?**

# Complexidade

- A **complexidade** de um algoritmo pode ser dividida em:
  - **Complexidade Espacial**: quantidade de **recursos** utilizados para resolver o problema
  - **Complexidade Temporal**: quantidade de tempo utilizado. Pode ser visto também como o **número de passos** necessários para resolver determinado problema
- Em ambos os casos, a complexidade é medida de acordo com o **tamanho dos dados de entrada (N)**

# Complexidade

- Definimos a **expressão matemática de avaliação do tempo de execução** de um algoritmo como sendo uma **função que fornece o número de passos efetuados** pelo algoritmo a partir de uma certa **entrada**

# Exemplos

- **Soma de vetores**

para I de 1 até N faça

$$S[I] \leftarrow X[I] + Y[I]$$

fimpara

- **Número de passos = número de somas (N somas)**
  - Ordem de N ou **O(N)**

- **Soma de matrizes**

para I de 1 até N faça

para J de 1 até N faça

$$C[I,J] \leftarrow A[I,j] + B[I,J]$$

fimpara

fimpara

- **Número de passos = número de somas (N\*N somas)**
  - Ordem de  $N^2$  ou  **$O(N^2)$**

# Exemplos

- **Produto de matrizes**

para I de 1 até N faça

para J de 1 até N faça

$P[I,J] \leftarrow 0$

para K de 1 até N faça

$P[I,J] \leftarrow P[I,J] + A[I,K] * B[K,J]$

fimpara

fimpara

Fimpara

- **Número de passos = Número de operações de somas e produtos ( $N * N * N$ )**

– Ordem de  $N^3$  ou  $O(N^3)$

# Tipos

- A complexidade pode ser qualificada quanto ao seu comportamento como:
  - **Polinomial**
    - à medida em que  $N$  aumenta o fator que estiver sendo analisado (tempo ou espaço) aumenta **linearmente**
  - **Exponencial**
    - A medida que  $N$  aumenta o fator que estiver sendo analisado (tempo ou espaço) aumenta **exponencialmente**



# Complexidade de Algoritmos

- Existem três escalas de complexidade:
  - Melhor Caso
  - Caso Médio
  - Pior Caso



# Melhor Caso

- Representado pela letra grega  $\Omega$  (Ômega)
- É o **menor tempo** de execução em uma entrada de tamanho N
- É **pouco usado**, por ter aplicação em poucos casos
- Exemplo
  - Se tivermos uma lista de N números e quisermos executar uma **busca sequencial** assume-se que a complexidade no melhor caso é de  **$N = 1$**
  - **$f(N) = \Omega(1) = 1$** , pois assume-se que o número estaria logo na primeira posição da lista

# Caso Médio

- Definido pela letra grega  $\theta$  (Theta)
- Dos três, é o mais **difícil** de se determinar
- Deve-se obter a **média** dos tempos de execução de todas as entradas de tamanho 1, 2,... até N, ou baseado em probabilidade de determinada situação ocorrer

# Pior Caso

- Representado pela letra grega **O** (**O** maiúsculo. Trata-se da letra grega *ômicron* maiúscula)
- É o método **mais fácil** de se obter
  - Baseia-se no maior tempo de execução sobre as entradas de tamanho N
- Exemplo:
  - Se tivermos uma lista de N números e quisermos executar uma **busca sequencial** assume-se que a complexidade no pior caso é  **$f(N) = O(N) = N$** , pois assume-se que o número estaria no pior caso, no final da lista

# Busca Sequencial

## Complexidade

- **Pior Caso:** é quando é necessário realizar **N** comparações (onde **N** é o número de elementos)
  - Qual o cenário de pior caso possível?
  - O elemento procurado na **última** posição
- **Melhor Caso:** é quando é necessário realizar somente uma comparação
  - Qual o cenário de melhor caso possível?
  - O elemento procurado na **primeira** posição
- **Caso Médio:**  $(\text{Pior Caso} + \text{Melhor Caso})/2$

