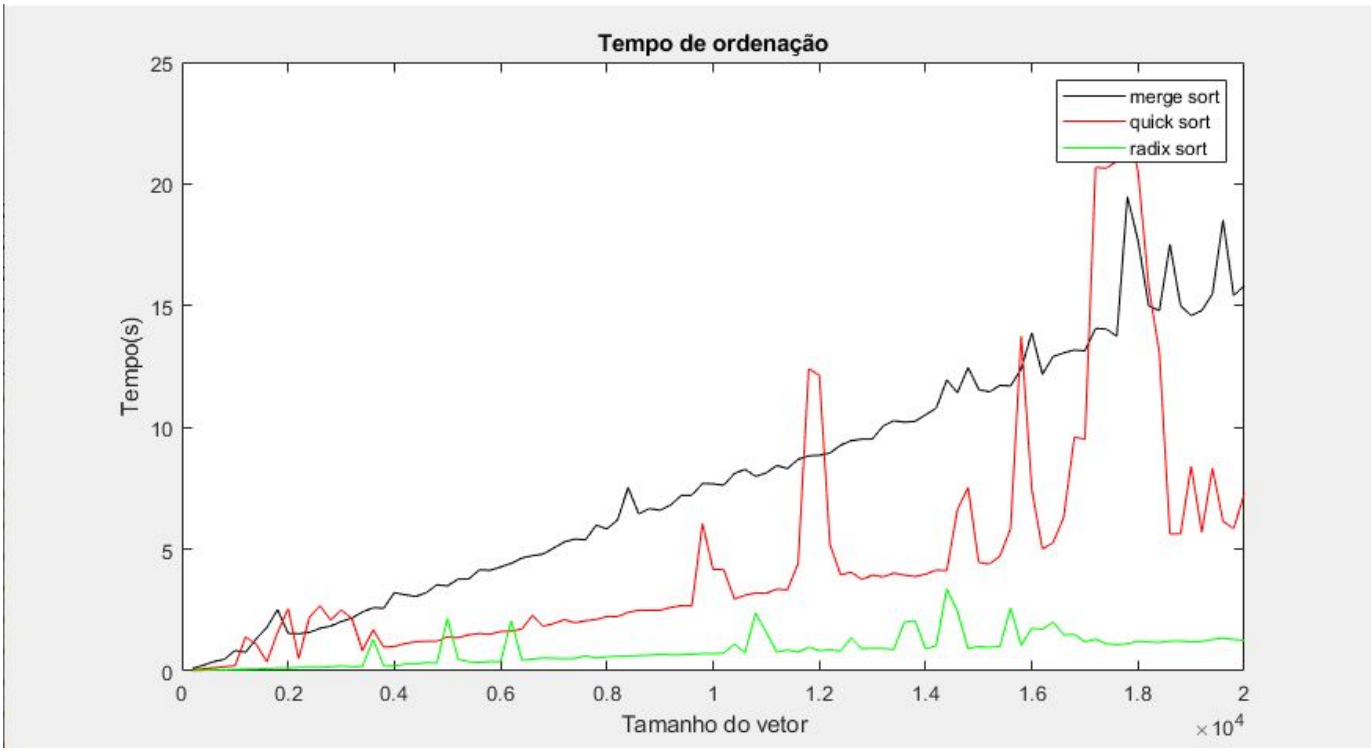
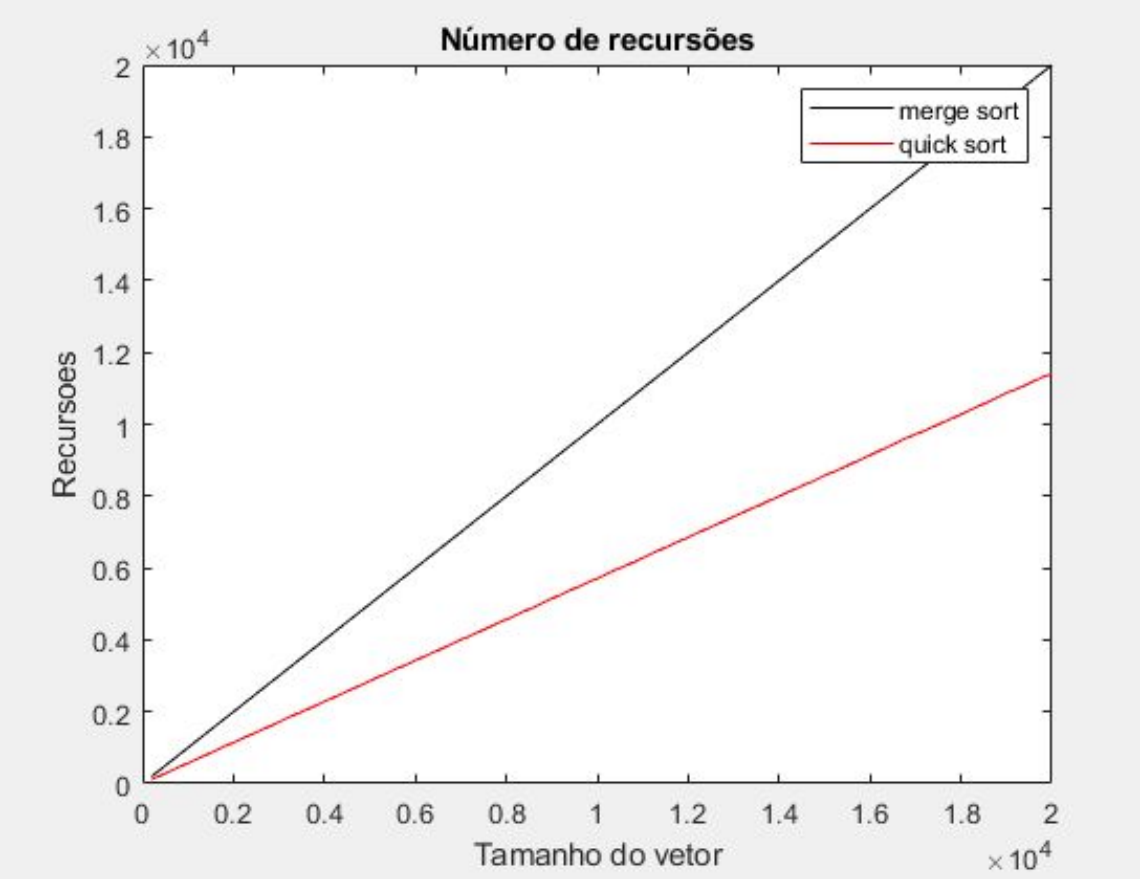
Aluno : Rodrigo Alves de Almeida
Laboratório 3

