Instituto Tecnológico de Aeronáutica - ITA
CES-12 Algoritmos e estruturas de dados II
Aluno: Bernardo Hoffmann da Silva

Relatório do Laboratório 3 - Variações de algoritmos de ordenação

1 QuickSort X MergeSort X RadixSort

Realizando a comparação entre QuickSort, MergeSort e RadixSort, temos os gráficos apresentados nas figuras 1, 2 e 3.

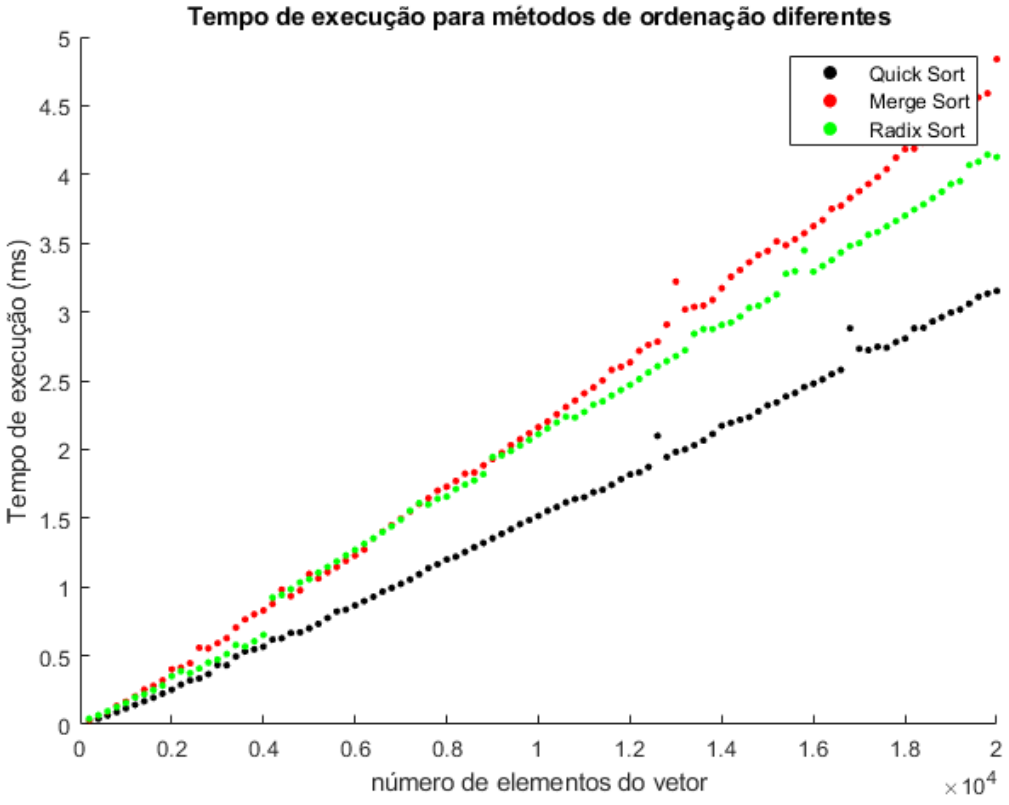

Figura 1

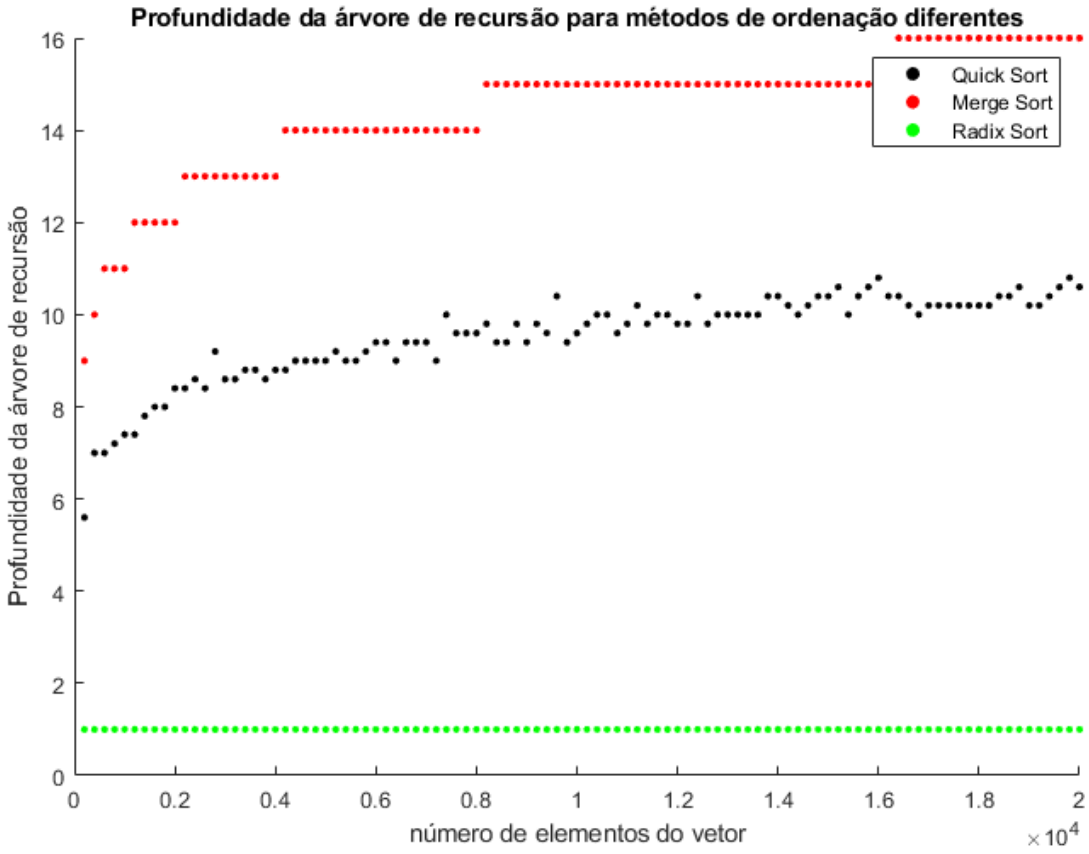
Profundidade da árvore de recursão para métodos de ordenação diferentes
Quick Sort
Merge Sort
Radix Sort
Profundidade da árvore de recursão
número de elementos do vetor
×10^4
0
2
4
6
8
10
12
14
16
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Figura 2