# Busca Sequencial Complexidade

- **Pior Caso:** n comparações
  - $O(n) = n$
- **Melhor Caso:** uma comparação
  - $\Omega(1) = 1$
- **Caso Médio:** (Pior Caso + Melhor Caso)/2
  - $\theta(n) = (n + 1) / 2$

# BUSCA BINÁRIA

# Busca Binária

- Algoritmo de busca em vetores com acesso **aleatório** aos elementos
- Parte do pressuposto de que o vetor está **ordenado**
- Realiza sucessivas **divisões** do vetor e compara o elemento buscado (chave) com o elemento no **meio** do segmento
- 3 opções:
  - Se igual, a busca termina com **sucesso**
  - Se o elemento do meio for menor que o elemento buscado, então a busca continua na **metade posterior** do vetor.
  - Se o elemento do meio for maior que a chave, a busca continua na **metade anterior** do vetor

# Busca Binária

## Metodologia

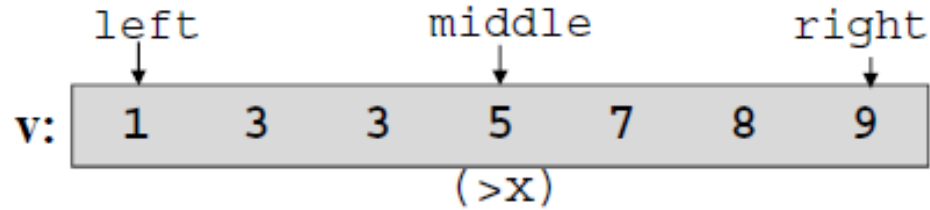
- 1) Checar onde está o **ponto médio** do vetor.
- 2) Comparar o **elemento do ponto médio (EPM)** com elemento **chave**.
- 3) Continuar a pesquisa da seguinte forma:
  - Se **chave=EPM**, então a pesquisa pára com sucesso, pois achou o dado desejado!
  - Se **chave<EPM** realizar a pesquisa no sub-vetor à **esquerda** do EPM, partindo do **passo 1**.
  - Se **chave>EPM** realizar a pesquisa no sub-vetor à **direita** do EPM, partindo do **passo 1**.



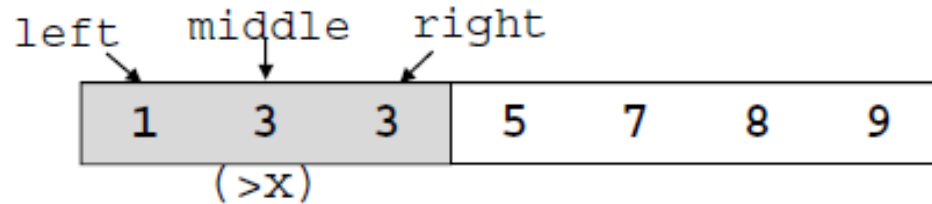
# Exemplo

x: 2

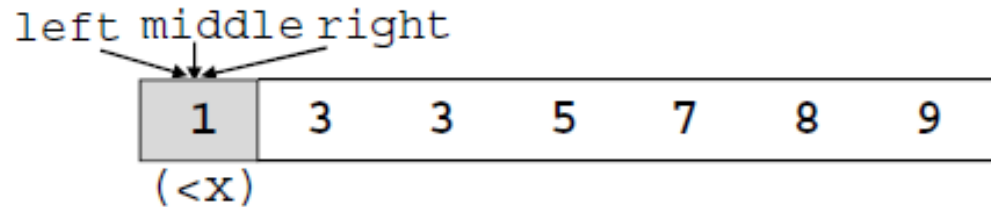
1ª iteração



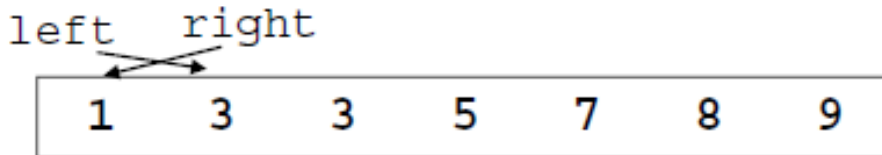
2ª iteração



3ª iteração



4ª iteração



**vetor a inspecionar vazio ⇒ o valor 2 não existe no vetor original !**

# Exemplo de Busca Binária

Exemplo Inicial:

