**Instituto Tecnológico de Aeronáutica - ITA**
**CTC-12 - Projeto e Análise de Algoritmos**
**Aluno**: Ulisses Lopes da Silva

# Relatório do Laboratório 3 - Variações de Algoritmos de Ordenação

---

**Objetivo**: Implementar variações de algoritmos de ordenação, realizar comparações e entender os resultados.

## 1 *Quick Sort* × *Merge Sort* Iterativo × *Radix Sort* × *STL::Sort*

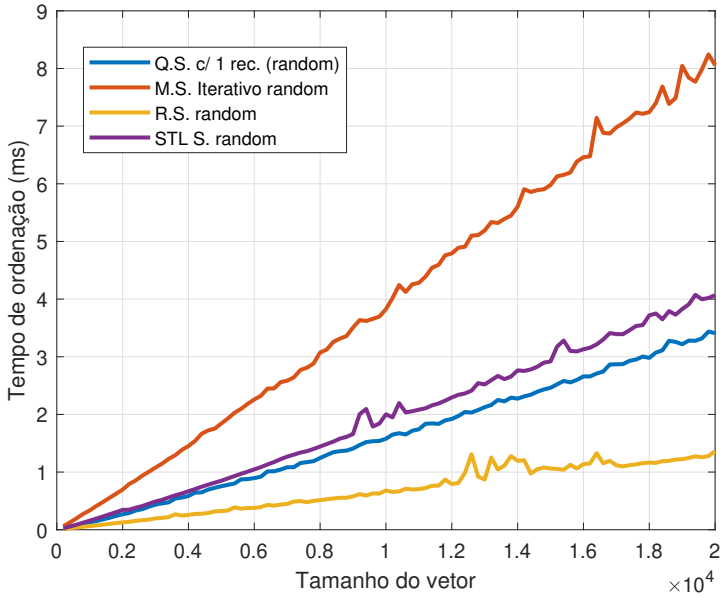


Fig. 1: Tempo de ordenação pelo tamanho do vetor

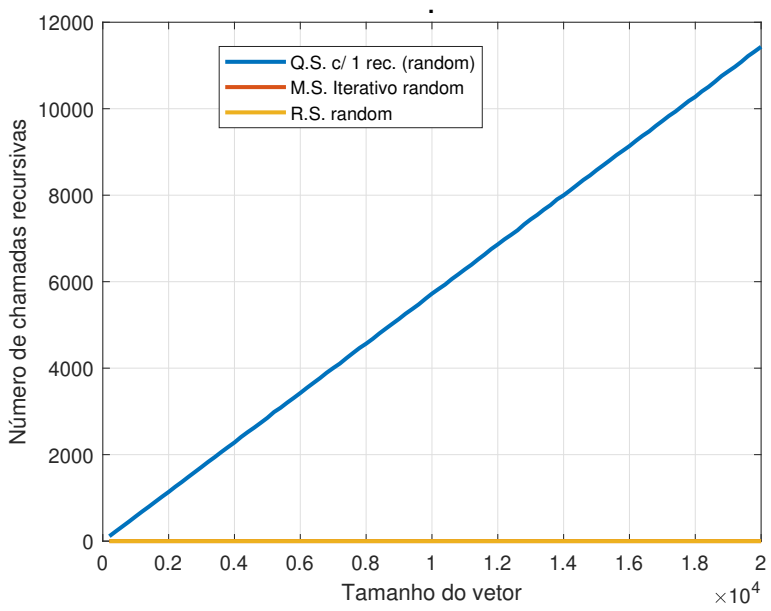


Fig. 2: Chamadas recursivas

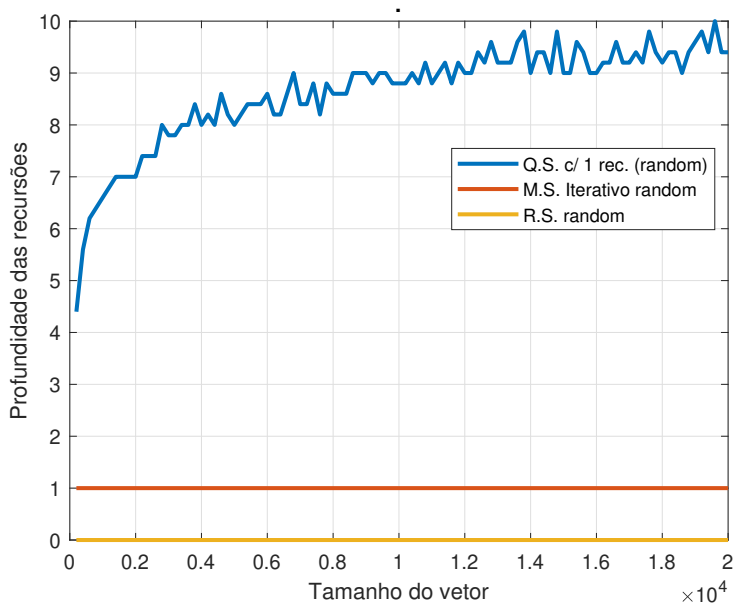


Fig. 3: Profundidade das recursões

Como se pôde verificar, o algoritmo Radix Sort apresentou o menor tempo de ordenação em relação aos demais, já que sua complexidade é da ordem de O(n). Não obstante, percebe-se que o modo iterativo do Merge Sort, para vetores muito grandes, apresentou um resultado pior que os demais. O Quick Sort, como esperado, apresentou o melhor tempo dentre os métodos

baseados em comparação, apesar de seu alto número de chamadas recursivas e da profundidade dessas chamadas, o que revela que, apesar de muito rápido, ele consome uma boa quantidade de memória para a pilha de execução.

