josif 2025
17ª Jornada Científica e Tecnológica
14º Simpósio de Pós-Graduação
do IFSULDEMINAS
ISSN: 2319-0124
REALIZAÇÃO
INSTITUTO FEDERAL Sul de Minas Gerais
APOIO
CNPq
FAPEMIG

AVALIAÇÃO DE DESEMPENHO EM LINGUAGENS DE PROGRAMAÇÃO: Medindo o Tempo de Execução e Uso de Memória

João P. ALVES[1]; Paulo C. dos SANTOS[2];

RESUMO

Este artigo realiza uma análise comparativa do desempenho de quatro linguagens de programação populares — Python, C++, Java e Go — em quatro algoritmos: QuickSort, MergeSort, verificação de números primos e cálculo de Fibonacci recursivo. A pesquisa busca avaliar o tempo de execução, o uso de memória e a estabilidade dos algoritmos de ordenação. Os resultados indicaram que C++ apresentou o melhor desempenho em velocidade e uso de memória, seguido por Go, Java e Python. A análise do MergeSort confirmou sua estabilidade em contraste com o QuickSort. A pesquisa conclui que, embora C++ seja mais rápido, a escolha da linguagem ideal depende de um balanço entre desempenho, produtividade e requisitos específicos do projeto, como a necessidade de concorrência, portabilidade ou estabilidade algorítmica.

Palavras-chave: Eficiência; Benchmarking; Algoritmo.

1. INTRODUÇÃO

O desempenho de linguagens de programação é um tema crucial em ciência da computação, especialmente na escolha da tecnologia ideal para sistemas de alto desempenho. O crescente número de linguagens e suas variações em desempenho geram dúvidas entre desenvolvedores sobre qual usar em diferentes cenários (SOUSA et al., 2023). Comparações sobre tempo de execução, uso de memória e eficiência ajudam a escolher a melhor ferramenta para cada tarefa computacional, impactando diretamente a experiência do usuário e a escalabilidade dos sistemas (FONSECA et al., 2017).

A eficiência em termos de tempo de execução e uso de memória é essencial, especialmente em ambientes com recursos limitados, como dispositivos móveis ou sistemas embarcados. A análise de linguagens populares como Python, C++, Go e Java em tarefas como algoritmos de ordenação e busca fornece uma visão clara das capacidades de cada uma. Este estudo visa comparar o desempenho dessas linguagens, levando em consideração sua eficiência e o comportamento de algoritmos em diferentes cenários.

Além do desempenho, a escolha da linguagem também influencia a manutenção do código, a paralelização, a interoperabilidade e o suporte da comunidade de desenvolvedores (ZAPALOWSKI et al., 2011). Estudos como o de NANZ et al. (2014) analisam linguagens em plataformas como o Rosetta Code, fornecendo insights sobre concisão e falhas. Com base nisso, este artigo realiza uma análise de desempenho de Python, C++, Go e Java em tarefas comuns.

[1]Discente de Bacharelado em Ciência da Computação, IFSULDEMINAS – *Campus* Muzambinho. E-mail: 12201000124@muz.ifsuldeminas.edu.br.
[2]Orientador, IFSULDEMINAS – *Campus* Muzambinho. E-mail: paulo.santos@muz.ifsuldeminas.edu.br.

2. FUNDAMENTAÇÃO TEÓRICA

O desempenho das linguagens de programação varia conforme sua abordagem ao gerenciamento de memória e processamento. Linguagens compiladas, como C++ e Go, tendem a ser mais rápidas, pois traduzem o código diretamente para a máquina, evitando a sobrecarga de uma máquina virtual, como ocorre em linguagens interpretadas como Python (SOUSA et al., 2023). Essa vantagem é evidente em tarefas computacionais intensivas, onde a eficiência no uso de recursos impacta diretamente o tempo de execução (FONSECA et al., 2017).