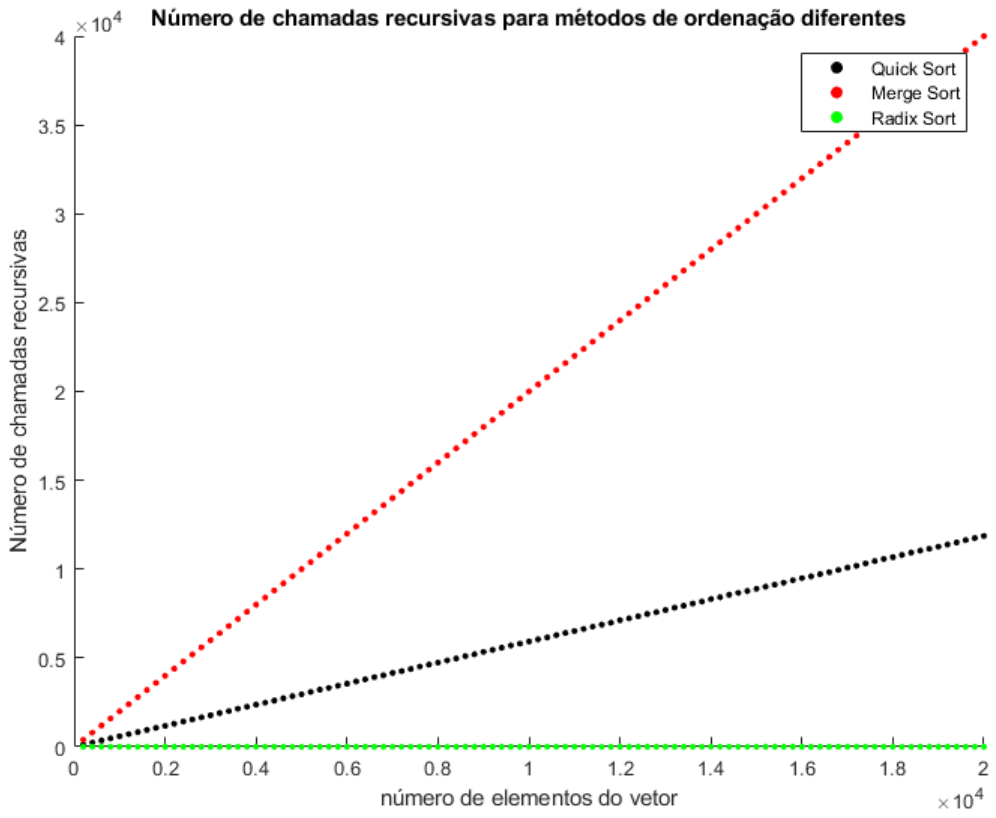

Figura 3

Nota-se que os três algoritmos de ordenação possuem tempos relativamente próximos, para os vetores apresentados, de 200 a 20000 elementos. Percebe-se que QuickSort e MergeSort possuem tempo aproximadamente de ordem $O(nlog(n))$ e RadixSort é aproximadamente linear, conforme prevê a teoria. Porém, como as constantes de tempo do RadixSort são muito altas, percebe-se que o QuickSort é mais rápido para os vetores utilizados, de até 20000 elementos aleatoriamente distribuídos.

Para a profundidade da pilha, percebe-se que tanto QuickSort quanto MergeSort possuem ordens de log(n) para a profundidade. Como RadixSort não é recursivo, sua profundidade é unitária. Porém, nota-se que MergeSort necessita de uma recursão mais profunda, o que mostra que o algoritmo Quick é mais efetivo.

Tal melhor efetividade do Quick é evidenciada ainda mais quando se observa o número de chamadas recursivas totais. Percebe-se como o coeficiente angular da reta do QuickSort é menor que do MergeSort, algoritmo que necessita de uma memória maior para ser executado. Para o Radix, apesar de não necessitar de recursão, apresenta constantes de tempo muito maiores, o que torna seu tempo maior que o QuickSort para a quantidade de elementos testada.

2 MergeSort: Recursivo x Iterativo

Podemos realizar comparações entre o Merge Sort recursivo e iterativo. A figura 4 apresenta um gráfico do tempo pela quantidade de elementos no vetor.

