Raiza Arteman

Relatório sobre Algoritmos de Ordenação

Dourados - MS

Outubro, 2014

Raiza Arteman

Relatório sobre Algoritmos de Ordenação

Relatório sobre algoritmos de ordenação, tempos de execução e comparação dos mesmos.

Universidade Estadual de Mato Grosso do Sul – UEMS

Ciência da Computação

Algoritmos e Estruturas de Dados II

Dourados - MS

Outubro, 2014

Lista de tabelas

Sumário

Introdução

Ordenar conjuntos é uma tarefa muito importante no campo da computação para diversos fins, o mais direto a citar é o caso de busca em que sempre é mais eficiente em conjuntos cujas as chaves a serem buscadas encontram-se ordenadas. Existem vários algoritmos de ordenação, entre eles o Heap Sort, Quick Sort, Merge Sort, Insertion Sort e Bubble Sort.

1 Algoritmos de Ordenação Analisados

1.1 Bubble Sort

O Bubble Sort é um algoritmo simples, bastante intuitivo e de fácil implementação, talvez justamente a essa simplicidade deva-se o seu baixo desempenho, a complexidade de tempo do Bubble Sort é de $O(n^2)$. O algoritmo consiste em comparar cada elemento do vetor com o próximo e troca-los caso seja necessário. Suponha o vetor V formado pelos seguintes elementos: $a_0, a_1, \ldots, a_{n-1}, a_n$ o algoritmo compara sempre o elemento a_i com o elemento a_{i+1} caso o segundo seja maior que o primeiro então estes dois elementos serão trocados, assim até que todos os elementos tenham sido comparados. Esta comparação torna-se desnecessária caso o vetor já esteja ordenado, para melhorar o desempenho do algoritmo nestes casos pode-se usar um critério de parada que verifica se na primeira iteração foi feita alguma troca no vetor, caso não tenha sido, é porque este já está ordenado, outra mudança no algoritmo é guardar a posição da última troca no vetor, pois os elementos posteriores a esta posição já estão ordenados (SZWARCFITER; MARKEZON, 2012).

1.2 Insertion Sort

O Insertion Sort também é um algoritmo muito simples, ele consiste em inserir os elementos no lugar correto. Ao início de cada iteração um elemento é comparado com os demais e inserido no lugar correto. A complexidade do algoritmo é $O(n^2)$ no pior caso e $O(n)$ no melhor caso.

1.3 Merge Sort

O princípio do Merge Sort dividir a lista a ser ordenada e intercalar estas quandi já estiverem ordenadas. Um ponto fraco deste algoritmo é a necessidade de um vetor auxiliar para a ordenação, entretanto em questão de tempo é um algoritmo bem eficiente com complexidade de pior caso de $O(nlogn)$.

1.4 Heap Sort

O Heap Sort utiliza um heap de máximo para a ordenação. O algoritmo foi criado por fulano . Seja $\{a_0, a_1, \ldots, an-1, a_n\}$ uma lista de elementos a serem ordenados o algoritmo constrói um heap com os elementos, onde o primeiro elemento do heap será o maior da lista, este elemento é trocado com o último, a lista passa a ser considerada de

tamanho $n-1$, e este processo é repetido até $n=1$. O Heap Sort é um algoritmo eficiente com complexidade $O(nlogn)$.

1.5 Quick Sort

1.5.1 O Pivô

Um dos problemas do Quick Sort é a escolha do pivô, pois deste depende o desempenho do algoritmo. A escolha mais comum de pivô é o primeiro elemento da lista, porém se a lista não for aleatória esta escolha pode dividir a lista muito desigualmente. Outra opção para o pivô seria escolher um aleatoriamente, mas este torna-se um problema na execução de gerar um número aleatório quando o tamanho da lista vai diminuindo. Por fim a melhor escolha de pivô seria a mediana que consiste em escolher o primeiro elemento, o elemento do meio do vetor e o último, ordená-los e usar como pivô a mediana dos três elementos.

2 Tempos de execução

Os tempos de execução em segundos dos algoritmos.