Por outro lado, linguagens como Python e Java oferecem maior produtividade e facilidade de uso. Python, amplamente adotado para prototipagem rápida e ciência de dados, sofre com overhead devido ao garbage collection e à sua execução interpretada (ZAPALOWSKI et al., 2011). Java, executado na JVM, proporciona um equilíbrio entre desempenho e portabilidade, mas não atinge a eficiência de linguagens compiladas em tarefas computacionais mais pesadas (NANZ et al., 2014).

Estudos prévios, como o de NANZ et al. (2014), mostram que a implementação dos algoritmos e a escolha das estruturas de dados impactam significativamente o desempenho. Algumas linguagens possuem otimizações específicas para listas, árvores binárias e outras estruturas fundamentais para algoritmos de ordenação e busca, tornando-as mais eficientes em determinados cenários (SILVA et al., 2024). Assim, a escolha da linguagem deve considerar não só o tempo de execução, mas também a necessidade de produtividade e facilidade de manutenção.

3. MATERIAL E MÉTODOS

Este estudo visa comparar o desempenho de quatro linguagens de programação (Python, C++, Java e Go) em quatro algoritmos: ordenação de números aleatórios com QuickSort, verificação de números primos, cálculo do 40º termo da sequência de Fibonacci com abordagem recursiva, e a ordenação de uma lista de objetos com QuickSort e MergeSort para avaliar a estabilidade.

Os testes foram realizados em um ambiente controlado, utilizando um computador com processador Intel Core i5 de 11ª geração, 16 GB de memória RAM e o sistema operacional Windows 11. A IDE utilizada foi o Visual Studio Code. As versões das linguagens foram: Python 3.12.8, C++ 14.1, Java 21.0.6 e Go 1.23.6.

Os três primeiros testes seguiram a metodologia original:

- QuickSort: Ordenação de 1.000.000 de números aleatórios.
- Números Primos: Verificação da primalidade de 1.000.000 de números inteiros.
- Fibonacci Recursivo: Cálculo do 40º termo da sequência.

O quarto teste foi adicionado para analisar a estabilidade dos algoritmos de ordenação. Para

isso, foi criada uma lista com 1.000.000 de objetos “Produto”, cada um com um ID numérico e um preço. A lista foi ordenada com base no atributo preço, utilizando tanto o QuickSort quanto o MergeSort. O MergeSort foi incluído por ser um algoritmo inerentemente estável, o que permite uma análise comparativa direta sobre como cada algoritmo preserva a ordem relativa de elementos com chaves de ordenação iguais.

Para garantir a precisão, cada teste foi executado 10 vezes, e a média do tempo de execução e do consumo de memória foi calculada.

4. RESULTADOS E DISCUSSÃO

Os testes revelaram diferenças significativas de desempenho. No teste de QuickSort com números, C++ foi a mais eficiente (0,34s), seguida por Go (0,73s), Java (1,47s) e Python (5,67s). Resultados semelhantes foram observados nos demais testes, conforme as Tabelas 1 e 2.

Teste (Algoritmo)	Python (s)	C++ (s)	Java (s)	Go (s)
Quicksort	5,67	0,34	1,47	0,73
Números primos	4,10	0,24	0,22	0,29
Fibonacci Recursivo	18,02	1,26	0,66	0,85

Tabela 1 - Médias dos tempos de execução (em segundos) para cada linguagem e teste.

Teste (Algoritmo)	Python (s)	C++ (s)	Java (s)	Go (s)
Quicksort	85,42	15,73	40,25	25,19
Números primos	60,81	10,45	30,51	20,37
Fibonacci Recursivo	120,34	20,82	50,78	35,92

Tabela 2 - Médias de consumo de memória (MB) para cada linguagem e teste.

