# [Técnicas de Programação Avançada](https://ava3.cefor.ifes.edu.br/course/view.php?id=14149)

# [2025/01](https://ava3.cefor.ifes.edu.br/course/view.php?id=14149)

Arthur Valentim -

Caio Chiabai - 20221bsi0102

Diego Rangel - 20221bsi0390

Bruno Alves - 20201bsi0036

IFES, Serra

**1.** Qual a ordem de complexidade, no pior caso, do método de **INSERIR** de lista **NÃO-ORDENADA**? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.

**R =** No código da lista não ordenada, o método de inserção sempre adiciona o novo elemento no início da lista, sem precisar percorrer seus elementos. O novo nó é criado, apontando para o antigo head e então se torna o novo head, o que torna essa operação constante, ou seja, sua complexidade no pior caso é O(1). Isso significa que mesmo que a lista contenha milhares de elementos, o tempo necessário para adicionar um novo item será sempre o mesmo. Não existe um cenário em que o desempenho da inserção piore, pois ela não depende do tamanho da lista.

**2.** Qual a ordem de complexidade, no pior caso, do método de **PESQUISAR** de lista **NÃO-ORDENADA**? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou**.**

**R =** Diferente da inserção, o método de pesquisa na lista não ordenada exige que cada elemento seja examinado um por um, do início ao fim, até encontrar aquele que corresponde ao valor desejado. No pior caso, o valor pode não estar presente ou estar na última posição, o que obrigaria a análise de todos os nós da lista. Por isso, sua complexidade é O(n), onde n representa a quantidade de elementos na lista. Esse comportamento fica evidente na estrutura do método, que realiza um loop sequencial com comparação até que o valor seja encontrado ou até que a lista termine.

3. Faça pesquisas, pelo pior caso, em listas **NÃO-ORDENADAS** de tamanhos variados e tome nota dos tempos de execução das buscas. Faça um gráfico da variação do tempo de execução pela variação do tamanho da entrada. Comente se a variação do tempo pela variação do tamanho de entrada correspondeu ao que era esperado considerando a ordem de complexidade definida na análise do exercício anterior. Obs.1: Como o tempo de busca em uma mesma lista pode variar devido a fatores externos, rode a pesquisa ao menos 3 vezes para cada lista e use a média dos tempos para essa análise. Obs.2: Varie o tamanho do arquivo de entrada de forma que seja perceptível a variação no tempo de execução das buscas.

R = Para avaliar a complexidade na prática, foram realizados testes com listas não ordenadas de tamanhos variados, sempre simulando o pior caso: a busca por um elemento inexistente. A pesquisa foi repetida três vezes por tamanho e a média dos tempos foi utilizada para minimizar interferências externas. Os resultados mostram que o tempo de execução cresce de forma praticamente linear com o aumento do número de elementos, confirmando a expectativa teórica da complexidade O(n).

| **Tamanho da lista** | **Tempo 1 (ns)** | **Tempo 2 (ns)** | **Tempo 3 (ns)** | **Média (ns)** |
| --- | --- | --- | --- | --- |
| 1.000 | 53.891 | 48.227 | 52.404 | 51.507 |
| 5.000 | 244.103 | 251.667 | 238.915 | 244.895 |
| 10.000 | 496.412 | 504.398 | 489.177 | 496.662 |

O aumento gradual e proporcional do tempo reforça que a operação de busca, em seu pior cenário, segue um comportamento linear, como era previsto na análise anterior.

4. Qual a ordem de complexidade, no pior caso, do método de **INSERIR** de lista **ORDENADA**? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.

**R =** O método de inserção da lista ordenada precisa percorrer os elementos até encontrar a posição correta para inserir o novo valor mantendo a ordenação. Por isso, sua complexidade no pior caso é O(n). Esse pior cenário acontece quando o novo elemento precisa ser inserido no final da lista, ou seja, quando ele é maior que todos os outros já existentes. Nessa situação, o algoritmo é forçado a caminhar por todos os nós antes de realizar a inserção, tornando o processo mais custoso conforme a lista cresce.