Entrada	Crescente	Decrescente	Aleatório
10	2,7021E-06	1,9073E-06	2,6226E-06
100	3,95775E-05	3,25839E-05	3,76701E-05
1000	0,000615676	0,00055337	0,000623385
10000	0,003820976	0,004282713	0,004015287
100000	0,047504425	0,044364055	0,051259677
1000000	0,562063058	0,276557366	0,633157412
10000000	6,621633053	0,518997669	8,726442973
100000000	76,01114734	2,909035365	123,0887283
500000000	412,5243237	13,54383564	789,1410852

Tabela 1 – Tempo de execução do algoritmo Heap Sort.

Entrada	Crescente	Decrescente	Aleatório
10	0	0,000002	0,000001351
100	0,000001	0,000062	6,50088E-05
1000	0,000013	0,004636	0,003847996
10000	0,000126	0,318326	0,347544273
100000	0,000592	31,838436	39,13753502
500000	146,998457	797,419381	982,0974757

Tabela 2 – Tempo de execução do algoritmo Bubble Sort com melhorias.

Entrada	Crescente	Decrescente	Aleatório
10	1,0331E-06	1,0331E-06	9,537E-07
100	4,89553E-05	4,20411E-05	7,70092E-05
1000	0,002621333	0,003514687	0,0040706
10000	0,224333286	0,348394791	0,372662942
100000	28,35690498	34,7604049	40,05228496
500000	612,4568298	850,4239405	1005,146459

Tabela 3 – Tempo de execução do algoritmo Bubble Sort.

Entrada	Crescente	Decrescente	Aleatório
10	6,358E-07	0,000001351	3,179E-07
100	9,537E-07	3,31402E-05	8,9804E-06
1000	9,6957E-06	0,003109296	0,00077343
10000	9,55264E-05	0,150158087	0,078080972
100000	7,55574131	14,99426762	7,851370414
500000	68,01354663	375,027132	192,8214437

Tabela 4 – Tempo de execução do algoritmo Insertion Sort.

Entrada	Crescente	Decrescente	Aleatório
10	1,5895E-06	1,6689E-06	1,9868E-06
100	1,26362E-05	1,20004E-05	1,77224E-05
1000	0,000159423	0,000159343	0,000265996
10000	0,001271645	0,001743237	0,00262634
100000	0,011003335	0,010996262	0,019776662
1000000	0,127586603	0,126383384	0,229385058
10000000	1,475922267	1,453179598	2,663177967
100000000	16,63652158	16,43337401	30,21646039
500000000	147,935184	109,5326704	169,1400704

Tabela 5 – Tempo de execução do algoritmo Merge Sort.

Entrada	Crescente	Decrescente	Aleatório
10	0,0000028	0,0000421	0,00000321
100	0,0000113	0,00000343	0,000000021
1000	0,00000932	0,00042114	0,000000543
10000	0,0045432	0,26743122	0,0055433788
100000	7,675434529	13,99549983	6,76543314
500000	68,9873463	372, 9654374323	189, 99745754

Tabela 6 – Tempo de execução do algoritmo Quick Sort com pivô primeiro elemento.

Entrada	Crescente	Decrescente	Aleatório
10	0,000001	0,000001	0,000001
100	0,000013	0,000013	0,000014
1000	0,000103	0,000432	0,000101
10000	0,002438	0,003981412	0,002438
100000	0,020021	0,03657804	0,020022
1000000	0,160580	0,30412013	0,178659
10000000	4,9564332	0,41244426	4,763123
100000000	75,734814	17,09035365	112,465328
500000000	402,643237	100,323264	602,143112

Tabela 7 – Tempo de execução do algoritmo Quick Sort com pivô mediano.

Entrada	Crescente	Decrescente	Aleatório
10	0,000001	0,000001	0,000001
100	0,000013	0,000043	0,000014
1000	0,000103	0,000543	0,000920
10000	0,00645778	0,004273456	0,00545920
100000	0,0 7975454	0,08763233	0,06452789
1000000	0,1846728	0,197643456	0,1786594
10000000	1,9976467	1,09755636	1,8778903
100000000	20,734814	19,090355	19,986528
500000000	100,6312237	100,034264	100,655712

Tabela 8 – Tempo de execução do algoritmo Quick Sort com pivô aleatorio.

