Maxley Soares da Costa 11911BCC038
Joao Vitor Afonso Pereira 11911BCC037

Resultados Algoritmos de ordenação

l1=[1..2000]
l2=[2000,1999..1]
l3=l1++[0]
l4=[0]++l2
l5=l1++[0]++l2
l6=l2++[0]++l1
l7=l2++[0]++l2
x1=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
x2=[20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]
x3=[11,12,13,14,15,16,17,18,19,20,1,2,3,4,5,6,7,8,9,10]
x4=[10,9,8,7,6,5,4,3,2,1,20,19,18,17,16,15,14,13,12,11]
x5=[11,12,13,14,15,5,4,3,2,1,16,17,18,19,20,10,9,8,7,6]
x6=[1,12,3,14,5,15,4,13,2,11,6,17,8,19,20,10,9,18,7,16]
x7 = [20,8,2,11,13,3,7,18,14,4,16,10,15,1,9,17,19,12,5,6]

Todos os resultados em valores abaixo são a média de 3 comparações iguais, ou seja, cada algoritmo rodou uma mesma lista 3 vezes e o resultado final foi uma média dessas medidas.

Exercício 1: Analisando o Bubble Sort

	Bubble Sort Original	Bubble Sort Variação 1	Bubble Sort Variação 2	Número de Trocas
l1	4.29	0.32	0.32	0
l2	5.32	4.78	4.04	1999000
l3	4.26	4.45	3.58	2000
l4	5.03	5.00	3.44	1999000
l5	19.98	19.73	15.53	4000000
l6	20.59	10.61	10.49	4000000
l7	19.53	17.77	15.12	5999000

*Tabela 1: Transparência da medidas de tempo em segundos para o algoritmo Bubble Sort.

Temos que o Bubble sort é um algoritmo O(n^2) . Logo percebemos que o mesmo leva um tempo considerável para ordenar listas grandes como a l7 e dentre as variações temos que a original é a que demanda um maior tempo para ordenar listas grandes. Em relação à variação 1, na qual a parada do algoritmo é antecipada quando não ocorre troca, vemos que há uma melhora no tempo em relação à listas que não possuem muitos elementos fora de ordem e consequentemente chegam em um estado de ordenação mais rápido. Com relação à variação 2 podemos perceber um tempo ainda menor para algumas listas, isso se deve ao fato de economizarmos tempo ignorando elementos já ordenados no fim da lista.
É trivial notar que listas maiores exigem um número maior de comparações e caso haja muitas trocas, é notável que mesmo a variação 2 leva bastante tempo para ordenar tal lista. Isso se deve ao fato de que a parada antecipada não surte muito efeito uma vez que há elementos de valor pequeno no final da lista.

Exercício 2: Analisando o selection sort

	Selection Sort Original	Selection Sort Variação 1	Selection Sort Variação 2	Número de Trocas
x1	0.00			0
x2	1.20			19
x3	0.05			10
x4	0.58			18
x5	0.61			10
x6	0.19			13
x7	0.42			19

Temos que para o selection sort o código original com contador implementado possui tempos relativamente rápidos para listas pequenas, porém para listas maiores o tempo se torna muito grande devido à maneira com que o mesmo foi implementado o que o torna inapropriado para grandes listas. Note que os testes para esse algoritmo foram feitos utilizando as listas x que são menores.
Podemos notar que o número de trocas é diferente para cada lista aplicada sobre o algoritmo original, uma vez que o número de trocas depende de como os elementos da lista estão dispostos. Por exemplo, podemos dar sorte de encontrar muitos elementos em sua posição ordenada apesar da lista estar desordenada.

Exercício 3: Analisando o insertion sort

	Insertion Sort Original	Insertion Sort Variação 1	Número de Comparações
I1	0.37		0
I2	4.89		1999000
I3	0.34		2000
I4	3.01		1999000
I5	6.36		4002000
I6	6.46		4002000
I7	8.47		6001000

*Tabela 2: Transparência da medidas de tempo em segundos para o algoritmo Insertion Sort.

O Insertion Sort é um algoritmo também com complexidade O(n^2) em que ao ser comparado ao Bubble Sort não vemos uma mudança drástica no número de comparações, porém vemos uma significativa melhora no tempo de execução. Para o código original vemos que o mesmo é pelo menos 2 vezes mais rápido do que a variação mais rápida do Bubble Sort para listas grandes.

Exercicio 4: Analisando o quick sort

	Quick Sort Original	Quick Sort Variação 1	Quick Sort Variação 2	Número de Comparações
I1	2.94	0.36	0.32	1999
I2	2.80	0.34	0.33	1999
I3	2.92	0.33	0.33	2000
I4	2.52	0.33	0.33	2000
I5	5.60	0.65	0.67	4000
I6	5.09	0.67	0.65	4000
I7	4.97	0.67	0.65	4000

*Tabela 3: Transparência da medidas de tempo em segundos e numero de comparações para o algoritmo Quick Sort.

O quick sort é o primeiro algoritmo O(log n) analisado aqui. Podemos perceber que seu tempo de execução para todas as listas cai drasticamente. Analisando suas variações, é possível notar que a variação 1 que utiliza uma função responsável por dividir a lista a partir

de um pivô qualquer (Aqui utilizamos o primeiro elemento da lista) por sua vez é bem mais rápida do que o algoritmo original. A variação 2 que utiliza o menor entre os 3 primeiros elementos como pivô também é muito mais rápida do que o algoritmo original, mas não apresentou melhorias tão significativas em comparação à variação 1.

Exercicio 5: Analisando o merge sort

	Merge Sort	Número de Comparações
l1	0.43	1023
l2	0.44	11088
l3	0.44	1023
l4	0.46	10096
l5	0.74	2048
l6	0.75	22178
l7	0.73	22178

*Tabela 4: Transparência da medidas de tempo em segundos para o algoritmo Merge Sort.

Temos que o Merge Sort é o segundo algoritmo com complexidade O(n * log n), logo ao ser comparado com o Quick Sort temos que o merge de forma pura alcança velocidades comparáveis a variação 2 do Quick sort, ou seja, o Merge é o algoritmo de ordenação mais eficaz, visto que concluiu a ordenação de forma rápida. Podemos notar também que o tempo de execução desse algoritmo depende apenas do tamanho da lista, uma vez que para listas de tamanhos iguais, independentemente da ordem dos elementos, o algoritmo gasta o mesmo tempo para ordenar.