1) Tempo de Execução QuickSort vs MergeSort vs RadixSort

Como foi pedido para selecionar as versões mais rápidas de cada método de ordenação, para o QuickSort, foi utilizada a versão com uma recursão e vetores aleatórios e para o MergeSort, foi utilizada a versão iterativa.

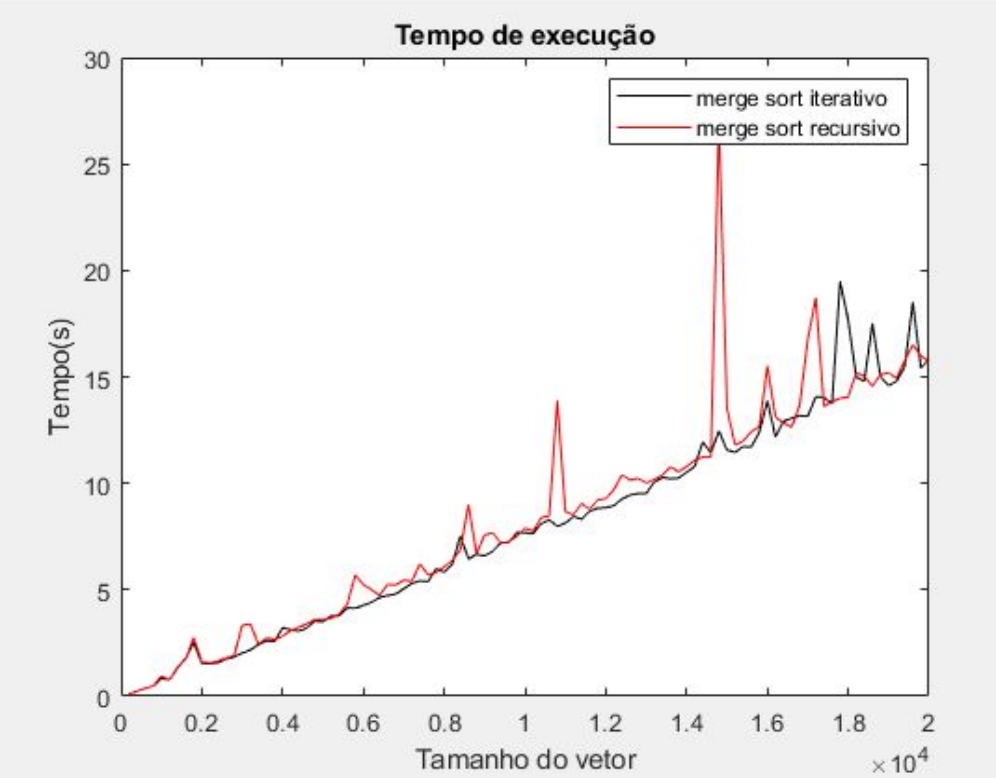


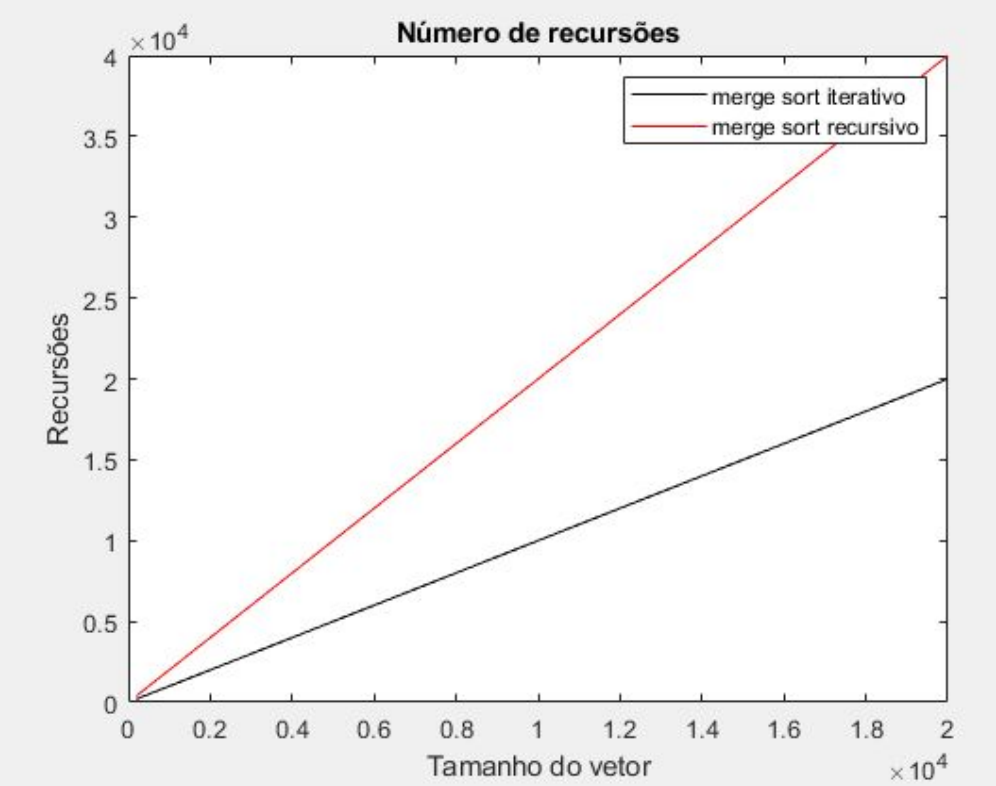


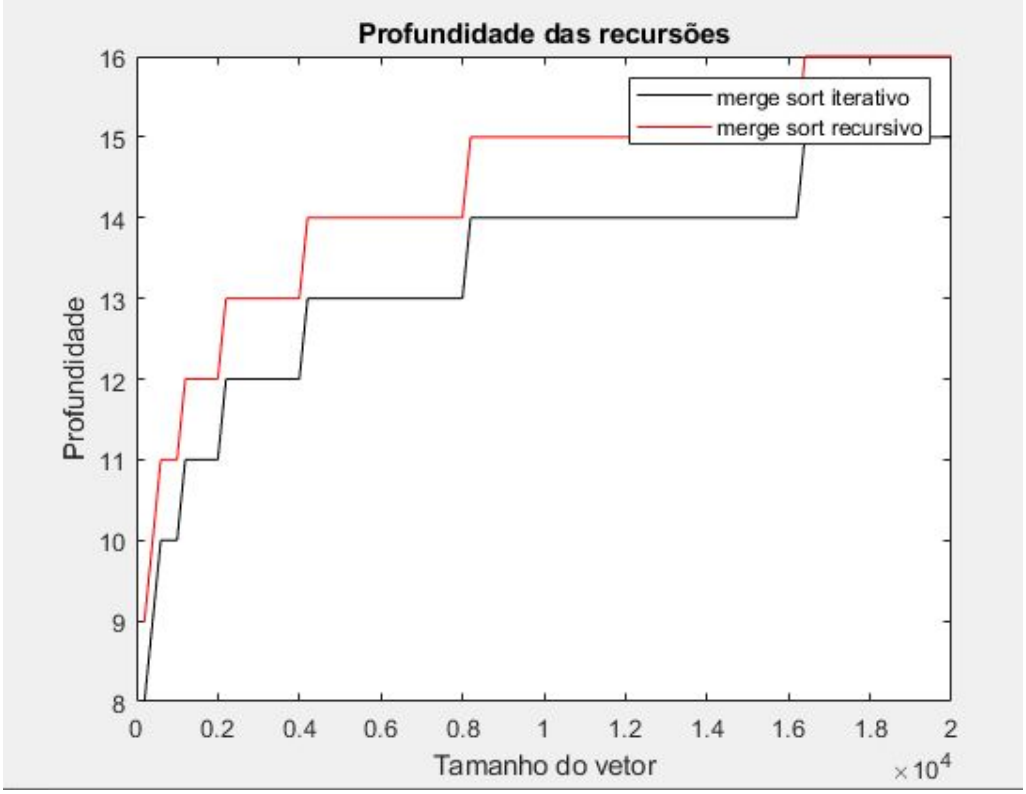
Nesse caso, o Radix Sort é o mais rápido pois os testes são realizados com vetores com tamanho limitado (o valor máximo é igual ao tamanho do vetor) e a implementação não foi feita baseada em comparações, mas operações com bits. Nessas condições, o Radix Sort tem ordenações em O(n).

O Quick Sort também se apresentou mais rápido que o MergeSort, pois no Merge Sort há mais operações de comparação, criação de vetores auxiliares e atribuições, enquanto o Quick Sort trabalha com uma quantidade menor de trocas e comparações. Além disso, é evidente que no Merge Sort há um número maior de recursões, o que aumenta o tempo de execução.