A superioridade de C++ e Go deve-se à sua natureza compilada, que gera código de máquina otimizado. C++ se beneficia ainda mais por permitir um gerenciamento manual de memória, eliminando o overhead de um garbage collector. Java, rodando na JVM, possui um desempenho intermediário, pois o compilador Just-In-Time (JIT) otimiza o bytecode em tempo de execução, mas a máquina virtual e o garbage collector ainda consomem recursos. Python apresentou os piores resultados devido à sua natureza interpretada e tipagem dinâmica, que exigem mais processamento a cada instrução e resultam em um gerenciamento de memória menos eficiente para tarefas computacionalmente intensivas.

No teste de ordenação de objetos (Tabela 3), o MergeSort, apesar de ligeiramente mais lento que o QuickSort em alguns casos, demonstrou sua relevância ao garantir a estabilidade — a preservação da ordem original dos elementos com o mesmo preço. O QuickSort, em sua implementação padrão, não oferece essa garantia. Este teste evidencia que, para além da velocidade, a escolha do algoritmo deve considerar propriedades como a estabilidade, crucial em aplicações que manipulam dados complexos.

Teste (Ordenação de Objetos)	Python (s)	C++ (s)	Java (s)	Go (s)
Quicksort (instável)	7,15	0,52	1,95	0,98
Merge Sort (estável)	8,02	0,61	2,10	1,05

Tabela 3 - Tempo de execução (s) na ordenação de 1.000.000 de objetos.

É importante reconhecer as limitações desta pesquisa. Os resultados refletem uma configuração de hardware/software específica e podem variar em outros ambientes. Foram usadas implementações padrão dos algoritmos, sem bibliotecas otimizadas. A análise focou em tarefas de uso intensivo da CPU, excluindo operações de I/O e cenários de concorrência avançada, que poderiam favorecer linguagens como Go.

5. CONCLUSÃO

Este estudo confirmou a superioridade de C++ em desempenho bruto, tornando-a ideal para aplicações críticas como jogos ou computação de alto desempenho. Contudo, a escolha da linguagem deve ser pragmática e alinhada aos objetivos do projeto. Go destaca-se pela eficiência em microsserviços; Java, pela robustez em sistemas corporativos; e Python, pela produtividade em ciência de dados e desenvolvimento ágil. A decisão final deve, portanto, ponderar as necessidades de performance, escalabilidade e tempo de desenvolvimento, reconhecendo que não há uma "melhor linguagem", mas sim a mais adequada para cada problema.

REFERÊNCIAS

FONSECA, B., D.; RODRIGUES, B., W.; CRUZ, U., R., J; PERINI, P., K., A. Comparativo de Desempenho na execução entre Linguagens de Programação. 2017. Disponível em: https://www.cc.faccamp.br/anaisdowcf/edicoes_anteriores/wcf2017/arquivos/03/paper_03.pdf. Acesso em: 21 jan. 2025.

ZAPALOWSKI, V. Análise quantitativa e comparativa de linguagens de programação. 2011. Disponível em: https://lume.ufrgs.br/handle/10183/31036. Acesso em: 21 jan. 2025.

SOUSA, C., D., R.; FLORIAN, F.; DALLILO, D., F. Uma análise de performance das linguagens de programação no processamento paralelo. 2023. Disponível em: https://www.nucleodoconhecimento.com.br/engenharia-da-computacao/linguagens-de-programacao . Acesso em: 21 jan. 2025.

SILVA, F., P., J.; BORGES, G., H., J.; FLORIAN, F. Análise comparativa de eficiência de estruturas de dados em diferentes linguagens de programação (Rust, C#, Java e Python 3). 2024. Disponível em: https://revistaft.com.br/analise-comparativa-de-eficiencia-de-estruturas-de-dados-em-diferentes-linguagens-de-programacao-rust-c-java-e-python-3/. Acesso em: 21 jan. 2025.

NANZ, S.; FURIA, A., C. A Comparative Study of Programming Languages in Rosetta Code. 2014. Disponível em: https://arxiv.org/abs/1409.0252. Acesso em: 21 jan. 2025.