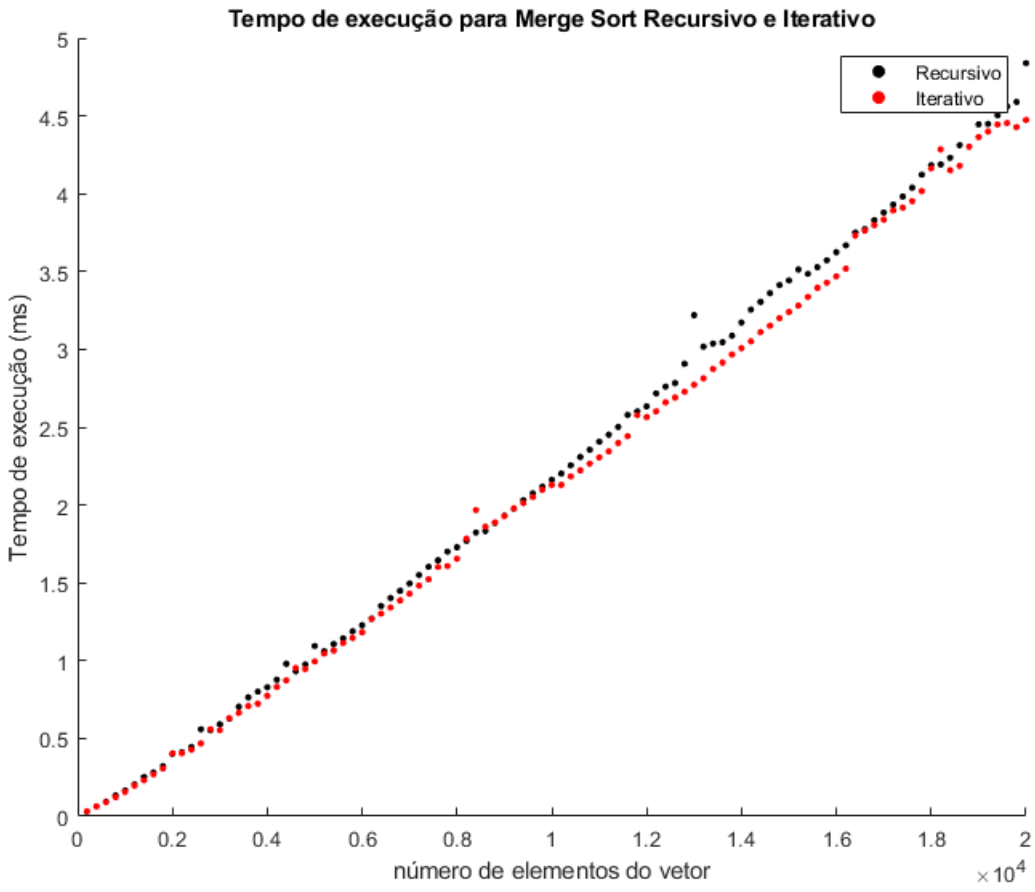

Figura 4

Percebe-se que o tempo é próximo, pois a complexidade de tempo para os dois algoritmos não muda. Porém, a utilização da memória é melhor otimizada para o Merge Sort iterativo, com menor uso da pilha de execução. Os gráficos do número de chamadas recursivas e de profundidade da pilha são omitidos pois ambas são 1 para o algoritmo iterativo. Essa melhor utilização da memória diminui as constantes do tempo, apesar da ordem se manter. Isso torna, de maneira geral, o Merge Sort iterativo um pouco mais rápido que o recursivo.

3 QuickSort com 1 recursão x QuickSort com 2 recursões

Pode-se comparar o QuickSort que trabalha com apenas 1 recursão com o que utiliza 2 recursões. Os gráficos do tempo, profundidade da recursão e número de chamadas recursivas são apresentados nas figuras 5, 6 e 7.