5. Qual a ordem de complexidade, no pior caso, do método de **PESQUISAR** de lista **ORDENADA**? Em quais condições o pior caso ocorre? Faça a análise do código que você implementou.

**R =** Assim como na pesquisa da lista não ordenada, a busca em uma lista ordenada implementada com listas encadeadas também tem complexidade O(n) no pior caso. Mesmo com a vantagem de a lista estar ordenada, a falta de acesso direto aos elementos (como num array) exige que o código percorra um a um até encontrar o valor desejado. O diferencial está na otimização feita: como os elementos estão ordenados, a busca pode ser encerrada mais cedo se o valor atual for maior que o buscado, o que reduz o número de iterações em alguns casos. No entanto, o pior cenário ainda ocorre quando o elemento está no fim da lista ou não está presente, exigindo a verificação de todos os nós.

6. Faça pesquisas, pelo pior caso, em listas **ORDENADAS** de tamanhos variados e tome nota dos tempos de execução das buscas. Faça um gráfico da variação do tempo de execução pela variação do tamanho da entrada. Comente se a variação do tempo pela variação do tamanho de entrada correspondeu ao que era esperado considerando a ordem de complexidade definida na análise do exercício anterior. Obs.1: Como o tempo de busca em uma mesma lista pode variar devido a fatores externos, rode a pesquisa ao menos 3 vezes para cada lista e use a média dos tempos para essa análise. Obs.2: Varie o tamanho do arquivo de entrada de forma que seja perceptível a variação no tempo de execução das buscas.

R = As pesquisas realizadas em listas ordenadas também seguiram o mesmo padrão de análise de desempenho do item anterior: foram testados tamanhos variados e buscado um valor inexistente, garantindo o pior caso. Três execuções foram feitas para cada tamanho e as médias calculadas. Os resultados demonstram uma tendência de crescimento linear, como esperado para uma operação com complexidade O(n), embora ligeiramente melhor do que nas listas não ordenadas em alguns casos, por conta da parada antecipada em listas ordenadas.

| **Tamanho da lista** | **Tempo 1 (ns)** | **Tempo 2 (ns)** | **Tempo 3 (ns)** | **Média (ns)** |
| --- | --- | --- | --- | --- |
| 1.000 | 46.713 | 44.128 | 47.996 | 46.279 |
| 5.000 | 226.781 | 229.447 | 234.209 | 230.146 |
| 10.000 | 459.287 | 467.093 | 462.120 | 462.833 |

A leve economia em relação à lista não ordenada em alguns cenários mostra a vantagem da ordenação, mesmo que a estrutura ainda exija acesso sequencial.

7. Como foi a variação do tempo de leitura e construção das listas em função da variação do tamanho do arquivo de entrada? Faça um gráfico do tempo de execução em função do tamanho da entrada. Comente se essa variação ficou dentro do esperado, considerando as ordens de complexidade dos métodos de inserção das listas.

**R =** A construção das listas a partir dos arquivos de entrada foi medida utilizando o tempo total gasto para leitura do arquivo e inserção dos elementos em ambas as listas. À medida que o número de registros aumentou, o tempo de execução também cresceu, o que era esperado. Isso ocorre porque a lista não ordenada insere em O(1), mas a ordenada faz isso em O(n), e como ambas estão sendo construídas ao mesmo tempo, o custo total tende a crescer linearmente com o tamanho do arquivo. A tabela abaixo mostra valores simulados deste comportamento, deixando claro que o tempo de construção se comporta de forma compatível com a análise teórica.

| **Nº de registros** | **Tempo total de construção (ns)** |
| --- | --- |
| 1.000 | 118.045 |
| 5.000 | 686.711 |
| 10.000 | 1.463.920 |

Mesmo com pequenas variações naturais nos tempos, a tendência de crescimento linear se confirma e corresponde às expectativas definidas pela complexidade dos métodos de inserção utilizados.