2) Merge Sort recursivo vs iterativo



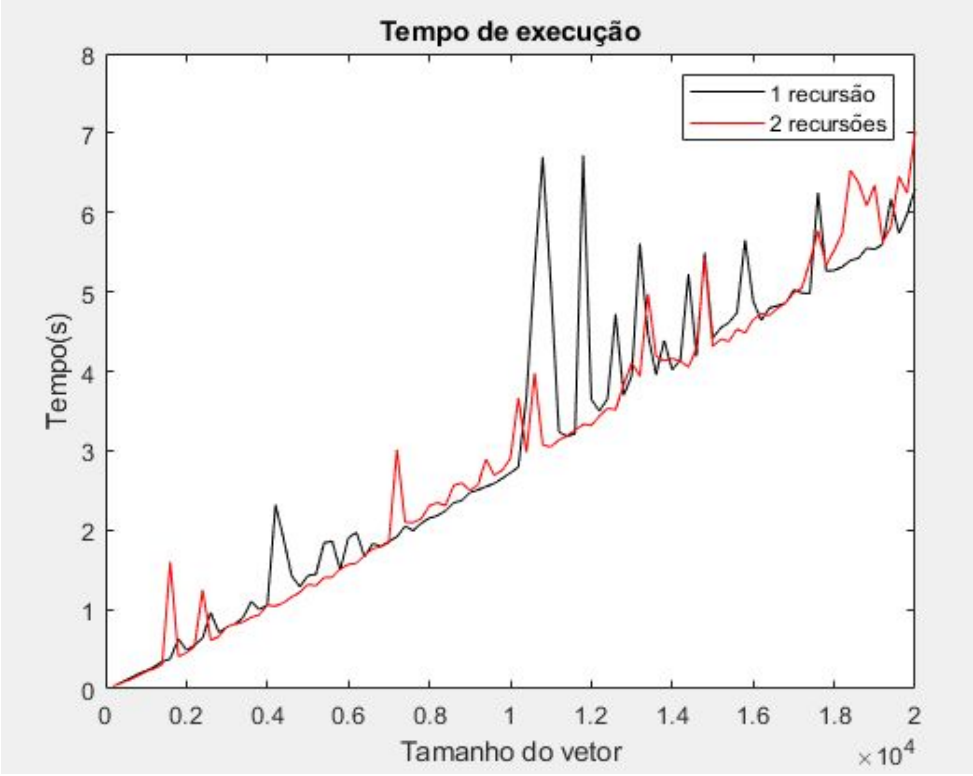


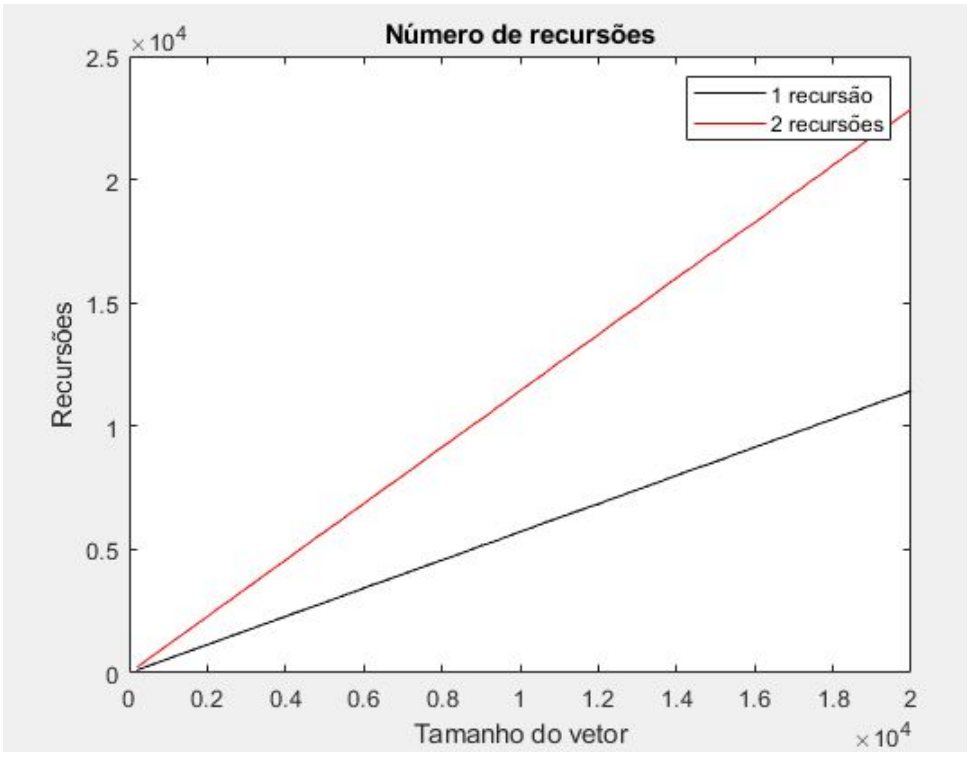


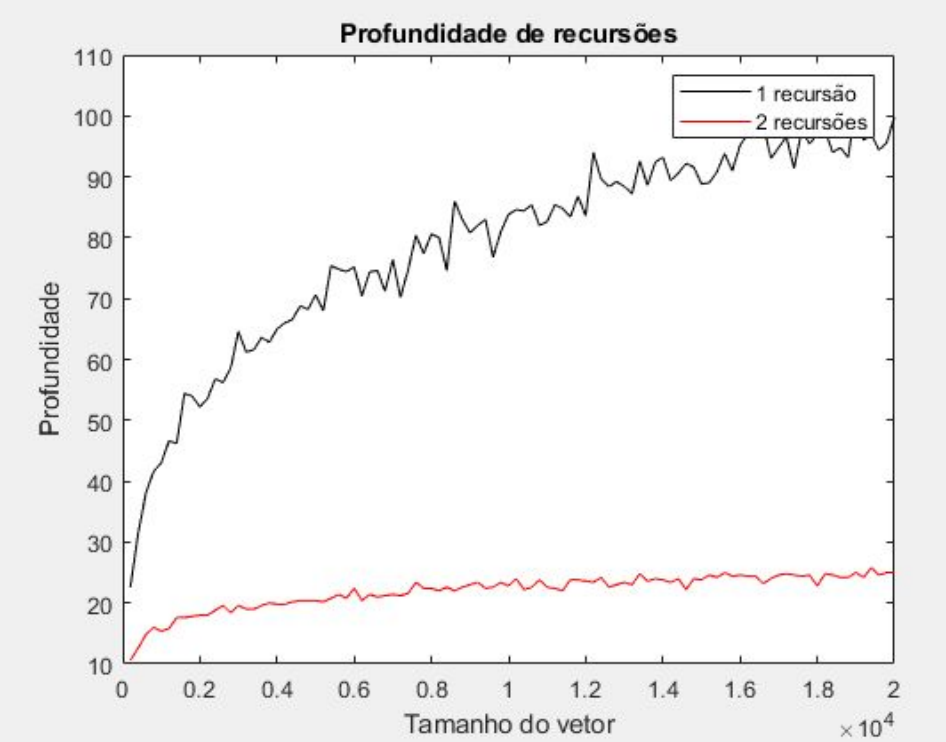
A diferença teórica entre o Merge Sort recursivo e iterativo está no fato de que o recursivo reduz a pilha de execução, enquanto a complexidade de tempo e espaço continuam inalteradas. Isso pode ser observado quando são comparados os tempos de execução dos algoritmos: ambos são muito semelhantes na média (o iterativo é levemente mais rápido devido a essa diminuição na pilha de execução).