## 2 *Merge Sort* Recursivo × *Merge Sort* Iterativo

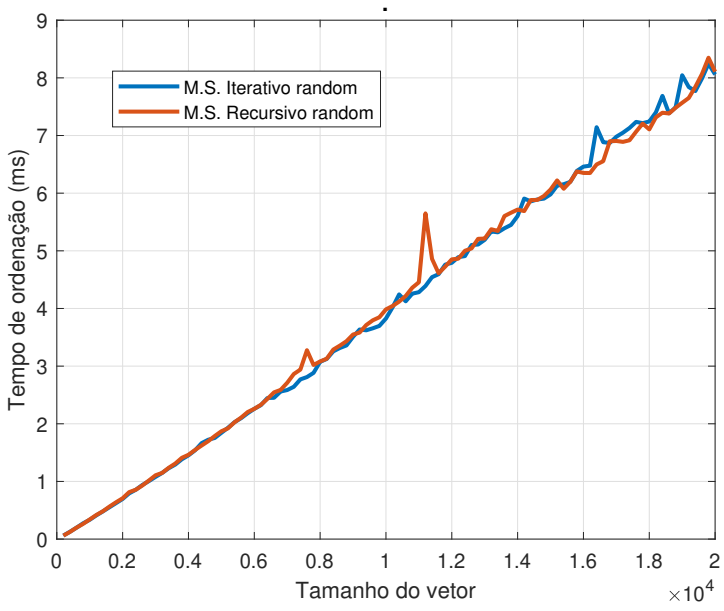


Fig. 4: Tempo de ordenação

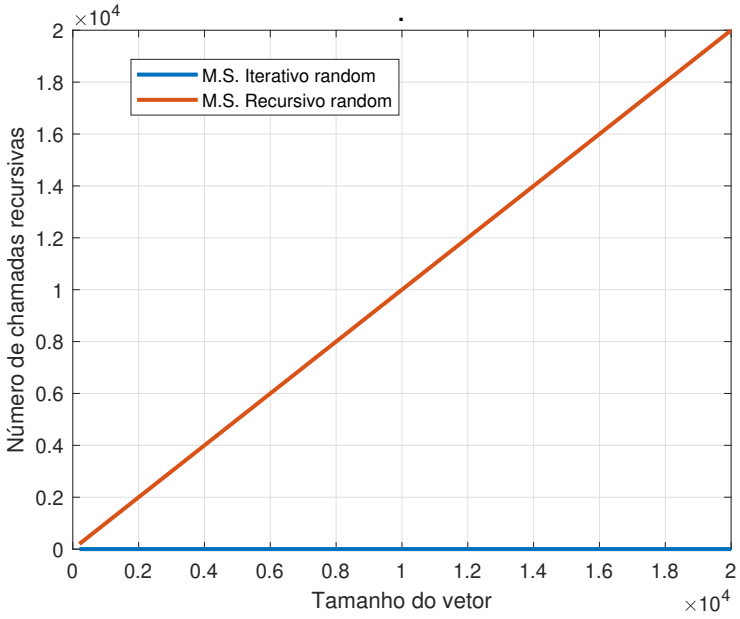


Fig. 5: Chamadas Recursivas

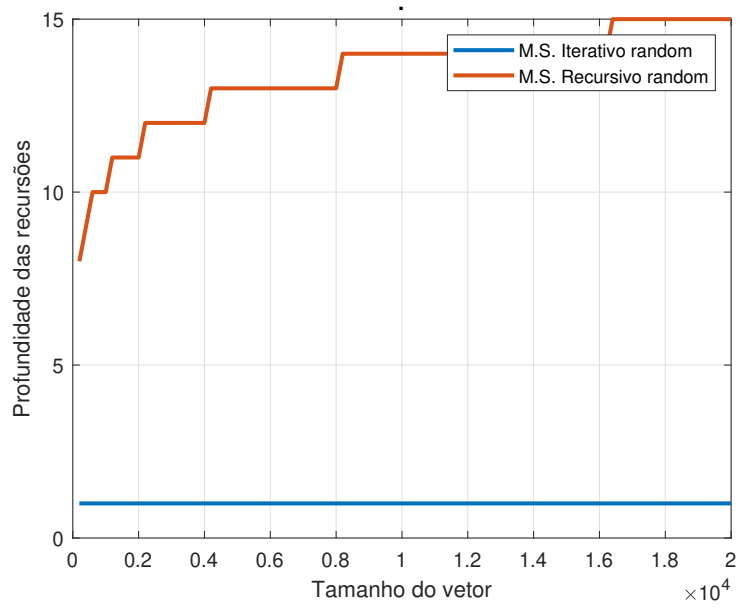


Fig. 6: Profundidade das recursões

O Merge Sort Iterativo mostrou-se levemente melhor que o Merge Sort Recursivo em alguns pontos do gráfico. todavia, em geral, eles tiveram praticamente o mesmo tempo de ordenação para as condições dos testes realizados. contudo, o Merge Sort Iterativo mostrou-se totalmente superior em relação ao consumo de memória na pilha de execução (pois é iterativo e não recursivo).

Assim, para uma ordenação em que se deseja economizar o gasto de memória paralela, o método iterativo se mostra bem melhor, apesar de também consumi-la para alocação das variáveis locais e/ou globais.