2.1 Estimativas

Entrada	Crescente	Decrescente	Aleatório
1000000	0,00450559	784,613908	394,4464009
10000000	0,044695843	7958,363039	4000,878219
10000000	0,446598374	79695,85435	40065,1964
5000000	2,232831845	398529,1491	200351,055

Tabela 9 – Estimativas de tempo para Insertion Sort.

Entrada	Crescente	Decrescente	Aleatório
1000000	1987,579187	1116,278866	1709,733194
10000000	20160,38976	11322,53853	17342,16372
10000000	201888,4955	113385,1351	173666,4689
50000000	1009568,966	566996,6756	868441,1589

Tabela 10 – Estimativas de tempo para Bubble Sort.

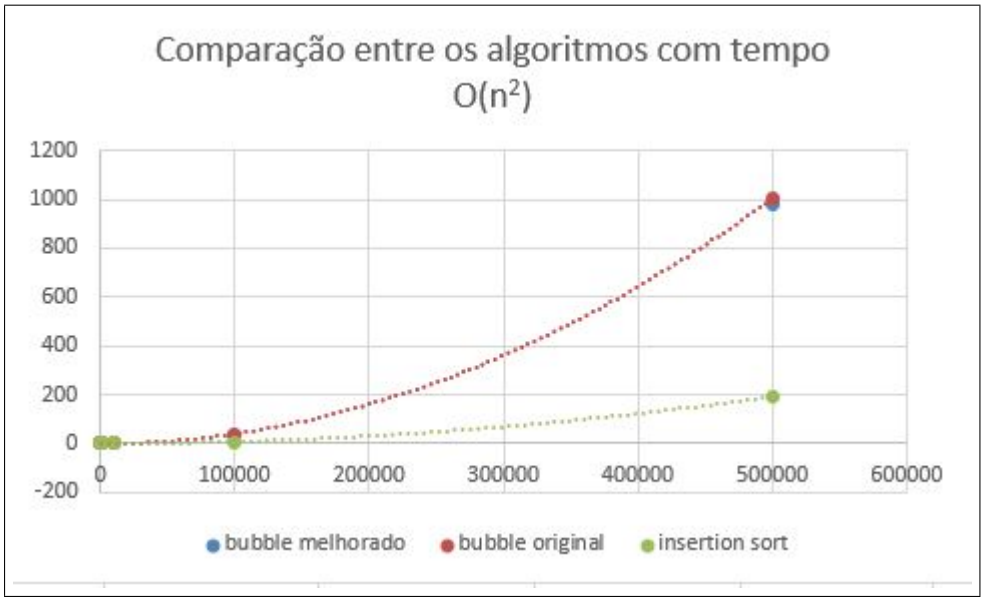
Entrada	Crescente	Decrescente	Aleatório
1000000	0,007545277	1587,539107	1883,526249
10000000	0,074916653	16100,8206	19105,10017
100000000	0,748630417	161233,6356	191320,8394
500000000	3,742913814	806268,3687	956724,1247

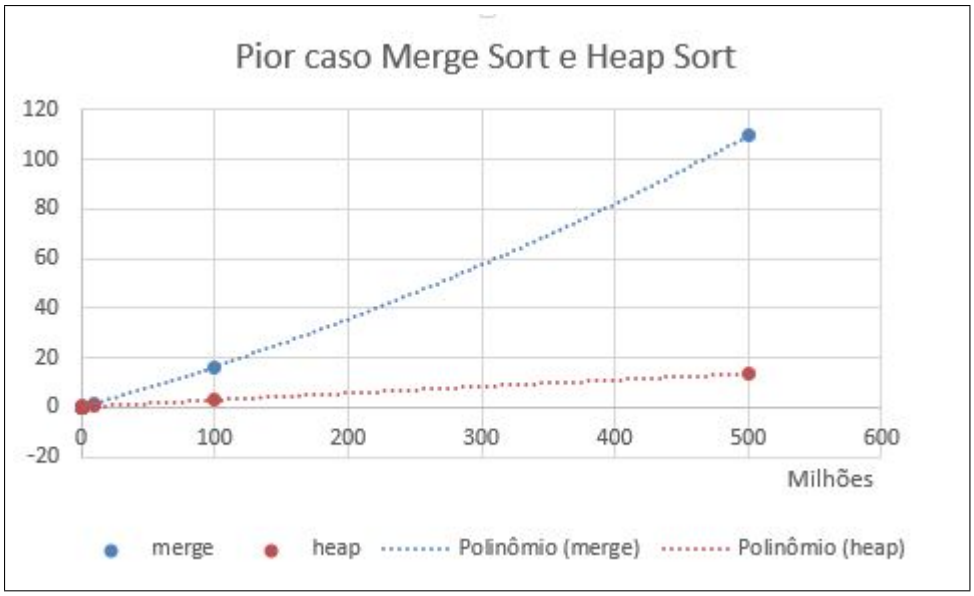
Tabela 11 – Estimativas de tempo para Bubble Sort com melhorias.

