Algoritmos de Ordenação
Estrutura de Dados II

Pedro Henrique Robadel da Silva Camâra[1], **Filipe Pinheiro da Costa Mello**[1]

[1]Departamento de Computação – Universidade Federal do Espírito Santo (UFES)
R. Felício Alcure, Conceição – 29500-000 – Alegre – ES – Brasil

Abstract. *With an increasing volume of data, it's been made necessary to arrange them efficiently. With that being said, the huge data structure's sorting algorithms are essential for the field, evading the intuitive idea of rearranging in exchange of save time and computer performance, decreasing the computational cost. This article will cover bubble sort, insertion sort, binary insertion, shell sort, heap sort, quick sort*, merge-sort, radix-sort and bucket sort analytically, using the data extracted from the code attached to the research and compare them by running time, number of comparisons and number of swaps. The goal is to conclude about speed and efficacy of each method for to observe in which environment each one will be better.*
**quick sort begin, center and median*

Resumo. *Com um volume grande de dados, se vê a necessidade de organizarlos de forma confiável e eficiente. Com isso, os algoritmos de ordenação de estruturas de dados grandes se fazem essenciais, saindo da ideia mais intuitiva de rearranjo para o maior ganho de tempo e processamento, diminuindo o custo computacional. Este trabalho irá abordar os algoritmos bolha, inserção direta, inserção binária, shellsort, heapsort, quicksort*, merge-sort, radix-sort e bucketsort de forma crítica e analítica, utilizando dados gerados pelo compilador e pelo código anexado ao artigo para compará-los por tempo de execução, número de comparações e número de trocas. O objetivo deste estudo é chegar em uma conclusão sobre a velocidade e eficácia de cada algoritmo, a fim de observar em qual ambiente cada um se destaca.*
**quicksortini, quicksortcentro e quicksortmediana*

1. sobre os algoritmos

A estratégia de cada algoritmo é o fator determinante para sua performance, dentro deste tópico será analisado brevemente cada uma das técnicas.

1.1. Bolha

Um dos métodos mais simples e intuitivos dentre a seleção, ele apenas joga os elementos maiores para as posições finais, tendo garantido ao final da execução que o maior valor estará na última posição. Curiosamente o pior caso e o melhor caso têm a mesma complexidade, $O(n^2)$, ou seja, o algoritmo é pouco eficiente.

1.2. Inserção Direta

Ele avança da esquerda para a direita na estrutura e deixa os elementos mais à esquerda ordenados. Para estruturas já ordenadas possui o custo de $O(n)$.A inserção direta é ideal para bases de dados quase ordenados, ou para adicionar entradas em uma estrutura já ordenada, e obter a estrutura ainda ordenada.

1.3. Inserção Binária

O algoritmo de inserção binária é o aperfeiçoamento do algoritmo algoritmo de inserção direta. Divide a sequêcia destino no seu ponto central, prosseguindo a divisão encontrar o ponto correto de inserção.

1.4. Seleção Direta

Assim como o método bolha, esse algoritimo é bem intuitivo. Ele percorre a base de dados da esqueda para a direita. Para cada posição, ele pecorre todos os elementos a direta em busca do menor elemento para realizar a troca e deixar o elemento selecionado na posição correta. A medida que avança, os elementos a esqueda já permanecem na posição correta.

1.5. Shellsort

O algoritmo passa várias vezes pela base de dados dividindo em bases menores, então é aplicado o método de inserção para ordenar essas bases. O método não é estável, pois para registro de chave iguais, não é sempre que ele deixa na mesma posição relativa.

1.6. Heapsort

1.7. Quicksort

1.8. Radix Sort

1.9. Bucket Sort

2. Performance

Tabela 1. Vetor de 100 Posições

	Ordem Crescente			**Ordem Decrescente**		
Algoritimo	Tempo(s)	Comp.	trocas	Tempo(s)	comparações	trocas
Bolha	0,000023	4950	0	0,000050	4950	4950
Insercao Direta	0,000002	99	0	0,000036	5049	4950
Insercao Binaria	0,000006	480	0	0,000033	573	4950
Seleção Direta	0,000022	4950	0	0,000028	4950	50
Shellsort	0,000004	342	0	0,000003	572	230
Heapsort	0,000015	1081	640	0,000007	944	516
Quicksortini	0,000026	4950	100	0,000015	4900	149
Quicksortcentro	0,000006	480	68	0,000003	386	118
Quicksortmediana	0,000067	4950	100	0,000023	1418	134
Mergesort	0,000016	672	672	0,000007	672	672
Radixsort	0,000013	0	0	0,000006	0	0
Bucketsort	0,000007	0	0	0,000003	0	0