## 3 *Quick Sort* c/ 1 recursão × *Quick Sort* c/ 2 recursões

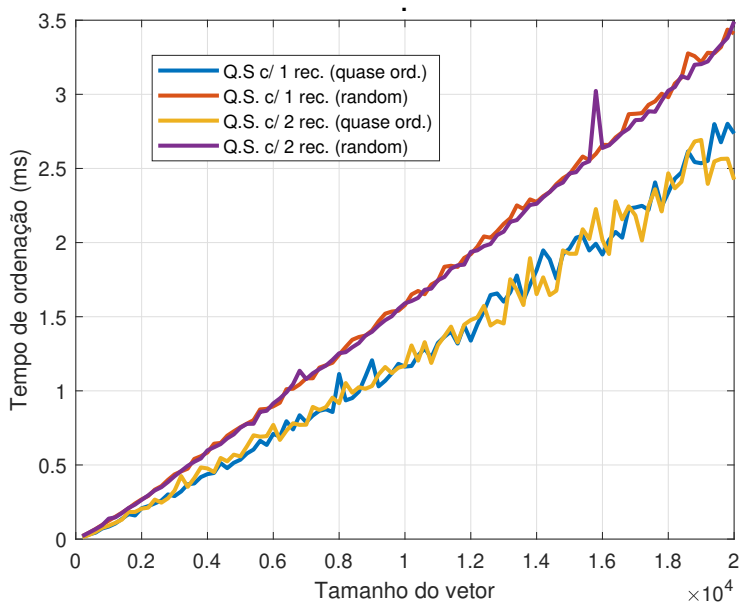


Fig. 7: Tempo de ordenação

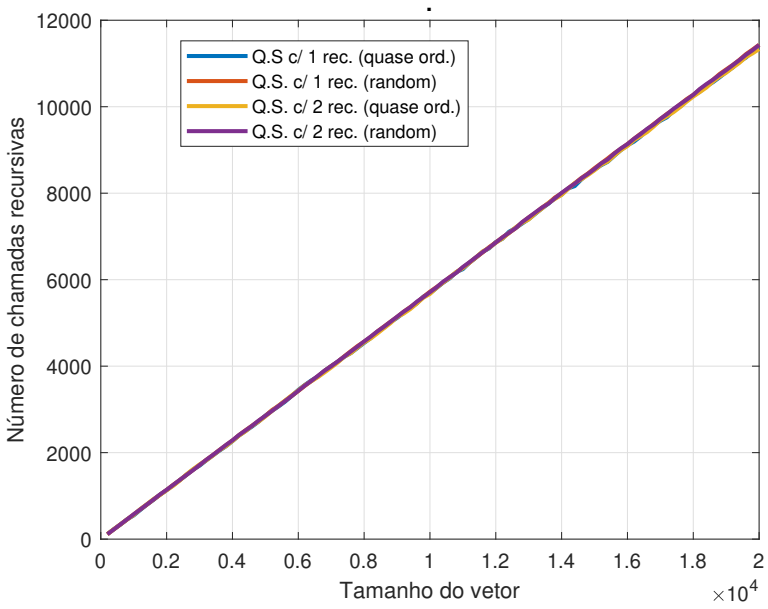


Fig. 8: Chamadas recursivas

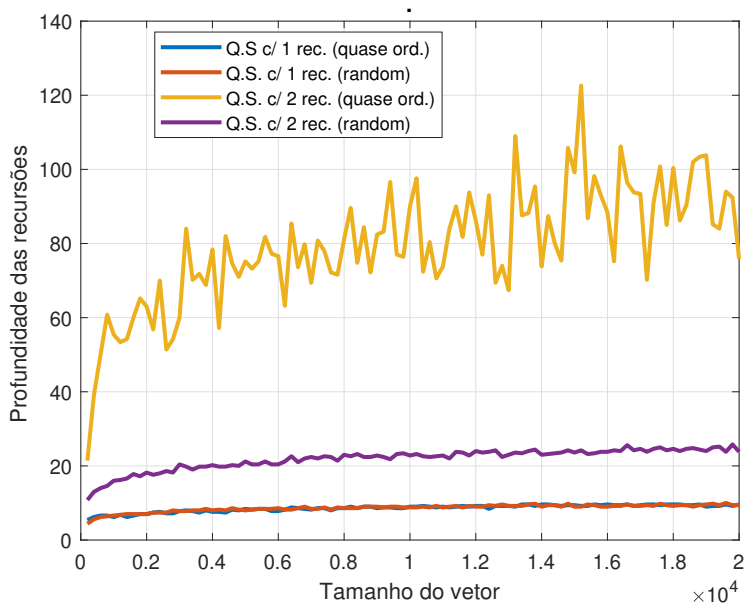


Fig. 9: Profundidade das recursões

O algoritmo Quick Sort com 1 recursão randômico teve tempo bem semelhante ao Quick Sorte com 2 recursões, igualmente randômico. Desse modo, é interessante observar que a escolha do tipo de algoritmo deverá ser feita com base em outros critérios, como a profundidade da pilha de execução, por exemplo. Semelhantemente, os algoritmos com 1 e 2 recursões, mas dessa

vez, quase ordenados, também gastaram tempos semelhantes. analisando o número de chamadas recursivas, há pouca diferença para os 4 métodos. Contudo, percebe-se, como esperado da teoria, que o algoritmo com 2 recursões e quase ordenado possui uma profundidade na pilha de execução bem maior, fato que foi comprovado quando chamamos recursivamente, pelo Quick Sort, um vetor ordenado ou quase ordenado. além disso, os gráficos da profundidade das chamadas recursivas é próximo de O(log(n)) para o Quick Sort com 2 recursões, mas quase linear para o Quick Sort com 1 recursão.