Após ordenação:

↓ Ponto médio

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	5	12	14	15	21	24	45	46	47	53	86	90	98

**Pergunta:** Como verificar se o **elemento 90** está presente no vetor acima?

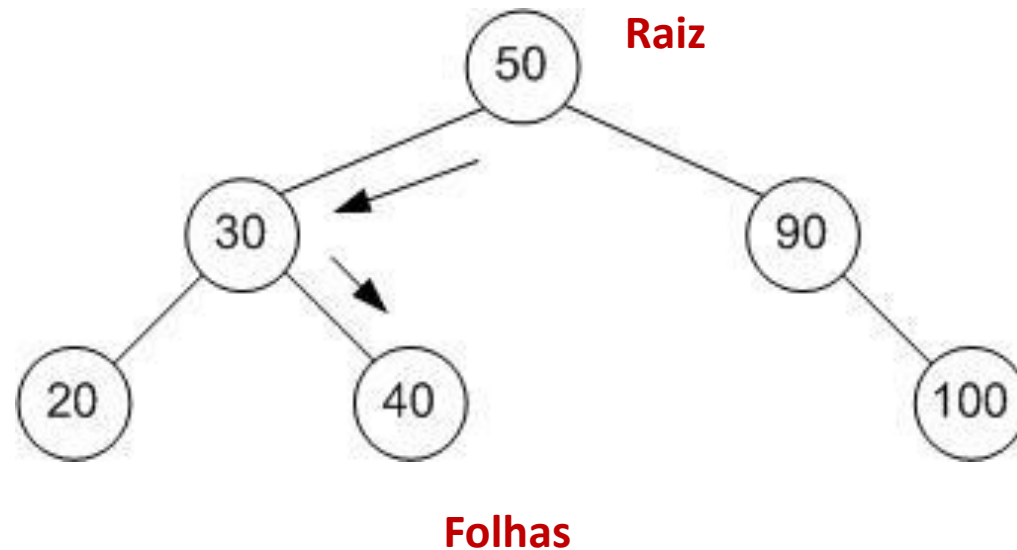
**Pergunta:** Quantas comparações são necessárias para achar o elemento 90?



# Busca Binária

- Divide-se o vetor como se este fosse uma “árvore”

20	30	40	50	90	100
----	----	----	----	----	-----



# Complexidade da Busca Binária

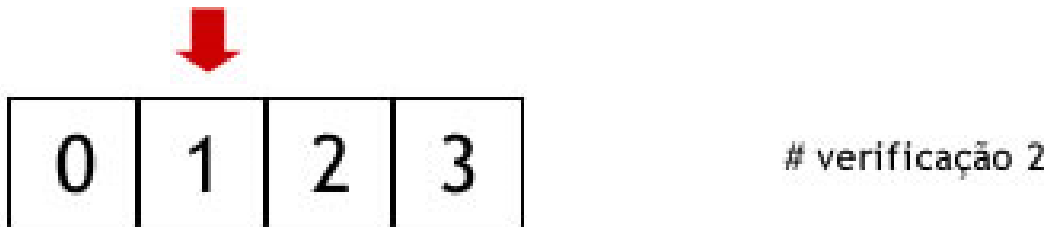
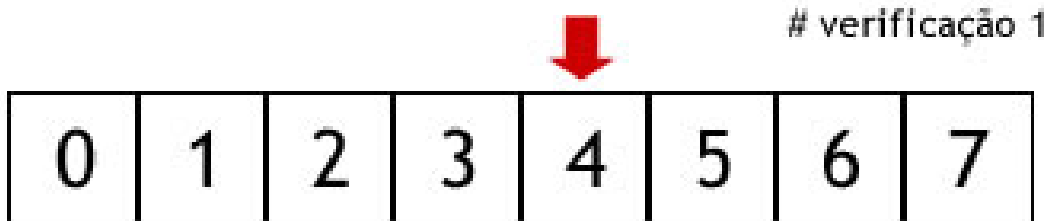
- **Pior Caso:** quando o dado desejado encontra-se na folha da árvore (**nas pontas**) ou não existe.  **$O(\log_2 N)$**
- **Melhor Caso:** quando o elemento procurado corresponde a exatamente o elemento do meio do vetor (raiz da árvore).  **$\Omega(1)$**
- **Caso Médio:** quando o dado desejado encontra-se próximo do “meio” da árvore.  **$\theta(\log_2 N)$**
- Lembrando que  **$\log_2 N = e \Rightarrow 2^e = N$**

