LG2 - Relatório: Trabalho de Ordenação

Mariana Cristina Sena Silva – SP3065677 – 213B

- ## Dados da máquina

O experimento foi realizado em uma máquina com processador Intel(R) Core (TM) i3-5005U CPU @ 2.00GHz 2.00 GHz, com 4,00 GB de RAM instalada, sendo seu sistema operacional de 64 bits, processador baseado em x64.

- ## BubbleSort

É um algoritmo de ordenação mais simples, que permite que valores mais baixos ou mais altos se dirijam até o topo, sendo que consiste em percorrer os N elementos de um vetor, para cada vez percorrida, todos os elementos são comparados com o seu próximo, para verificar se estão na ordem desejada. Com uma complexidade de O(n^2), esta ordenação é a mais lenta. A vantagem é que é um dos algoritmos de ordenação mais fáceis de entender e implementar do zero. Ele é pouco eficiente e é razoável ao executar algoritmos de ordenação em computadores com recursos de memória notavelmente limitados.

- ## SelectionSort

A ordenação por seleção consiste em selecionar o menor item e colocar na primeira posição, selecionar o segundo menor item e colocar na segunda posição, executando esta mesma ação até o último elemento. Para todos os casos, o SelectionSort possui complexidade $C(n) = O(n^2)$ e não é um algoritmo estável.

- ## InsertionSort

É a ordenação por inserção e é um algoritmo simples para um pequeno número de elementos. Ele percorre um vetor de elementos da esquerda para a direita e à medida que avança vai ordenando os elementos à esquerda, ou seja, se compara o elemento chave com os elementos anteriores. Se os elementos anteriores são maiores do que o elemento chave, então você move o elemento anterior para a próxima posição. Possui complexidade $C(n) = O(n)$ no melhor caso e $C(n) = O(n^2)$ no caso médio e pior caso. É considerado um método de ordenação estável.

- **ShellSort**

O ShellSort é uma melhoria do InsertionSort. Este método não é estável. Ele permite a troca de registros distantes um do outro, diferente do algoritmo de ordenação por inserção que possui a troca de itens adjacentes para determinar o ponto de inserção. A complexidade de pior caso do algoritmo ShellSort para qualquer sequência com O (log c n) passos, para qualquer constante , é de O (n 2 log n). Os elementos são separados em h posições e são ordenados: o elemento na posição x é comparado e trocado (caso satisfaça a condição de ordenação) com o elemento na posição x-h. Este processo repete até h=1, quando esta condição é satisfeita o algoritmo é equivalente ao método de inserção.

- **MergeSort**

É um exemplo de algoritmo de ordenação que faz uso da estratégia "dividir para conquistar". É um método estável e possui complexidade C(n) = O (n log n) para todos os casos. Ele divide o array de entrada em duas metades, chama a si mesmo pelas duas metades e depois mescla as duas metades já ordenadas, ou seja, ele divide o problema em pedaços menores, resolve cada pedaço e depois junta os resultados. O vetor será dividido em duas partes iguais, que serão cada uma divididas em duas partes, e assim até ficar um ou dois elementos cuja ordenação é trivial. Para juntar as partes ordenadas os dois elementos de cada parte são separados e o menor deles é selecionado e retirado de sua parte. Em seguida os menores entre os restantes são comparados e assim se prossegue até juntar as partes.

- **QuickSort**

É o método de ordenação interna mais rápido que se conhece, sendo ele um eficiente algoritmo de ordenação que emprega a técnica de divisão e conquista. Quanto a sua complexidade, quando produz consistentemente 2 subarrays com uma grande diferença em termos de tamanho, o algoritmo de ordenação rápida pode atingir a complexidade temporal de é O (n log(n)), por outro lado, se o algoritmo, que seleciona o elemento pivô dos arrays de entrada, produz consistentemente 2 subarrays com uma grande diferença em termos de tamanho, o algoritmo de ordenação rápida pode atingir a complexidade temporal de pior caso de O(n^2). Ele não é um método de ordenação estável. Resumidamente, a operação do algoritmo pode ser baseada na estratégia de: dividir a lista de entrada em duas sub-listas a partir de um pivô, para em seguida realizar o mesmo procedimento nas duas listas menores até uma lista unitária.

- **Relatório tabelado com os registros dos experimentos**

Semente 13	Tempo em ms (milissegundos)					
Vetores	BubbleSort	SelectionSort	InsertionSort	ShellSort	MergeSort	QuickSort
10.000	181	65	44	12	3	5
30.000	1517	381	350	14	15	14
90.000	13881	3223	3077	22	41	29
270.000	118752	27821	27382	65	82	43
810.000	----------	----------	----------	178	190	102
2.430.000	----------	----------	----------	574	514	299
65.610.000	----------	----------	----------	22396	11867	18921

Semente 21	Tempo em ms (milissegundos)					
Vetores	BubbleSort	SelectionSort	InsertionSort	ShellSort	MergeSort	QuickSort
10.000	129	40	45	5	2	1
30.000	1232	347	88	4	8	2
90.000	11407	3335	776	12	12	8
270.000	111198	32990	8027	39	31	25
810.000	----------	----------	----------	169	161	132
2.430.000	----------	----------	----------	558	497	267
65.610.000	----------	----------	----------	24259	120098	15130

Semente 55	Tempo em ms (milissegundos)					
Vetores	BubbleSort	SelectSort	InsertionSort	ShellSort	MergeSort	QuickSort
10.000	139	38	10	1	2	1
30.000	1234	394	89	4	4	2
90.000	11857	3276	792	12	15	8
270.000	112244	33048	7713	45	43	27
810.000	----------	----------	----------	187	188	119
2.430.000	----------	----------	----------	623	513	287
65.610.000	----------	----------	----------	22126	11961	19607

Semente 89	Tempo em ms (milissegundos)					
Vetores	BubbleSort	SelectionSort	InsertionSort	ShellSort	MergeSort	QuickSort
10.000	122	41	10	1	3	1
30.000	1239	401	81	4	6	3
90.000	11635	3396	782	11	13	7
270.000	111506	32474	7635	43	41	27
810.000	----------	----------	----------	179	242	131
2.430.000	----------	----------	----------	611	500	273
65.610.000	----------	----------	----------	23562	12072	29101

Semente 144	Tempo em ms (milissegundos)					
Vetores	BubbleSort	SelectionSort	InsertionSort	ShellSort	MergeSort	QuickSort
10.000	217	35	12	2	3	1
30.000	1284	377	88	5	4	2
90.000	11749	3660	841	13	18	12
270.000	118466	27777	27356	49	71	51
810.000	---------	---------	---------	194	198	109
2.430.000	---------	---------	---------	567	549	304
65.610.000	---------	---------	---------	2510	12101	28598

- **Conclusão**

Os tempos entre um método de ordenação e outro mudam bastante, entretanto, numa questão analítica, o método QuickSort é o que apresenta melhores resultados no geral, sendo o BubbleSort o método com o menor desempenho, e isso pelo fato de ele ser o menos eficiente. A semente, que é como se fosse o "ponto de partida" para a geração de uma sequência de números aleatórios, definiu a variação de todos os tempos em todos os métodos, com isso, não foi identificável exatamente um padrão, senão pelo aumento de milissegundos concomitante a quantidade de vetores na maioria dos métodos, mas esse tempo é determinado pela sua complexidade e na análise do nível dos casos (se é um bom, médio ou pior caso).