## 4 *Quick Sort* c/ mediana de 3 × *Quick Sort* c/ pivô fixo (ambos para vetores quase-ordenados

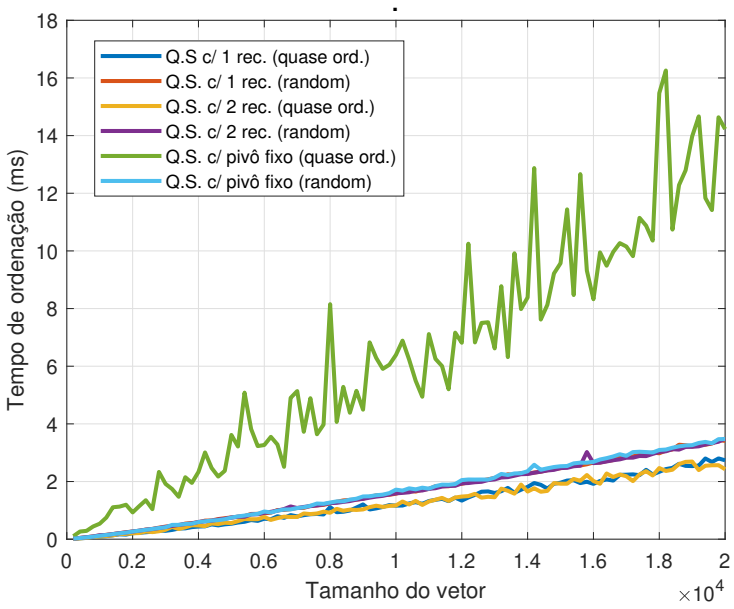


Fig. 10: Tempo de ordenação pelo tamanho do vetor

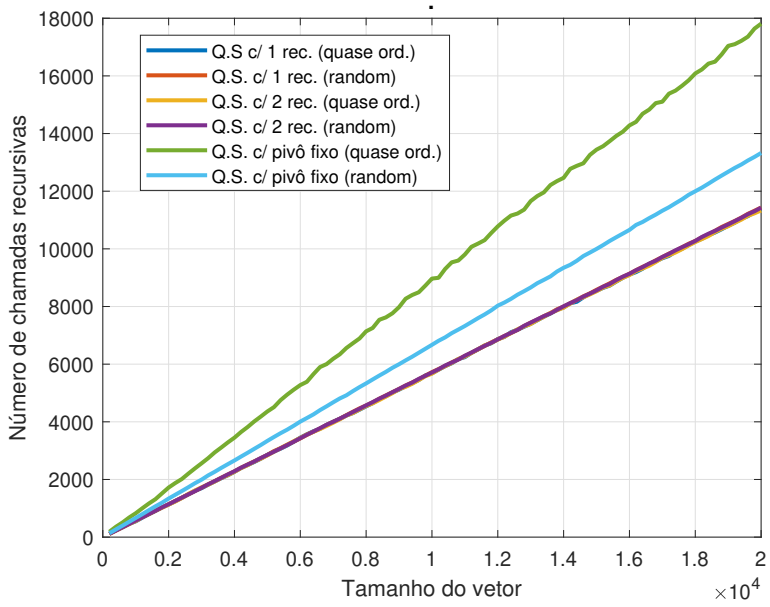


Fig. 11: Chamadas recursivas

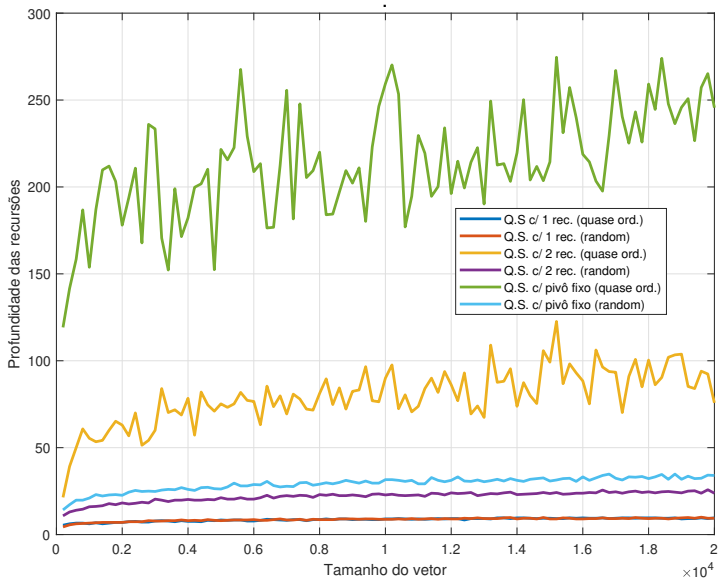


Fig. 12: Profundidade das recursões

A comparação entre os diferentes métodos do Quick sort mostram claramente que, conforme a teoria, o pivô fixado, apesar de ainda ser rápido, torna o Quick Sort bem menos eficiente, já que, em vetores quase ordenados, a pilha de recursão cresce muito, e, igualmente, o tempo de ordenação. O gráfico mostra sensivelmente a diferença entre o Quick Sort com pivô fixo e

quase ordenado pros demais. Por outro lado, num vetor randômico, mesmo o pivô fixo ainda é bastante rápido, o que confirma que, atualmente, o Quick Sort é o algoritmo mais rápido nos casos médios, dentre os algoritmos baseados em comparação. O gráfico com a profundidade das recursões também mostra que, apesar de rápido, o quick Sort pode consumir uma boa quanitdade de memória, principalmente nos piores casos (vetores quase ordenados e ordenados).

## 5 Caso Real

A implementação do Quicksort em uma biblioteca de propósito geral provavelmente deveria levar em consideração o pior caso, que é quadrático. Contudo, é importante lembrar que o pior caso do Quicksort ocorre ***raramente***, especialmente se forem usadas técnicas para escolher um bom pivô, como o pivô da media de 3, também visto neste lab.

