Resolução de Problemas Estruturados em Computação
Tde3- Ordenação **Prof: Andrey Cabral Meira**
Aluno: João Vitor Zambão **BCC turma: 4B**
Endereço GitHub: https://github.com/JVZambao/TDE_Sort 1.
Descrição de cenário:

1.1. Neste trabalho pediu-se que desenvolvêssemos algoritmos de sorteamento, na linguagem Java, o tipo a qual ficou ao critério dos alunos, para primeiramente comprovar o aprendizado do conteúdo e para testar a efetividade dos métodos normalmente utilizados.

1.2. Em meu projeto foram utilizados os algoritmos: BubbleSort (Lista A), MergeSort e QuickSort (Lista B). Feito no NetBeans.

2. Como rodar o código:

2.1. De o Run no main da classe SortingPerformanceAnalyzer para gerar um array aleatório, o qual será entregue para as três classes Sort simultaneamente. Para visualizar as estatísticas de cada de cada método e conferir a ordenação, acesse o arquivo sorting_performance.csv emcontrado na pasta TDE_Andrey.

2.2. Para mudar o tamanho do vetor, deve-se mudar diretamente no código o número de elementos desejados.

Funções Ultilizadas: Abaixo está uma breve explicação sobre cada metodo de sort junto com uma planilha, contendo 5 interações de cada método, em Arrays de 50, 500, 1000, 5000 e 10000 elementos, como orientado no execicio.

BubbleSort: O BubbleSort é um algoritmo de ordenação simples que funciona comparando e trocando elementos adjacentes, movendo o maior elemento para o final em cada iteração. Esse processo se repete até que o array inteiro esteja ordenado.

Dados:

Teste	N° de termos	Tempo(ns)	N° de Interações	N° de Trocas
1°	50	56700	1225	547
2°	50	53500	1225	493
3°	50	54600	1225	574
4°	50	56900	1225	627
5°	50	68500	1225	611
Media	xxxxxxxxxxx	58040	1225	570.4
1°	500	3698200	124750	61744
2°	500	3476200	124750	63927
3°	500	2922800	124750	62204
4°	500	3585800	124750	62274
5°	500	4951500	124750	61127
Media	xxxxxxxxxxx	3726900	124750	62255.2
1°	1000	7092900	499500	242548
2°	1000	5798300	499500	252276
3°	1000	6680800	499500	253273
4°	1000	6380900	499500	253534
5°	1000	6677000	499500	248540
Media	xxxxxxxxxxx	6525980	499500	250034.2
1°	5000	85078700	12497500	6218472
2°	5000	83836600	12497500	6245183
3°	5000	85221900	12497500	6288693
4°	5000	90122200	12497500	6211947
5°	5000	76428100	12497500	6245014
Media	xxxxxxxxxxx	84137500	12497500	6241861.8
1°	10000	182975000	49995000	24784604
2°	10000	176212500	49995000	24820076
3°	10000	177443400	49995000	25157884
4°	10000	181630400	49995000	25175560
5°	10000	191202600	49995000	24875125
Media	xxxxxxxxxxx	181892780	49995000	24962649.8

MergeSort: O MergeSort divide o array em duas metades, ordena essas metades e depois mescla as partes ordenadas para obter o array final ordenado.

Dados:

Teste	N° de termos	Tempo(ns)	N° de Interações	N° de Trocas
1°	50	31400	221	133
2°	50	60800	222	133
3°	50	35700	224	133
4°	50	30900	222	133
5°	50	37700	214	133
Media	xxxxxxxxxxx	39300	220.6	133
1°	500	388400	3841	2216
2°	500	386400	3861	2216
3°	500	403600	3852	2216
4°	500	400100	3861	2216
5°	500	880900	3864	2216
Media	xxxxxxxxxxx	491880	3855.8	2216
1°	1000	779700	8723	4932
2°	1000	984200	8715	4932
3°	1000	678300	8715	4932
4°	1000	663300	8696	4932
5°	1000	652800	8689	4932
Media	xxxxxxxxxxx	751660	8707.6	4932
1°	5000	1714800	55227	29804
2°	5000	2195000	55212	29804
3°	5000	1966300	55230	29804
4°	5000	1970500	55265	29804
5°	5000	1676600	55244	29804
Media	xxxxxxxxxxx	1904640	55235.6	29804
1°	10000	3033700	120465	64608
2°	10000	3092900	120437	64608
3°	10000	2933000	120515	64608
4°	10000	3162800	120364	64608
5°	10000	3073000	120489	64608
Media	xxxxxxxxxxx	3059080	120454	64608

QuickSort: O QuickSort é um algoritmo de ordenação eficiente baseado na estratégia "dividir para conquistar". Ele escolhe um elemento chamado de pivô e particiona o array em duas partes, uma com elementos menores que o pivô e outra com elementos maiores. Em seguida, ele ordena recursivamente essas duas partes.

Dados:

Teste	N° de termos	Tempo(ns)	N° de Interações	N° de Trocas
1°	50	19100	236	139
2°	50	24200	270	149
3°	50	19300	247	178
4°	50	19200	273	140
5°	50	21800	240	145
Media	xxxxxxxxxxxx	20720	253.2	150.2
1°	500	221400	4471	2532
2°	500	237200	4873	2882
3°	500	213700	4226	2692
4°	500	248800	5135	2690
5°	500	339900	4601	3053
Media	xxxxxxxxxxxx	252200	4661.2	2769.8
1°	1000	501300	10236	6089
2°	1000	625900	11261	7219
3°	1000	451600	10681	6554
4°	1000	456700	10826	6956
5°	1000	431700	10312	6604
Media	xxxxxxxxxxxx	493440	10663.2	6684.4
1°	5000	1302700	73319	47908
2°	5000	1729600	74572	47771
3°	5000	1725900	75877	47583
4°	5000	1327900	73492	44788
5°	5000	1350100	72013	50082
Media	xxxxxxxxxxxx	1487240	73854.6	47626.4
1°	10000	2381000	171601	121252
2°	10000	2469800	172981	123634
3°	10000	2199100	173362	118246
4°	10000	2617500	170074	124619

5°	10000	1792300	179139	127198
Media	xxxxxxxxxxx	2291940	173431.4	122989.8

Conclusão:

QuickSort: Este apresenta as maiores mudanças entre interações, porém ele também apresenta o menor tempo de operação e o menor número de trocas, sendo o método mais eficiente entre os meios utilizados.

MergeSort: Este método apresenta uma maior consistência os métodos utilizados, apesar de não ser tão eficiente quanto o QuickSort, ele continua sendo um bom método de sort com um tempo de interação relativamente baixo.

BubbleSort: Este método é apresenta o pior desempenho entre os métodos apresentados, tendo um tempo de execução quase 60 vezes maior que os outros métodos, e um número de interações muito maior também, no senário observado não se vê um motivo para se utilizá-lo sobre os outros.