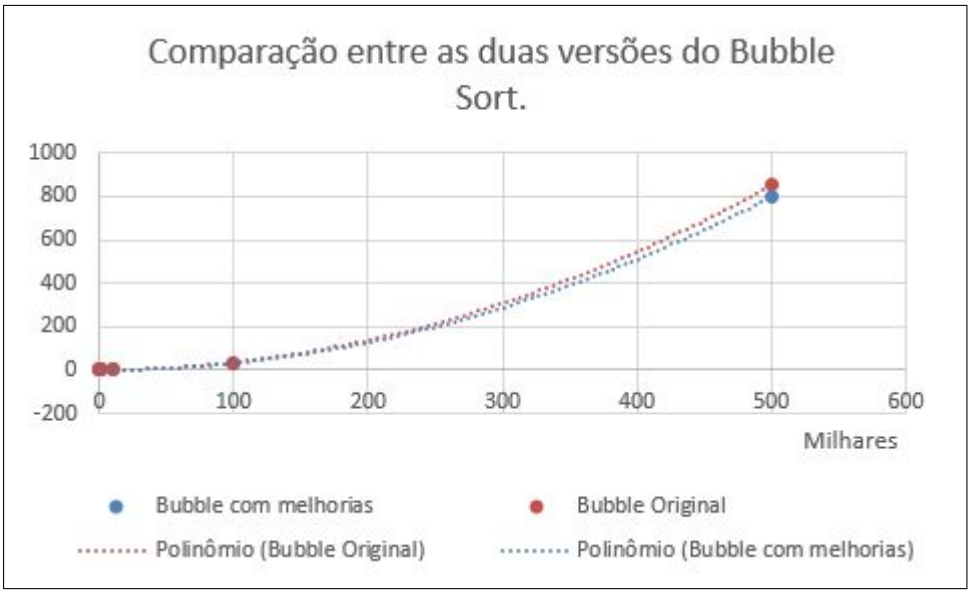
Entrada	Crescente	Decrescente	Aleatório
1000000	526,9620373	0,171271149	542,2686338
10000000	5342,568149	1,704788374	5498,785702
100000000	53498,62926	17,03996062	55063,95639
500000000	267525,5676	85,19628173	275353,6039

