Relatório de CES-12

Lab 03 - Variações de Algoritmos de Ordenação

Aluno: Fernando de Moraes Rodrigues

Turma Comp-22

Professor Luiz Gustavo Bizarro Mirisola

Instituto Tecnológico de Aeronáutica – ITA

Maio 2020

Comparações

1) Tempo de execução QuickSort X MergeSort X RadixSort

Pela figura 1 a seguir, percebemos que os tempos de execução crescem na ordem QuickSort, RadixSort e MergeSort, sendo todos Θ (n.log n). O QuickSort apresentou o melhor desempenho por aplicar operações rápidas, como incrementos, decrementos, comparações e algumas atribuições. Já o MergeSort apresentou o pior desempenho por necessitar de um maior processamento para manipular o vetor auxiliar, necessitando um espaço extra Θ (n).

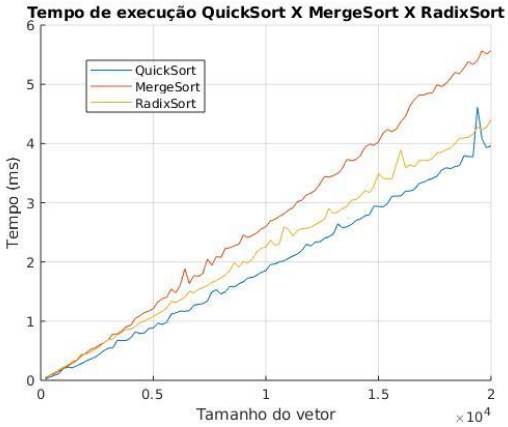

Figura 1. *Comparação entre os tempos de execução de diferentes algoritmos de ordenação*

2) MergeSort: Recursivo X Iterativo

Pela figura 2 a seguir, percebemos que os tempos de execução das variantes recursiva e iterativa do MergeSort são bastante semelhantes, independentemente do tamanho do vetor a ser ordenado, sendo sempre Θ (n.log n).

A diferença entre esses dois modelos se dá pela pilha de execução e quantidade de chamadas recursivas, como mostrado nas figuras 3 e 4 a seguir, mostrando como o espaço gasto na memória é muito maior no caso recursivo, o que explica o fato de o modo iterativo apresentar tempo de execução levemente menor que o modo recursivo para vetores maiores.