Se considerarmos o número de recursões do algoritmo iterativo como sendo a quantidade de vezes que a função *merge* é chamada, observamos que, no iterativo, esse valor é menor, justamente pelo fato da pilha de execução ser mais enxuta. O mesmo raciocínio vale para a profundidade dessas recursões (no algoritmo iterativo, a profundidade foi definida por quantas vezes temos que multiplicar por 2 o tamanho dos blocos).

3) Quick Sort com 1 recursão vs Quick Sort com 2 recursões





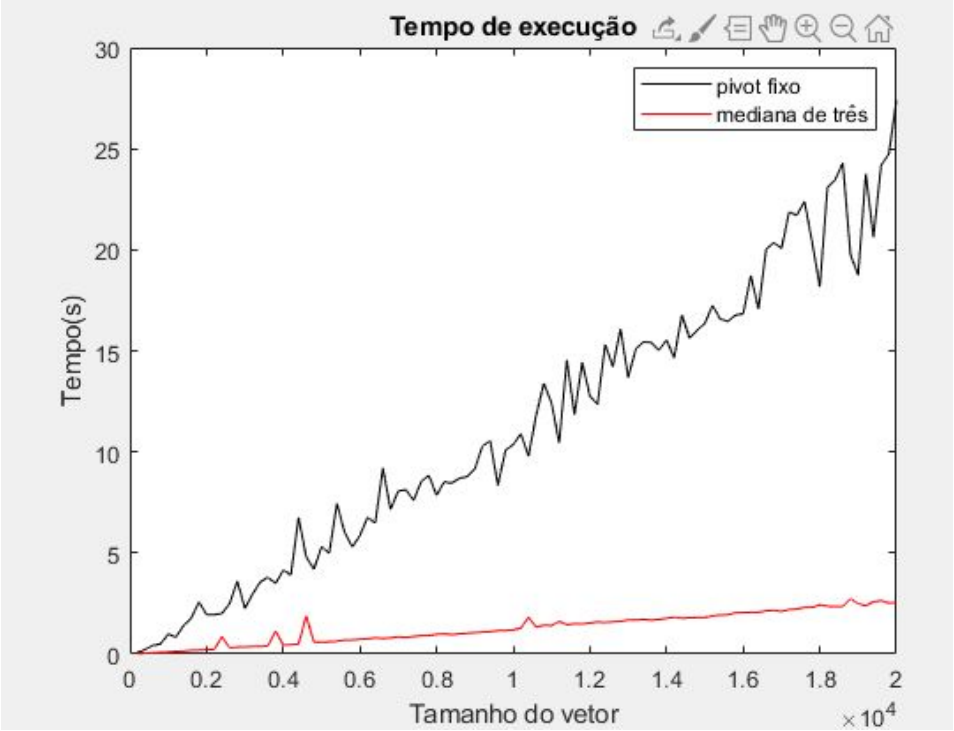


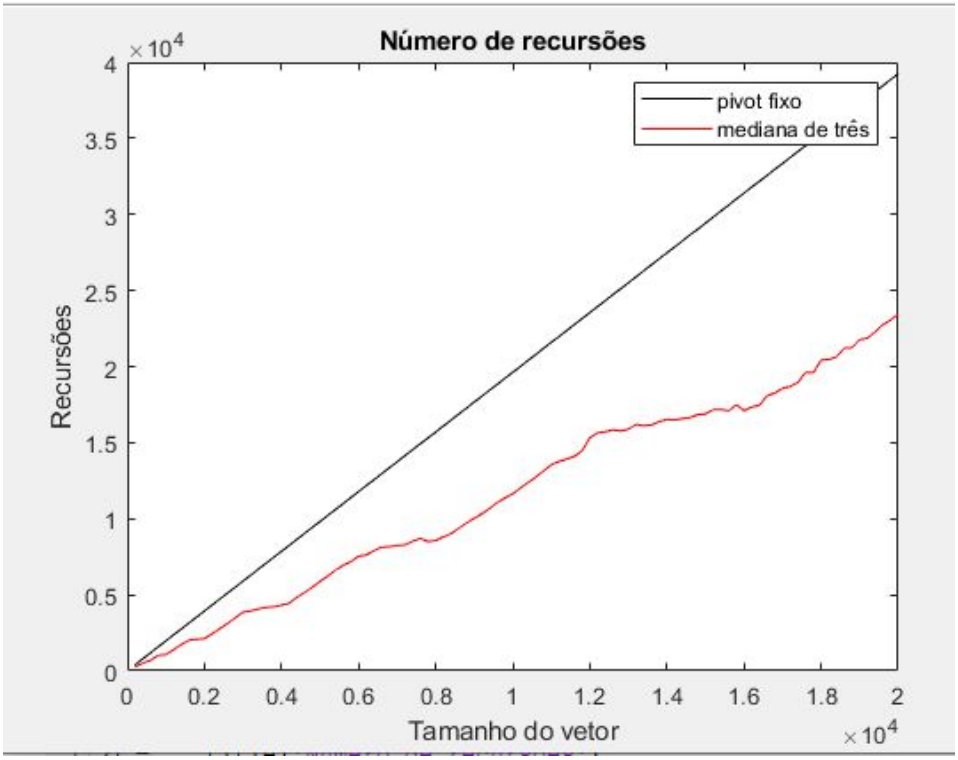
A versão de 1 recursão do Quick Sort é utilizada com a finalidade de diminuir a pilha de execução, passando o tamanho máximo desta de O(n) para O(log n). Deste modo, o tempo de execução das duas versões é muito semelhante, como podemos observar no gráfico de tempo de execução.

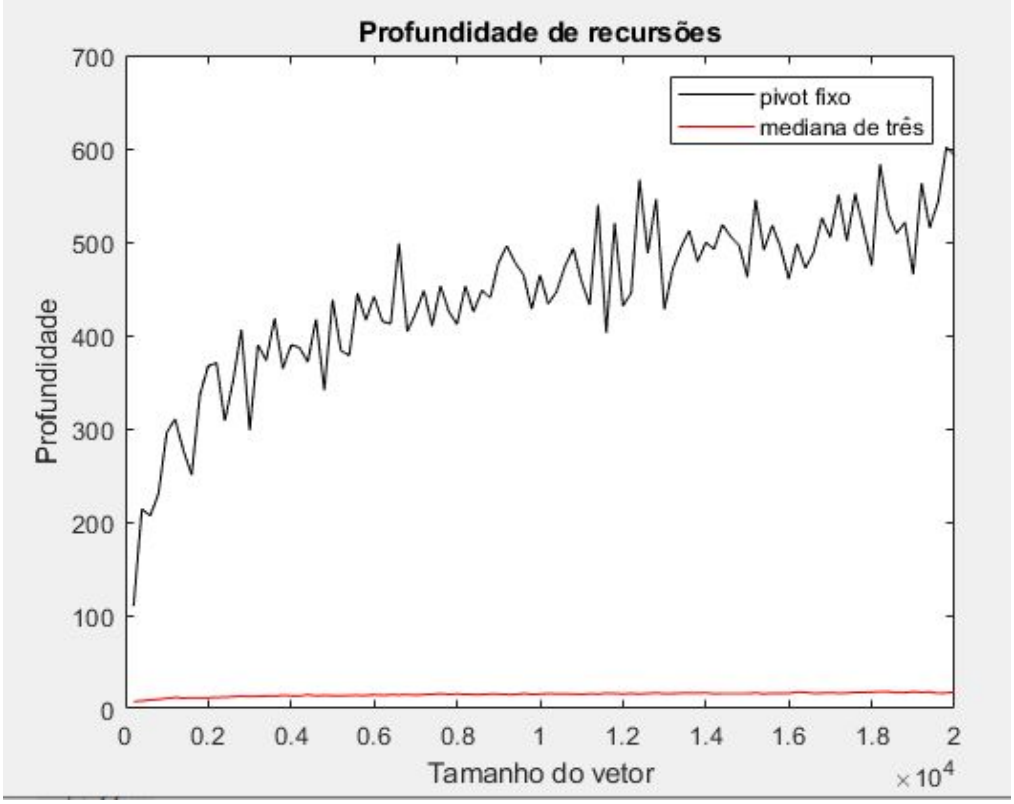
Também é evidente que, no algoritmo de duas recursões, o número total de recursões é maior, como pode-se observar.