# Complexidade da Busca Binária

- Para um vetor de 8, é necessário apenas 3 comparações para se encontrar a chave no pior caso:
- Exemplo: localizar a chave **2** no vetor:
  - 11 21 34 39 41 45 89 98
  - **Comparação 1: 11 21 34 39 41 45 89 98**
  - **Comparação 2: 11 21 34 | 39 41 45 89 98**
  - **Comparação 3: 11 | 21 34 39 41 45 89 98**

# Complexidade da Busca Binária

- Exemplo: localizar o valor 0 (zero)





# Qual das duas buscas é melhor?



- Para uma lista com  $N = 1000$ , o algoritmo de pesquisa **sequencial** irá executar **1000** comparações no pior caso, e cerca de **500** operações no caso médio;
- Por sua vez, o algoritmo de pesquisa **binária** irá executar **10** comparações no pior caso, para o mesmo  $N$ . ( $\log_2 1000 \approx 10$ )
  - O logaritmo de base 2 aparece porque divide-se o intervalo de busca pela metade: 1000, 500, 250, 125, 63, 32, 16, 8, 4, 2, 1 (**10 divisões**)





# Qual das duas buscas é melhor?



- O algoritmo de pesquisa binária assume que a lista está **ordenada**
  - Ordenar uma lista também tem um **custo**, geralmente **superior** ao custo da pesquisa sequencial.
- Se for para fazer uma só pesquisa, **não vale à pena ordenar a lista**
- Por outro lado, se pretende-se fazer muitas pesquisas, o esforço da ordenação **pode valer a pena**

# EXERCÍCIOS

# Exercício

1. Faça um programa em python que realize a pesquisa sequencial em uma lista de números inteiros. Peça para o programa ler os números até que o valor -999 seja digitado. Em seguida o programa deve pedir para o usuário digitar o número a ser procurado na lista e efetuar a busca sequencial, informando se o mesmo foi ou não encontrado e se foi, em qual posição da lista este se encontra.
2. Refaça o programa anterior para usar o algoritmo de busca binária.

# Exercício

3. Implementar a busca e remoção da agenda, buscando por nome e telefone. Implementar a busca de duas formas (sequencial ou binária)

# Exercício

Considere o vetor com 11 elementos abaixo e diga quantas **comparações de igualdade** realizam os algoritmos de **Busca Linear** e **Busca Binária**, na tentativa de se encontrar no vetor os valores:

- a) 3
- b) 25
- c) 70

1	2	7	15	23	56	57	58	70	72	78
---	---	---	----	----	----	----	----	----	----	----

- Cormen, Thomas H. et. al. Algoritmos: Teoria e Prática. Editora Campus, 2002.
  
- Ziviani, Nivio. Projeto de Algoritmos. Editora Nova Fronteira, 2004.
  
- Complexidade (Prof. Jones Albuquerque)
  - [http://www.cin.ufpe.br/~joa/menu\\_options/school/cursos/ppd/aulas/complexidade.pdf](http://www.cin.ufpe.br/~joa/menu_options/school/cursos/ppd/aulas/complexidade.pdf)



**UFRPE**  
Universidade  
Federal Rural  
de Pernambuco