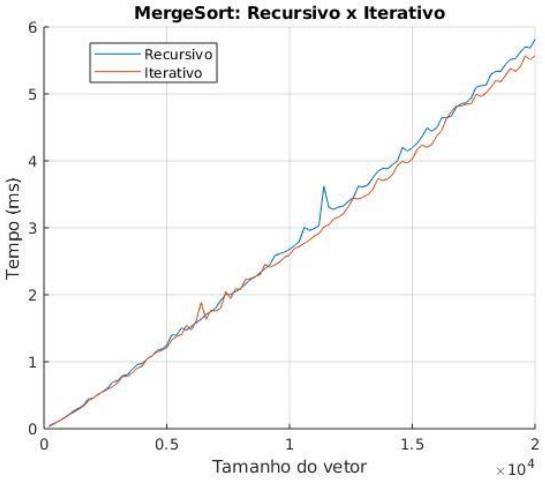

Figura 2. *Comparação do tempo de execução entre as duas formas de MergeSort*

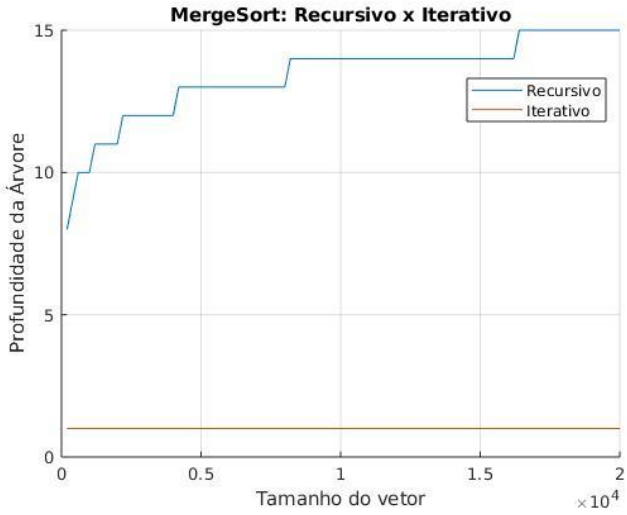

Figura 3. *Comparação da profundidade da árvore entre as duas formas de MergeSort*

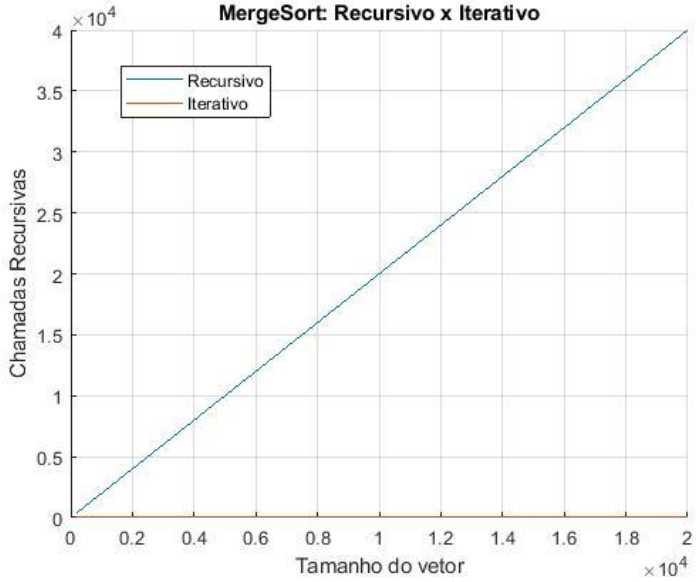

Figura 4. *Comparação da quantidade de chamadas recursivas entre as duas formas de MergeSort*

3) QuickSort com 1 recursão X QuickSort com 2 recursões

Para a análise desse item, os dois métodos de QuickSort foram implementados utilizando a mediana de três, que é mais eficaz que utilizando pivô fixo, como veremos posteriormente nesse relatório no item 4.

Assim como na comparação dos modos de MergeSort, os modos de QuickSort (com uma ou duas recursões) apresentam comportamento bastante semelhante no que se refere a tempo de execução, como visto na figura 5 a seguir, sendo ambos Θ (n.log n).

A diferença, novamente, se dá pela profundidade da árvore de execução e quantidade de chamadas recursivas, que podem ser observadas, respectivamente, nos gráficos das figuras 6 e 7 a seguir. Assim, por necessitar de maior processamento da memória, o método utilizando duas recursões é levemente mais demorado para vetores muito grandes.