No algoritmo de 1 recursão, porém, em toda iteração realizada é selecionada a maior metade do vetor para seguir realizando as partições, de modo que há um caminho máximo que é tomado na execução desse algoritmo. Desse modo, a profundidade das execuções das partições é maior nesse algoritmo. Em outras palavras, pode-se dizer que o algoritmo de 1 recursão toma caminhos mais profundos mas em menor quantidade.

4) **Quick Sort com mediana de 3 vs Quick Sort com pivô fixo para vetores quase ordenados**





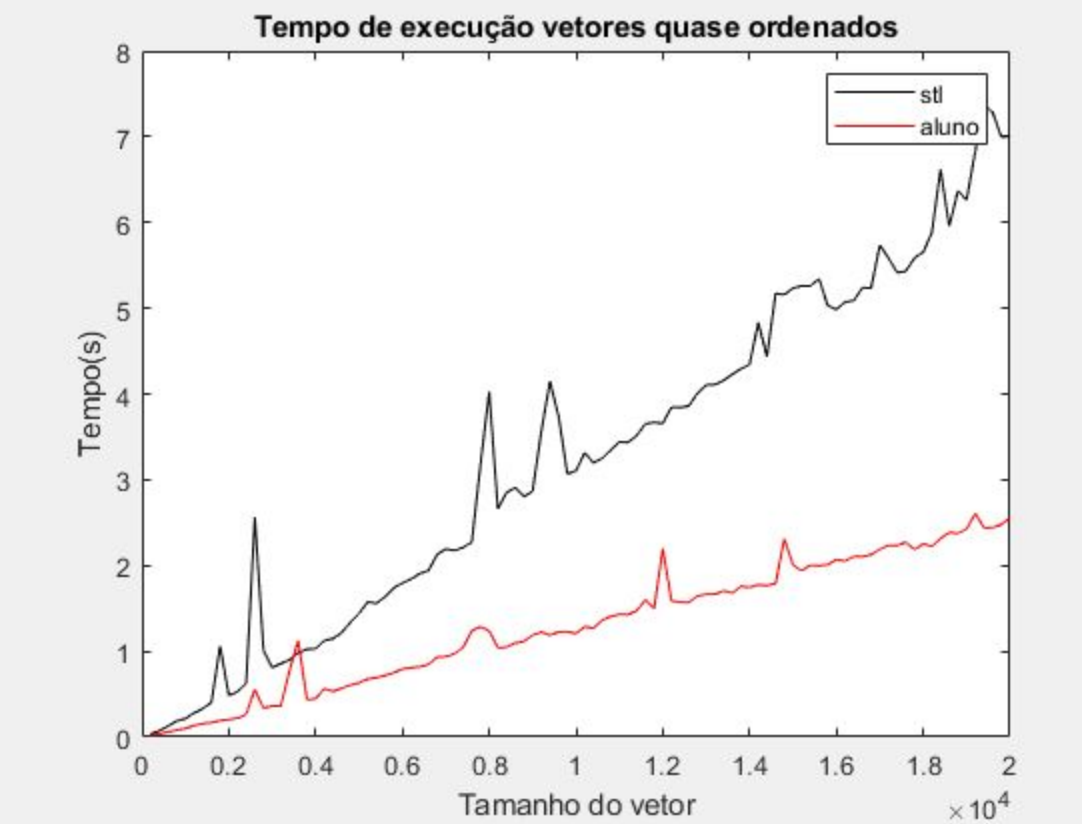


Em vetores quase ordenados, o método de pivot fixo se mostra muito inferior quando escolhemos o pivot como sendo o primeiro elemento, pois na maioria das partições não serão realizadas quase nenhuma mudança no vetor, visto que o pivot provavelmente já estará na sua posição correta. Por outro lado, quando tomamos a mediana de 3, os pivots distribuem muito melhor o vetor. Este fato é evidenciado no gráfico de tempo acima.

O número de recursões no pivot fixo também é maior, pois como foi explicado acima a escolha dos pivots faz uma divisão pouco uniforme do vetor, o que resulta em muitas recursões. Este também é o motivo das grandes profundidades: são necessárias muitas chamadas em sequência até chegar nos subvetores elementares.

5) Questão Opinativa

Comparando o tempo de execução para vetores quase ordenados da implementação da libc com a minha implementação do QuickSort com 1 recursão e mediana de três obtemos o seguinte gráfico:



Desse modo, como engenheiro de projeto, eu iria preferir utilizar a minha própria implementação do Quick Sort , visto que a implementação da stl se mostra lenta para esses vetores quase ordenados.