Entretanto, se os dados já vêm mais ou menos ordenados, como mencionado no problema, em toda implementação em que for ser usado, o Quicksort pode, de fato, ter um desempenho próximo ao quadrático, já que apresenta uma performance ruim em dados já ordenados se o pivô for escolhido como o primeiro ou o último elemento (forma mais comum).

Sendo assim, acredito que a implementação do Quicksort poderia ser considerada em uma biblioteca de propósito geral, considerando o custo-benefício das alternativas de implementação. Embora o Quicksort possa ter um desempenho ruim em dados quase ordenados, ele é muito eficiente em muitos outros casos, com uma complexidade média de tempo O(n log n). Além disso, o Quicksort é um algoritmo que não requer memória adicional significativa, ao contrário de outros algoritmos de ordenação, como o Mergesort.

Não obstante, a dificuldade adicional de implementação e testagem deve ser levada em conta. Embora o Quicksort seja um algoritmo complexo, ele é bem compreendido e amplamente utilizado, o que significa que existem muitos recursos disponíveis para auxiliar na sua implementação e teste.

Portanto, embora haja ressalvas, o Quicksort ainda seria uma escolha válida para uma biblioteca de propósito geral, desde que sejam tomadas precauções para lidar com o pior caso e dados quase ordenados, tais como escolha do pivô pela mediana de 3, etc. No entanto, é primordial considerar as necessidades específicas do projeto e os tipos de dados que serão mais comumente classificados, ao tomar essa decisão.

## 6 Referências Bibliográficas

[1]. **CORMEN**, Thomas M. Algoritmos - Teoria e Prática. 3a edição, Editora ELSEVIER, Rio de Janeiro, RJ, 2012.