	Ordem Aleatória		
Algoritimo	Tempo(s)	Comp.	Trocas
Bolha	0,000023	4950	2598
Insercao Direta	0,000010	2697	2598
Insercao Binaria	0,000012	529	2598
Seleção Direta	0,000015	4950	96
Shellsort	0,000007	820	478
Heapsort	0,000008	1026	582
Quicksortini	0,000008	615	223
Quicksortcentro	0,000007	438	209
Quicksortmediana	0,000032	500	207
Mergesort	0,000011	672	672
Radixsort	0,000006	0	0
Bucketsort	0,000004	0	0

Tabela 2. Vetor de 1000 posições

	Ordem Crescente			Ordem Decrescente		
Algoritimo	Tempo(s)	Comp.	Trocas	Tempo(s)	comp	trocas
Bolha	0,001594	499500	0	0,002787	499500	499500
Insercao Direta	0,000005	999	0	0,001576	500499	499500
Insercao Binaria	0,000042	7987	0	0,001137	8977	499500
Seleção Direta	0,001328	499500	0	0,001107	499500	500
Shellsort	0,000025	5457	0	0,000034	9377	3920
Heapsort	0,000114	17583	9708	0,000088	15965	8316
Quicksortini	0.001204	499500	1000	0,000965	499000	1499
Quicksortcentro	0,000031	7987	744	0,000025	6996	1244
Quicksortmediana	0,001548	499500	1000	0,000486	128248	1372
Mergesort	0,000093	9976	9976	0,000104	9976	9976
Radixsort	0,000081	0	0	0,000084	0	0
Bucketsort	0,000025	0	0	0,000027	0	0

	Ordem Aleatória		
Algoritimo	Tempo(s)	Comp	Trocas
Bolha	0,002440	499500	240978
Insercao Direta	0,001014	241977	240978
Insercao Binaria	0,000763	8599	240978
Seleção Direta	0,001115	499500	994
Shellsort	0,000094	13873	8416
Heapsort	0,000105	16888	9120
Quicksortini	0,000084	8583	3098
Quicksortcentro	0,000080	7113	2900
Quicksortmediana	0,000477	6797	3034
Mergesort	0,000132	9976	9976
Radixsort	0,000068	0	0
Bucketsort	0,000032	0	0

Tabela 3. Vetor de 10000 posições

Algoritimo	Tempo(s)	Comp.	Trocas	Tempo(s)
Bolha	0,119926	49995000	0	0,224326
Insercao Direta	0,000032	9999	0	0,156269
Insercao Binaria	0,000575	113631	0	0,112581
Seleção Direta	0,105156	49995000	0	0,110008
Shellsort	0,000263	75243	0	0,000446
Heapsort	0,001139	244460	131956	0,001102
Quicksortini	0,094582	49995000	10000	0,094283
Quicksortcentro	0,000327	113631	7952	0,000301
Quicksortmediana	0,098389	49995000	10000	0,029189
Mergesort	0,000909	133616	133616	0,000985
Radixsort	0,000877	0	0	0,000771
Bucketsort	0,000231	0	0	0,000219

	Ordem Aleatória		
Algoritimo	Tempo(s)	Comp.	Trocas
Bolha	0,241743	49995000	24000517
Insercao Direta	0,077722	24010516	24000517
Insercao Binaria	0,057159	119026	24000517
Seleção Direta	0,104932	49995000	9991
Shellsort	0,001525	234162	158919
Heapsort	0,001411	235785	124569
Quicksortini	0,001093	124035	38358
Quicksortcentro	0,001000	104103	36429
Quicksortmediana	0,005592	109084	37374
Mergesort	0,001600	133616	133616
Radixsort	0,000814	0	0
Bucketsort	0,000268	0	0

3. Conclusão

Referências

Backes, A. (2018). Youtube[ed] aula 124 - ordenação bucketsort. `https://www.youtube.com/watch?v=4J89y2Pv_qM&t=284s`. Acessado em 06/06/2019.

BUDIMAN M., ZAMZAMI, E. R. D. (2017). Multi-pivot quicksort: an experiment with single, dual, triple, quad, and penta-pivot quicksort algorithms in python. *Conference Series: Materials Science and Engineering. [S.l.]*.

et al, J. (2013). Radix sort. *International Journal of Emerging Research in Management Technology*.

Jackson É. G. Souza, João V. G. Ricarte, N. C. A. L. (2017). Algoritmos de ordenação: Um estudo comparativo.