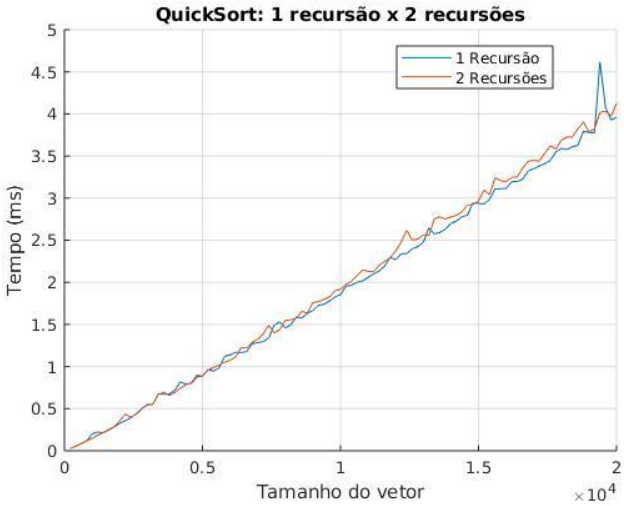

Figura 5. *Comparação do tempo entre as duas formas de QuickSort*

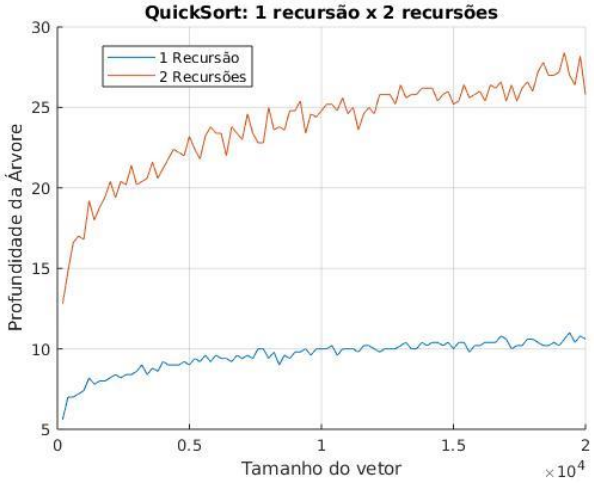

Figura 6. *Comparação da profundidade da árvore entre as duas formas de QuickSort*

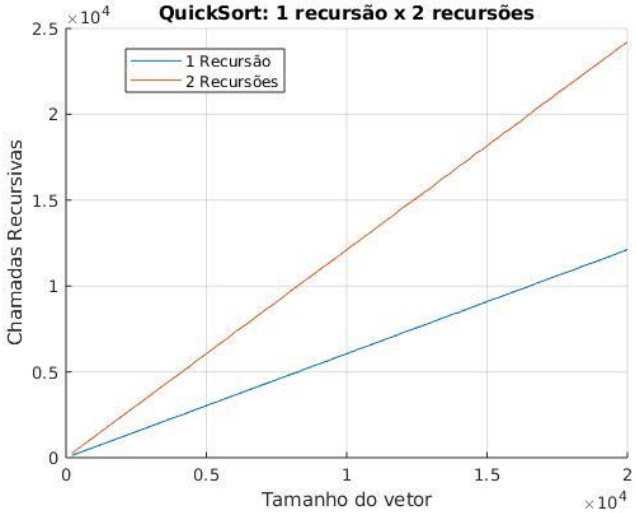

Figura 7. *Comparação das quantidades de chamadas recursivas entre as formas de QuickSort*

4) QuickSort com mediana de 3 X QuickSort com pivô fixo para vetores quase ordenados

Para vetores quase ordenados, podemos chegar, no pior caso, a tempos de execução da ordem Θ (n^2), ao invés do tempo ótimo de Θ (n.log n). Para minimizar essa possibilidade, determinamos o pivô utilizando a mediana de três ao invés de um pivô fixo. Assim, conforme a figura 8 a seguir, utilizando a mediana de três temos tempo de execução menores. Além disso, com esse método diminuímos a pilha de execução e a quantidade de chamadas recursivas, conforme as figuras 9 e 10 respectivamente.