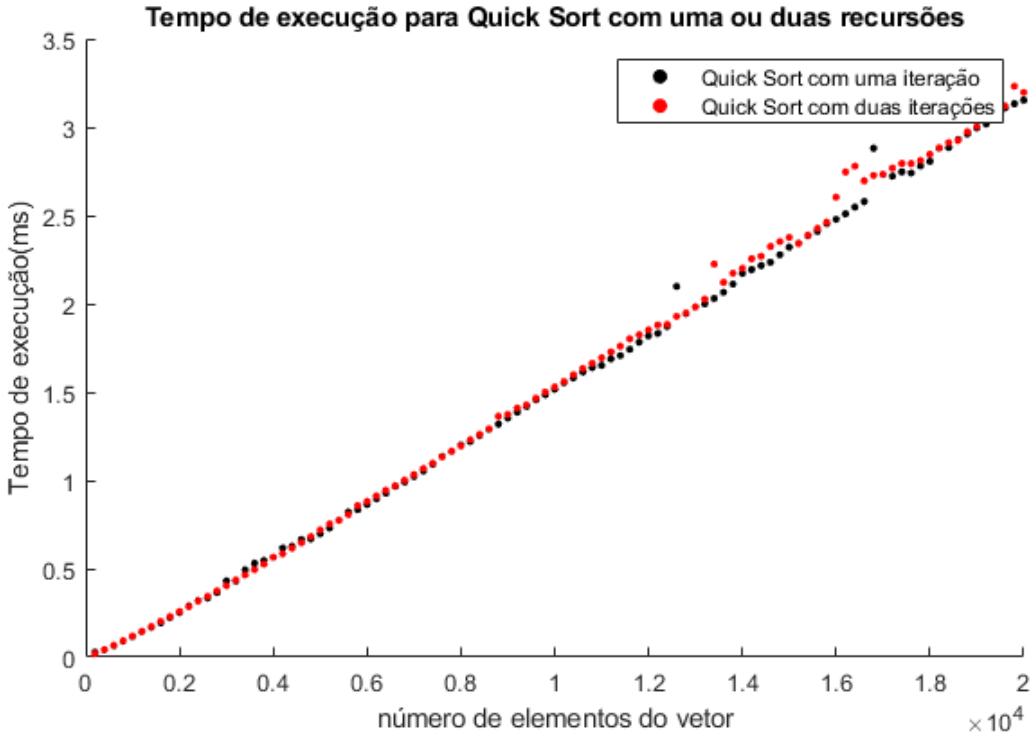

Figura 5

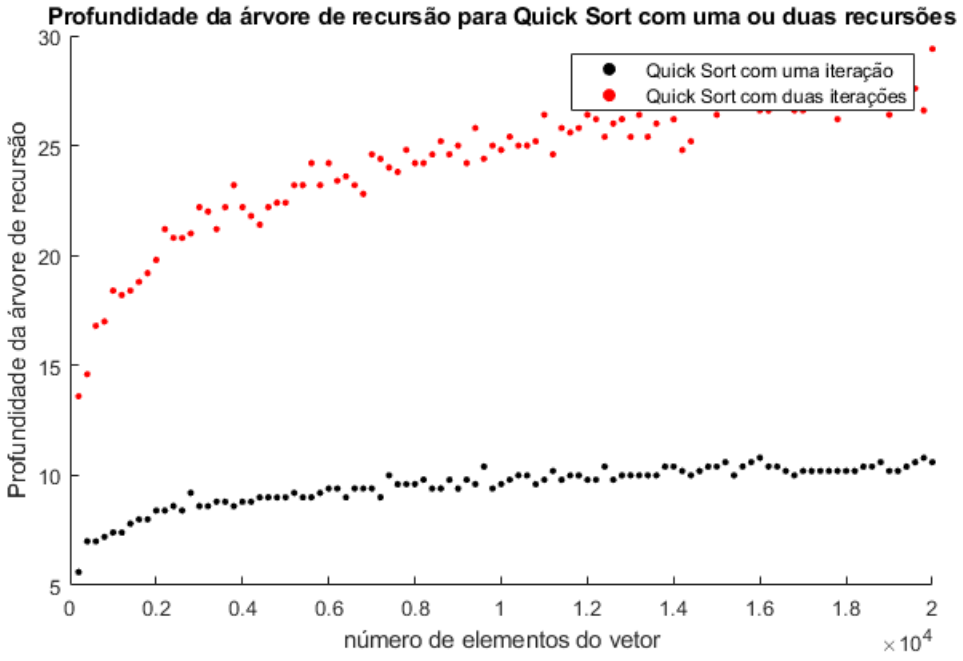

Figura 6

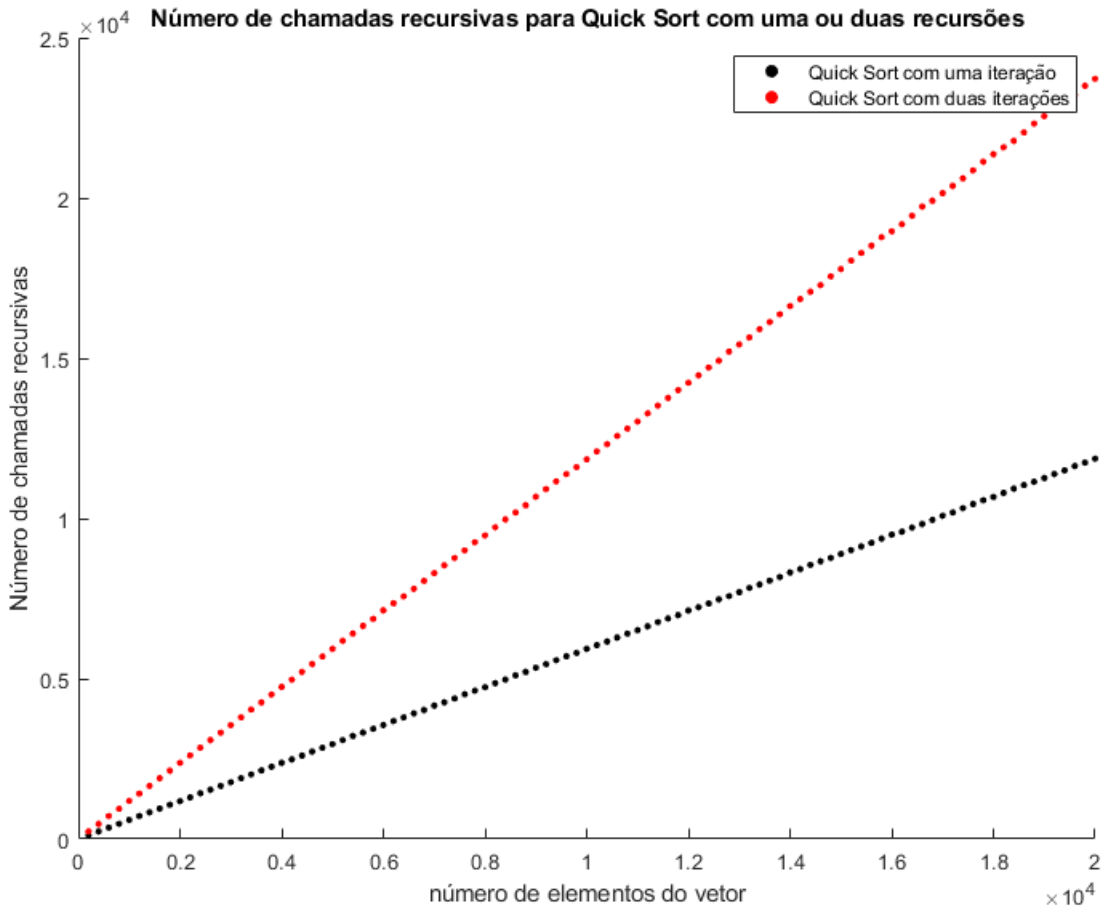

Figura 7

Nota-se que o tempo de execução não muda de complexidade com o número de recursões utilizadas (Para ambos, o caso médio é $O(nlog(n))$. Porém, há uma grande diferença na profundidade da pilha de execução. Para duas recursões, a pilha pode, no pior caso, se tornar $O(n)$, devido às recursões. Já para uma única recursão, a pilha se torna $O(log(n))$, o que traz uma economia de espaço grande. É expressiva também a diferença do número de chamadas recursivas.

4 QuickSort com mediana de 3 x Quicksort com pivô fixo para vetores quase ordenados

Outra comparação que pode ser feita é entre o QuickSort que utiliza mediana de 3 elementos para achar o pivô e o algoritmo QuickSort que utiliza um pivô fixo (no caso, o primeiro elemento do vetor). A figura 8 apresenta a diferença do tempo de execução para esses dois algoritmos em duas situações: Vetores aleatórios e vetores quase ordenados. Ambos são feitos com apenas uma recursão.

