

# Relatório – Atividade Prática 01:

## Algoritmos de Busca + Recursão

Universidade Federal do Rio Grande do Norte – Instituto Metrópole Digital

**Disciplina:** IMD0029 – Estruturas de Dados Básicas I

**Professor:** João Guilherme

**Aluno:** Lucas Toshio Nascimento da Silva

**Matrícula:** 20240001500

**Natal/RN – Setembro 2025**

---

## Introdução

A atividade teve como objetivo exercitar a implementação de algoritmos de busca estudados na disciplina, assim como o uso de funções recursivas.

Foram implementados:

- **Busca Binária** (para encontrar a primeira versão defeituosa de um software.)
- **Busca Sequencial Ordenada e contagem de especialidades distintas.**
- **Função Recursiva** para contar a ocorrência de caracteres em uma string.

Além das implementações, testes automatizados foram realizados para validar os resultados e uma análise teórica sobre complexidade, ordenação, recursão e escolha de algoritmos em cenários reais.

---

## Implementação Prática

### Funções Implementadas

- `busca_binaria(int n)`: encontra a primeira versão defeituosa utilizando o conceito da API `isBadVersion`.

- `busca_seq_ordenada(int arr[], int n, int alvo)`: busca linear em vetor ordenado.
- `conta_especialidades_distintas(int arr[], int n)`: conta quantas especialidades únicas existem, utilizando a função de busca/sequência.
- `recursao(const std::string &str, char alvo)`: conta recursivamente quantas vezes um caractere aparece em uma string.

## Resultados dos Testes

Os testes foram organizados no arquivo `test_algorithms.cpp` e executados via comando através do VSCode (terminal).

A execução retornou os seguintes resultados:

- Conta especialidades distintas → OK (Resultado: 3, Esperado: 3)
- Busca binária (isBadVersion) → OK (Resultado: 3, Esperado: 3)
- Busca sequencial ordenada → OK (Resultado: 2, Esperado: 2)
- Recursiva → OK (Resultado: 3, Esperado: 3)

The screenshot shows a VS Code editor with a C++ file named `test_algorithms.cpp` open. The file contains test cases for various algorithms. The terminal at the bottom shows the output of running the program, which matches the expected results listed in the text above.

```

11 // Teste para bubble sort e conta_especialidades_distintas
12 int arr1[] = {4, 2, 1, 4, 2, 1};
13 int n1 = 6;
14 bubble_sort(arr1, n1); // Ordena o array
15 RUN_TEST("conta especialidades distintas", conta_especialidades_distintas(arr1, n1), 3);
16
17 // Teste para busca binaria
18 int arr2[] = {1, 2, 3, 4, 5};
19 int n2 = 5;
20 bad = 4; // Define a versão defeituosa
21 RUN_TEST("Busca binária (isBadVersion)", busca_binaria(arr2, n2, 4), 3);
22
23 // Teste para busca sequencial ordenada
24 int arr3[] = {1, 2, 3, 4, 5};
25 int n3 = 5;
26 bad = 4; // Define a versão defeituosa
27 RUN_TEST("Busca sequencial ordenada", busca_seq_ordenada(arr3, n3, 4), 2);
28
29 // Teste para recursao
30 int arr4[] = {1, 2, 3, 4, 5};
31 int n4 = 5;
32 bad = 4; // Define a versão defeituosa
33 RUN_TEST("Recursiva", recursao(arr4, '4'), 3);
34
35 return 0;
36 }
37
38 // Variável global para simular a API isBadVersion
39 int bad = 4;
40
41 int main() {
42     // Teste para bubble sort e conta_especialidades_distintas
43     int arr1[] = {4, 2, 1, 4, 2, 1};
44     int n1 = 6;
45     bubble_sort(arr1, n1); // Ordena o array
46     RUN_TEST("conta especialidades distintas", conta_especialidades_distintas(arr1, n1), 3);
47
48     // Teste para busca binaria
49     int arr2[] = {1, 2, 3, 4, 5};
50     int n2 = 5;
51     bad = 4; // Define a versão defeituosa
52     RUN_TEST("Busca binária (isBadVersion)", busca_binaria(arr2, n2, 4), 3);
53
54     // Teste para busca sequencial ordenada
55     int arr3[] = {1, 2, 3, 4, 5};
56     int n3 = 5;
57     bad = 4; // Define a versão defeituosa
58     RUN_TEST("Busca sequencial ordenada", busca_seq_ordenada(arr3, n3, 4), 2);
59
60     // Teste para recursao
61     int arr4[] = {1, 2, 3, 4, 5};
62     int n4 = 5;
63     bad = 4; // Define a versão defeituosa
64     RUN_TEST("Recursiva", recursao(arr4, '4'), 3);
65
66     return 0;
67 }

