UTFPR - Universidade Tecnológica Federal do Paraná
Campus Campo Mourão

Aluno: *Jessé Pires Barbato Rocha*
RA: *2149389*
Disciplina: *Algoritmos e Estrutura de Dados 2*

ADNP - SEMANA 2

A)

Tempo Gasto Na Ordenação (em segundos):

	1000	10000	100000	500000
BubbleSort	$7,8x10^{-3}$	$579,616x10^{-3}$	$66.723,122x10^{-3}$	$1.663.737,183x10^{-3}$
InsertionSort	$1,191x10^{-3}$	$119,909x10^{-3}$	$12.633,122x10^{-3}$	$3.000,177x10^{-1}$
SelectionSort	$5,165x10^{-3}$	$204,893x10^{-3}$	$20.322,504x10^{-3}$	$53.367,511x10^{-2}$
MergeSort	$7,34x10^{-4}$	$3,282x10^{-3}$	$4,098x10^{-2}$	$214,808x10^{-3}$
QuickSort	$4,68x10^{-4}$	$2,354x10^{-3}$	$30,418x10^{-3}$	$2,248x10^{-3}$

B)

Tempo Gasto (em segundos), Com O Vetor Já Ordenado

	1000	10000	100000	500000
BubbleSort	$3,01x10^{-3}$	$225,385x10^{-3}$	$23.013,918x10^{-3}$	$1.663.737,183x10^{-3}$
InsertionSort	$1,214x10^{-3}$	$119,044x10^{-3}$	$12.536,123x10^{-3}$	$3.000,177x10^{-1}$
SelectionSort	$2,079x10^{-3}$	$203,482x10^{-3}$	$20.310,144x10^{-3}$	$517657,837x10^{-3}$
MergeSort	$5,35x10^{-4}$	$2,134x10^{-3}$	$27,449x10^{-3}$	$185,826x10^{-3}$
QuickSort	$8,646x10^{-3}$	$523,907x10^{-3}$	$52.503,922x10^{-3}$	$1.346.252,197x10^{-3}$

C) De acordo com os resultados obtidos nos itens a e b, responda as perguntas a seguir:

i) Qual algoritmo você usaria em um vetor que está praticamente ordenado, ou seja, tem apenas alguns elementos fora do lugar? Por quê?

R: Se também forem levados em consideração os algoritmos *BubbleSort*, *InsertionSort* e *SelectionSort*, a melhor opção para essa configuração do vetor, é o algoritmo de ordenação por inserção (*InsertionSort*), que demonstrou ser o mais eficiente nos testes feitos. Isso é justificado pelo fato de que, neste algoritmo, uma alteração só é feita no vetor quando se encontra a posição correta do maior elemento em questão.

Já se forem levados em consideração apenas os algoritmos *MergeSort* (ordenação por intercalação), e o *QuickSort* (ordenação rápida), a melhor opção para este caso, vide testes, é o *MergeSort*. Em casos onde o vetor está quase ordenado ou ordenado, o algoritmo de ordenação rápida perde eficiência de maneira considerável.

ii) Qual algoritmo você usaria para ordenar um vetor que está ordenado em ordem decrescente? Por quê? (vamos fazer de conta que essa é a melhor forma de ordenar um vetor ordenado em ordem decrescente)

R: Para esta configuração do vetor, o algoritmo que se mostrou mais eficiente nos testes foi o *MergeSort*. Isso pode ser justificado pela complexidade da ordenação por intercalação ser sempre a mesma em todos os casos.

iii) Qual algoritmo você usaria em um vetor que você não faz idéia sobre a ordem dos elementos? Por quê?

R: No caso de ordenar um vetor totalmente desconhecido, usaria o algoritmo *MergeSort* (ordenação por intercalação), pois em todos os casos, sua complexidade é $\theta(n \lg n)$. Mesmo que uma implementação eficiente deste algoritmo seja mais trabalhosa, pelo menos se terá certeza de um bom desempenho em todas as situações.

iv) Você usaria os algoritmos ineficientes em alguma situação? Se sim, qual deles você usaria? Em que situação?

R: Sim, usaria o algoritmo de ordenação por inserção. Como respondido em i (no caso os algoritmos ineficientes forem levados em consideração), se o vetor estiver quase ordenado, o *InsertionSort* custa menos tempo.