Exercício prático individual: Tempo de execução de algoritmos
Aluno: Luiz Felipe Vieira

1. Hardware:
   Nome do sistema operacional: Microsoft Windows 11 Home
   Versão do sistema operacional: 10.0.22621 N/A compilação 22621
   Tipo de sistema: x64-based PC
   Processador(es): Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
   Memória física total: 16.215 MB

2. Resultados obtidos (MergeSort e QuickSort):

MergeSort

Tamanho do vetor	variação %	Ordenações feitas	Média de comparações	variação %	Média de trocas	variação %	Média tempo (Milissegundo)	variação %	n*log n para comparaçoes
62.500	-	50	917.127,64	-	454.924,64	-	6,0	-	690.183
125.000	100%	50	1.959.263,78	114%	972.424,32	114%	11,9	98%	1.467.009
250.000	100%	50	4.168.571,18	113%	2.069.801,66	113%	24,9	110%	3.107.304
375.000	50%	50	6.477.616,54	55%	3.199.810,80	55%	45,4	82%	4.813.005
1.000.000	167%	50	18.674.302,62	188%	9.279.117,96	190%	110,2	143%	13.815.511

QuickSort

Tamanho do vetor	variação %	Ordenações feitas	Média de comparações	variação %	Média de trocas	variação %	Média tempo (Milissegundo)	variação %	n*log n para comparaçoes
62.500	-	50	1.209.168,44	-	41.681,62	-	3,9	-	690.183
125.000	100%	50	2.574.292,64	113%	83.343,32	100%	7,6	92%	1.467.009
250.000	100%	50	5.530.958,26	115%	166.716,06	100%	16,2	115%	3.107.304
375.000	50%	50	8.577.480,26	55%	250.121,06	50%	25,2	55%	4.813.005
1.000.000	167%	50	24.752.170,32	189%	669.039,42	167%	71,7	185%	13.815.511

Tanto o QuickSort quanto o MergeSort exibem complexidade de tempo O(n * log n), geralmente analisada em termos de comparações entre elementos nesses algoritmos. Embora a literatura sugira que o MergeSort seja superior ao QuickSort para volumes de dados mais elevados, **meus resultados mostraram que o QuickSort teve um desempenho superior em termos de tempo**.

Ambos os algoritmos demonstraram comportar-se conforme o esperado. Conforme o tamanho da entrada de dados aumentou, observamos um aumento correspondente no tempo de execução, na quantidade de operações e nas trocas de elementos. Essa relação é um **indicativo de que os algoritmos são estáveis**.

É importante notar que o aumento percentual no tempo de execução é um pouco maior do que o aumento na entrada de dados, o que está alinhado com a complexidade O(n * log n), como esperado.

Ao validar a complexidade O(n * log n) com entradas de tamanho n, observamos valores razoáveis que se aproximam das comparações teóricas. No entanto, é importante ressaltar que o desempenho dos algoritmos na prática pode variar um pouco em relação ao que a literatura descreve, devido a fatores específicos da implementação e do ambiente de execução.

Além da análise de tempo, é importante considerar o uso de espaço pelos algoritmos. **O QuickSort tende a ser mais eficiente em termos de espaço quando comparado ao MergeSort**. Isso ocorre porque o QuickSort é um algoritmo de ordenação "in-place", o que significa que ele não requer espaço adicional significativo para armazenar dados temporários durante a ordenação.

Por outro lado, o MergeSort é um algoritmo "out-of-place", o que implica que ele geralmente requer espaço adicional para armazenar os novos arrays durante o processo de ordenação. Em situações onde a memória disponível é um recurso crítico, o QuickSort pode ser preferível devido ao seu uso mais eficiente de espaço.

3. Resultados obtidos QuickSort (ordenado e não ordenado):

QuickSort							
Tipo do vetor	Tamanho	Ordenações feitas	Média de comparações	Média de trocas	Média tempo (Milissegundo)	n*log n para comparaçoes	n^2 para comparaçoes pior caso
sorted	10.000	10	49.995.000	9.999,00	65,0	92.103	100.000.000
random	10.000	10	155.954	6.666,10	0,5	92.103	100.000.000

A análise do desempenho do algoritmo QuickSort revelou diferenças significativas entre sua eficiência ao lidar com vetores ordenados e vetores com dados aleatórios. **Para vetores com elementos embaralhados, o QuickSort demonstrou um desempenho que se aproximou da complexidade teórica esperada, que é O(n * log n)**. No entanto, ao lidar com **vetores ordenados, o algoritmo atingiu um desempenho próximo do pior caso, que é O(n^2)**. Isso ocorre devido à escolha do pivô, que desempenha um papel crucial quando os dados já estão parcialmente ou totalmente ordenados.

O QuickSort opera selecionando um elemento como pivô e rearranjando o vetor de modo que os elementos menores que o pivô fique à esquerda e os elementos maiores fiquem à direita. Quando o pivô é consistentemente escolhido como o primeiro elemento (ou o último, que é comum em muitas implementações), e o vetor já está ordenado, o QuickSort realiza muitas comparações e trocas redundantes.

Essa ineficiência ocorre porque, em um vetor ordenado, o pivô sempre será o menor ou o maior elemento, dependendo da escolha do pivô. Isso resulta em partições desbalanceadas, levando a múltiplas chamadas recursivas com apenas um elemento à esquerda ou à direita do pivô, o que cria a complexidade quadrática.