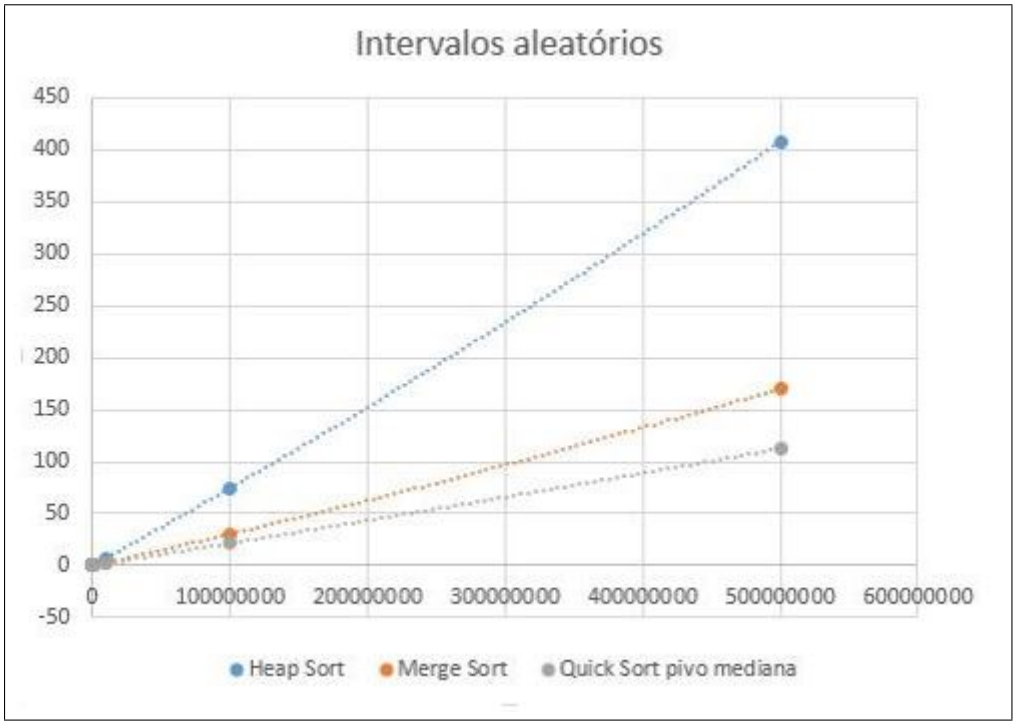
Tabela 12 – Estimativas de tempo para Quick Sort com o primeiro elemento pivô.

3 Gráficos


Comparação entre os algoritmos com tempo
O(n²)
1200
1000
800
600
400
200
0
-200
0
100000
200000
300000
400000
500000
600000
bubble melhorado
bubble original
insertion sort


Pior caso Merge Sort e Heap Sort
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Milhões
merge
heap
Polinômio (merge)
Polinômio (heap)


Comparação entre as duas versões do Bubble Sort.
1000
800
600
400
200
0
-200
0
100
200
300
400
500
600
Milhares
Bubble com melhorias
Bubble Original
Polinômio (Bubble Original)
Polinômio (Bubble com melhorias)


Intervalos aleatórios
450
400
350
300
250
200
150
100
50
0
-50
0
100000000
200000000
300000000
400000000
500000000
600000000
Heap Sort
Merge Sort
Quick Sort pivo mediana

Considerações finais

A ordenação é necessária como pré-processamento em diversas aplicações. Os algoritmos de ordenação aqui testados e estudados se mostram eficientes em vários casos, devendo-se levar em consideração alguns fatores ao optar por um destes algoritmos, como facilidade de implementação, memória e principalmente o tamanho da tabela a ser ordenada. Os algoritmos Bubbe Sort e Insertion Sort embora tenham o tempo de execução $O(n^2)$ são eficientes em tabelas pequenas, com até mil elementos, são estáveis e promovem ordenação *inplace*, sem a necessidade de vetor auxiliar. Os algoritmos de tempo $O(nlogn)$ são realmente muito eficientes para a ordenação de grandes quantidades de elementos, porém são mais complexos para implementar e ocupam memória, pois usam vetores auxiliares e criam uma pilha de chamadas recursivas. Entre os algoritmos Quick Sort, Merge Sort e Heap Sort, o segundo e o terceiro são mais viáveis, pois o Quick deve-se levar em consideração o pivô se este cair no pior caso , quando o pivô é o primeiro elemento e a tabela esta ordenada em ordem inversa a desejada, o tempo de execução do algoritmo é $O(n^2)$, no caso do pivô aleatório gasta-se tempo na execução para encontrar este, o melhor caso é a mediana de três. Mas com todos estes casos a se ponderar posso dizer que a melhor escolha em relação custo - beneficio é o Heap Sort. É possível ver uma pequena comparação de algoritmos de ordenação no site : <http://nicholasandre.com.br/sorting/t>.

Referências

SZWARCFITER, J. L.; MARKEZON, L. *Estrutura de dados e seus algoritmos.* 3nd. ed. Rio de Janeiro - RJ: LTC, 2012. Citado na página 5.