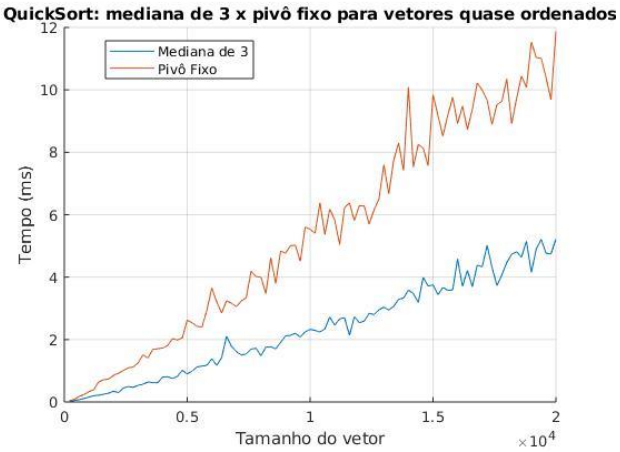

Figura 8. *Comparação do tempo entre mediana de 3 e pivô fixo para QuickSort*

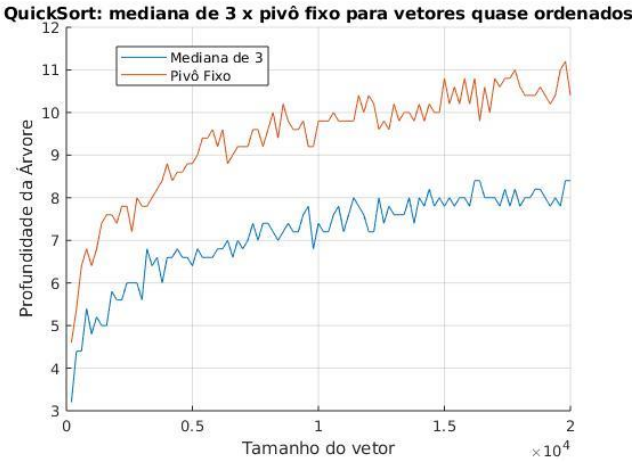

Figura 9. *Comparação da profundidade da árvore entre mediana de 3 e pivô fixo para QuickSort*

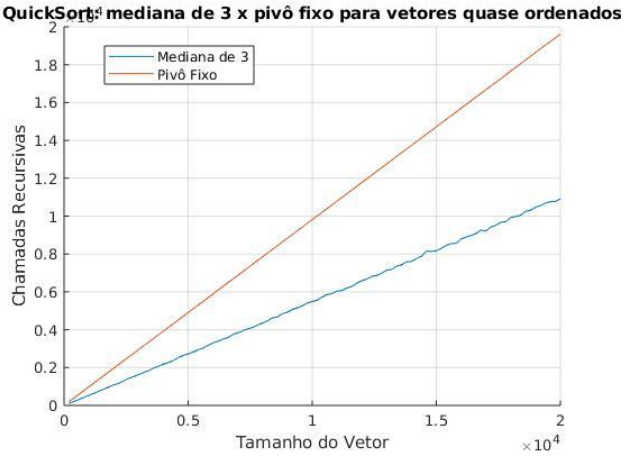

Figura 10. *Comparação das quantidades de chamadas recursivas entre mediana de 3 e pivô fixo para QuickSort*

5) Questão opinativa: Caso real de opinião de engenheiro: "Pode estar demorando muito por culpa da implementação de QuickSort da libc, já que alguns dos nossos dados já vem quase ordenados, fica quadrático". Certamente é possível, mas é plausível ou esperado que uma biblioteca importante, antiga, muito utilizada e bem testada, implemente QuickSort de forma a causar o problema citado? Em outras palavras, você, como chefe de um projeto, aprovaria um QuickSort implementado assim em um projeto real, considerando o custo-benefício? Justifique.

Não é esperado que a biblioteca cause tal problema, apesar de serem casos previstos em teoria. Sendo uma biblioteca tão consolidada e amplamente utilizada, certamente o QuickSort da libc foi projetado da melhor maneira possível, de modo a estar preparado para tratar corretamente diversos tipos de sequências a serem ordenadas, sejam elas randômicas, quase ordenadas ou mesmo ordenadas.

Assim, como o engenheiro responsável pelo projeto, eu aprovaria o uso dessa biblioteca, pois é muito mais provável que um novo método implementado apresente falhas na tentativa de tratar casos específicos do que utilizar uma função já conhecida, testada e implementada como a da libc. Além disso, pensando no custo-benefício de uma nova implementação, os casos que dão problema são bastante específicos, não valendo o esforço extra de buscar otimizá-los fora da biblioteca padrão, que já deve satisfazer a maior parte dos casos.