```

Terminal Output:

```

PS C:\Users\toshi\Desktop\aulas bti\EDB\Atividade 1\atividade_1> .\test_algorithms.exe
Conta especialidades distintas : OK (Resultado: 3, Esperado: 3)
Busca binária (isBadVersion) : OK (Resultado: 3, Esperado: 3)
Busca sequencial ordenada : OK (Resultado: 2, Esperado: 2)
Recursiva : OK (Resultado: 3, Esperado: 3)
PS C:\Users\toshi\Desktop\aulas bti\EDB\Atividade 1\atividade_1>

```

# Questões Teóricas

## Questão 6.1 — Complexidade de Algoritmos de Busca

A **busca sequencial** possui complexidade  $O(n)$ , pois percorre todos os elementos até encontrar o alvo ou até acabar. É indicada em vetores pequenos ou quando os dados não estão ordenados.

A **busca binária** tem complexidade  $O(\log n)$ , mas exige que o vetor esteja ordenado previamente. É muito mais eficiente para grandes volumes de dados que permitem essa ordenação.

---

## Questão 6.2 — Impacto da Ordenação nos Algoritmos de Busca

Ordenar um vetor tem custo de  $O(n \log n)$ . Se houver apenas uma busca, pode não compensar ordenar antes. Mas se for realizar várias buscas no mesmo conjunto de dados, ordenar uma vez e depois usar a busca binária repetidamente compensa bastante. Essa abordagem reduz o tempo médio por busca quando há muitas consultas.

---

## Questão 6.3 — Recursão x Iteração

- **Recursão:** abordagem elegante para problemas naturalmente recursivos. É mais simples de implementar em alguns casos, mas consome mais memória de pilha e pode ter overhead de chamadas.
- **Iteração:** geralmente mais eficiente em memória e às vezes em tempo (menos overhead). Boa quando o problema pode ser resolvido com laços simples.

Exemplo recursivo e iterativo para fatorial demonstram isso bem.

---

## Questão 6.4 — Análise de um Cenário Real

Em uma empresa de logística, pacotes chegam de forma contínua e desordenada. Se o sistema precisar responder imediatamente, usar busca sequencial pode ser melhor.

Se for possível armazenar os pacotes numa lista ordenada, ou reorganizar periodicamente, então a busca binária se torna uma alternativa mais eficiente para as consultas.

A decisão depende do trade-compensação entre o custo de reordenação e a frequência de buscas.

---

## Conclusão

A atividade permitiu reforçar os conceitos de busca (sequencial, ordenada, binária) e recursão. Escolher o algoritmo certo dependendo da situação real, da estrutura de dados e da necessidade de desempenho pode fazer muita diferença. O teste automatizado foi importante para confirmar a correção das funções. nunca tinha feito um teste automatizado antes foi um pouco desafiador...

---

## Repositório

 Link para o repositório: [https://github.com/Toshiosam/Atividade\\_1](https://github.com/Toshiosam/Atividade_1) [GitHub](#)