FAESA CENTRO UNIVERSITÁRIO

ARTHUR SILVA VASONCELOS XAVIER
ICARO RODRIGUES PORTO
JUAN RODRIGUES DE OLIVEIRA
ISRAEL DE OLIVEIRA BARBOSA

PESQUISA E ORDENAÇÃO

VITÓRIA, 2021

ARTHUR SILVA VASONCELOS XAVIER

ICARO RODRIGUES PORTO

JUAN RODRIGUES DE OLIVEIRA

ISRAEL DE OLIVEIRA BARBOSA

PESQUISA E ORDENAÇÃO

Atividade Desenvolvida por Arthur Silva Vasoncelos Xavier Icaro Rodrigues Porto Juan Porto Israel Oliveira na matéria Pesquisa e ordenação, sob orientação de Cíntia Caliari.

VITÓRIA, 2021

SUMÁRIO

INTRODUÇÃO

Pesquisa realizada a fim de observar a eficiência e velocidade de algorítmos de ordenação

VITÓRIA, 2021

1 ASPECTOS DO COMPUTADOR

COMPONENTE	MODELO
PROCESSADOR	Ryzen 7 5700g 3.80Ghz Octa core
MEMÓRIA RAM	T-Force 8gb DDR4 3600mhz

2 OS ALGORÍTMOS

Inserção Direta

o ArrayList é dividido em dois, sua primeira parte contém um único CPF de cliente que já está ordenado. A segunda parte consequentemente vai conter n-1 CPF de cliente restantes. A medida que o método segue, a partir de i=2 o i-ésimo valor é enviado do segundo segmento para o primeiro, com isso é inserido na sua posição correta do ArrayList.

Shellsort

um conjunto de CPF de clientes é separado por h posições.O CPF na posição j, vai ser comparado e trocado com o CPF na posição j-h. O procedimento será repetido até h ser igual a 1

Quicksort

No Quicksort é escolhido um CPF de cliente como pivô, com ele é o ArrayList é re-organizado fazendo com que os CPF’s menores que o pivô fiquem de um lado, e os CPF’s maiores que o pivô fiquem do outro. Com recursividade é ordenado a sub lista de CPF’s antes e depois do pivô.

Quicksort c/ Inserção direta

Não conseguimos achar material sobre esse método

VITÓRIA, 2021

3 A EXECUÇÃO DOS ALGORÍTIMOS

Método	Quadro comparativo = Tempo de Execução								
	500			1000			5000		
	Ordenado	Aleatório	Invertido	Ordenado	Aleatório	Invertido	Ordenado	Aleatório	Invertido
Inserção Direta	1,327	5,641	1,084	11,778	5,263	11,858	0,391	59,47	4,328
Shellsort	0,208	1,133	0,166	1,102	0,45	1,253	0,127	1,409	0,115
Quicksort	0,174	0,926	0,219	0,969	0,585	1,072	2,371	1,175	0,147
Quicksort com inserção									

Figura 1: Tabela comparativa de arquivos com até 5000 (cinco mil) entradas

Método	Quadro comparativo = Tempo de Execução					
	10000			50000		
	Ordenado	Aleatório	Invertido	Ordenado	Aleatório	Invertido
Inserção Direta	2,31	18,36	5,726	544,475	392,22	243,879
Shellsort	0,116	0,25	0,141	1,089	1,334	0,514
Quicksort	0,119	0,196	0,169	1,21	1,188	0,711
Quicksort com inserção						

Figura 2: Tabela comparativa de arquivos com 10000 (dez mil) e 50000 (cinquenta mil) entradas

Com o quadro abaixo providenciado pelos materiais da professora foi possível atestar com alguns ajustes de regra de 3, que os métodos Shellsort e Quicksort seguem quase que parelhos alguns valores gerados pelo código. Por exemplo, multiplicar o N=256 da tabela abaixo por 2, para ter um valor de N=512, um valor aproximado do N=500 utilizado em nosso teste. Optamos por esse caminho de comparação porque essa tabela é também em segundos.

Método	*Quadro Comparativo = Tempo de Execução*					
	N = 256			*N = 2048*		
	Ordenado	*Aleatório*	*Invertido*	*Ordenado*	*Aleatório*	*Invertido*
Inserção Direta	0.02	0.82	0.64	0.22	50.74	103.80
Seleção Direta	0.94	0.96	1.18	58.18	58.34	73.46
Bubblesort	1.26	2.04	2.80	80.18	128.84	178.66
Shakersort	0.02	1.66	2.92	0.16	104.44	187.36
Shellsort	0.10	0.24	0.28	0.80	7.08	12.34
Heapsort	0.20	0.20	0.20	2.32	2.22	2.12
Quicksort	0.08	0.12	0.08	0.72	1.22	0.76

VITÓRIA, 2021

Um outro exemplo, método Shellsort com 500 elementos ordenado em comparação com N=256 x 2 da tabela acima também ordenado. No nosso teste dando 0,208 segundos e no da tabela quando 0,20 segundos.
A tabela abaixo também foi utilizada na comparação, nos ajudando a entender melhor as proporções das medidas de tempo.

	Ordenado				***Aleatório***				***Invertido***			
	500	**5000**	**10000**	**30000**	**500**	**5000**	**10000**	**30000**	**500**	**5000**	**10000**	**30000**
Inserção	1	1	1	1	11.3	87	161	–	40.3	305	575	–
Seleção	128	1524	3066	–	16.2	124	228	–	29.3	221	417	–
Shellsort	3.9	6.8	7.3	8.1	1.2	1.6	1.7	2	1.5	1.5	1.6	1.6
Quicksort	4.1	6.3	6.8	7.1	1	1	1	1	1	1	1	1
Heapsort	12.2	20.8	22.4	24.6	1.5	1.6	1.6	1.6	2.5	2.7	2.7	2.9

Os métodos que ficaram com valores muito discrepantes provavelmente se devem à diferença de código, por linhas do nosso código, pode ser também por diferenças de peça de hardware no que tange memória, processador e todo hardware responsável por transferência de dados. Em último caso também pode ser porque nós comparamos os CPF's de um arquivo que pra apenas 1 elemento pode possuir mais 3 ou mais 4 informações, atrasando o tempo do método de comparação.