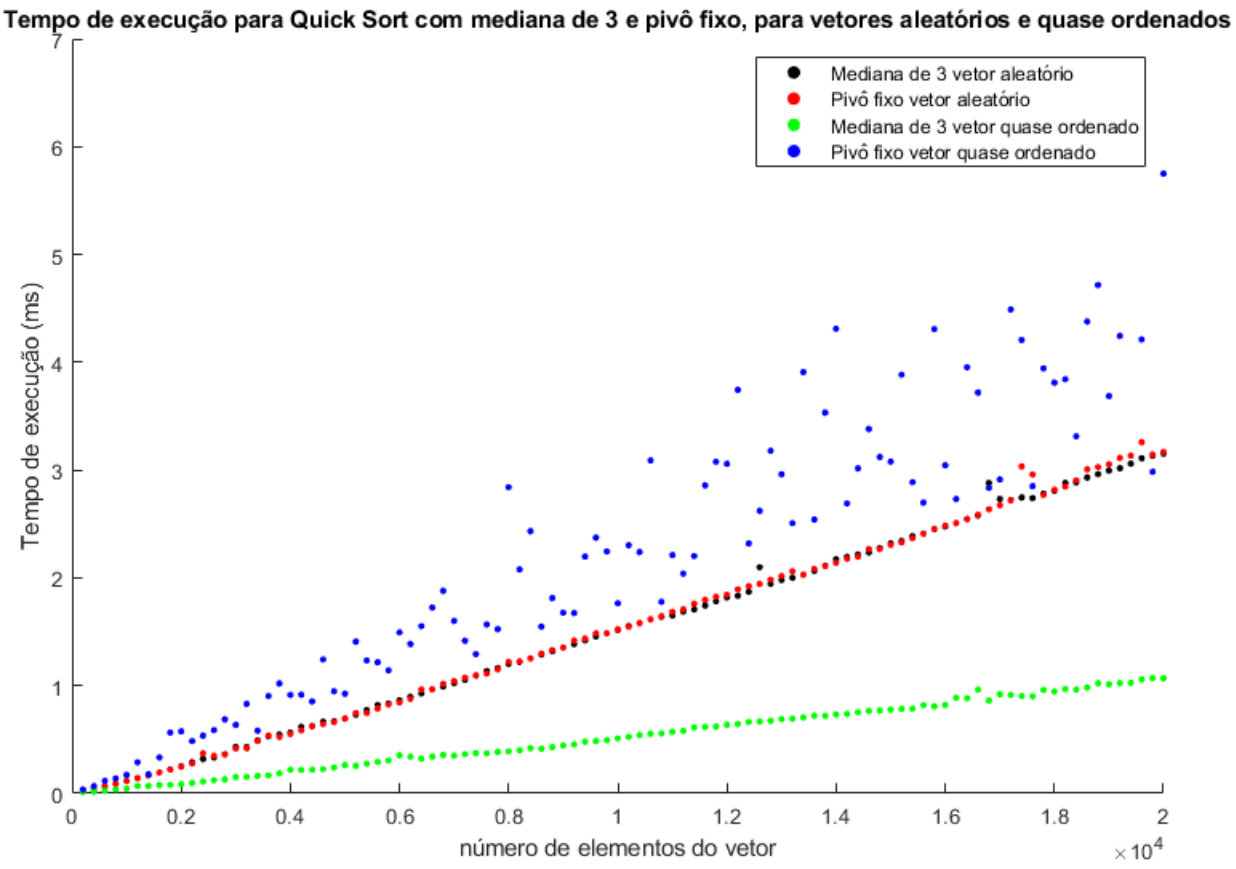

Figura 8

Percebe-se que o tempo de ambos é muito próximo no caso em que o vetor é aleatório. Isso ocorre porque um pivô mediana de 3 e o fixo possuem a mesma probabilidade de serem "bons"para a eficiência do algoritmo, isto é, estarem longe das extremidades do vetor quando ordenado. Já para o vetor quase ordenado, percebe-se como o tempo da mediana é expressivamente melhor do que o do algoritmo que utiliza pivô fixo. Isso ocorre pois em um vetor quase ordenado, a mediana de três elementos (o primeiro, o do meio e o último) possui a probabilidade alta de ser aproximadamente a mediana do vetor. Já o primeiro elemento do vetor quase ordenado possui a probabilidade alta de estar longe da mediana do vetor, aumentando muito a complexidade de tempo do algoritmo, que fará muito mais recursões que o necessário.

5 Caso real

É muito improvável que o QuickSort de uma biblioteca testada e aprovada esteja suscetível a um problema conhecido como o caso do vetor quase ordenado. É possível afirmar isso pois esse problema do algoritmo é muito conhecido e soluções possíveis são estudadas há tempos. Portanto, o custo-benefício de adicionar no QuickSort um pivô não fixo, como a mediana de três elementos do vetor, por exemplo, é muito bom. A implementação é simples e não custosa de

tempo ou de memória, enquanto a utilização de um pivô fixo pode gerar um problema de tempo muito grande, que é o caso do tempo quadrático.