

# Escola Superior Dom Helder Câmara

Escola de Engenharia

Disciplina: Algoritmo e Estruturas de Dados II

Professor: Diego Silva Caldeira Rocha

Lista 6, 7 – Métodos de ordenação

Para cada problema proposto postar as soluções no Moodle. Para isto, compacte em único  
arquivo ou conjunto das soluções (os arquivos com extensão .JAVA e docx, pdf..).

1)Implemente os algoritmos em JAVA: BubblleSort, SelectionSort ,InsertSort, Mergesort, QuickSort ,ShellSort e HeapSort. Para o tipo **DOUBLE**.

O código a seguir calcula o tempo gasto para executar uma determinada função. Modifique-o para que possa mensurar o tempo estimado para a ordenação dos métodos implementados. Além disso, contém uma função de geração de valores aleatórios e decrescente. DICA: não se esqueça de randomizar o vetor antes de inicializar a contagem do tempo para cada algoritmo.

public static void main(String[] args) {

double [] vet= gerarAleatorio(10000, 1, 50000);

long startTime = System.nanoTime();

// Coloque sua função aqui de ordenação

suaFuncao(...);

long endTime = System.nanoTime();

// Calcula o tempo decorrido em milissegundos

long elapsedTimeInMillis = (endTime - startTime) ;

System.out.println("Tempo decorrido: " + elapsedTimeInMillis + " nanosegundos"); }

public static void suaFuncao(...) {

// Implemente sua função aqui

}

// Método para gerar um vetor de valores aleatórios double dentro de um intervalo

public static double[] gerarAleatorio(int tamanho, double minimo, double maximo) {

double[] vetorAleatorio = new double[tamanho];

Random random = new Random();

for (int i = 0; i < tamanho; i++) {

vetorAleatorio[i] = minimo + (maximo - minimo) \* random.nextDouble(); // valores dentro do intervalo

}

return vetorAleatorio;

}

// Método para gerar um vetor decrescente com aleatorização da parte fracionada

public static double[] gerarDecrescenteAleatorio(int tamanho) {

double[] vetorDecrescenteAleatorio = new double[tamanho];

Random random = new Random();

for (int i = 0; i < tamanho; i++) {

double valor = tamanho - i; // gera o valor decrescente

// Aleatoriza a parte fracionada

double parteFracionadaAleatoria = random.nextDouble();

valor += parteFracionadaAleatoria;

vetorDecrescenteAleatorio[i] = valor;

}

return vetorDecrescenteAleatorio;

}

2) Execute os algoritmos e anote os resultados na tabela a seguir. obs: execute o algoritmo em uma mesma máquina.

Tabela Vetor já ordenado

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tamanho do Vetor | BublleSort | SelectionSort | InsertSort | Mergesort | Quicksort | Shellsort | HeapSort |
| 10000 | 51.861.000 | 58.144.300 | 83.700 | 4.647.000 | 222.800 | 370.800 | 407.700 |
| 15000 | 116.451.100 | 92.310.600 | 96.500 | 6.980.200 | 432.400 | 458.300 | 609.600 |
| 30000 | 460.647.000 | 357.791.900 | 299.200 | 8.205.400 | 1.053.500 | 869.900 | 758.800 |

Tabela 2 Vetor Aleatório

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tamanho do Vetor | BublleSort | SelectionSort | InsertSort | Mergesort | Quicksort | Shellsort | HeapSort |
| 10000 | 140.738.700 | 38.384.100 | 43.327.700 | 6.585.800 | 307.000 | 3.194.700 | 496.800 |
| 15000 | 318.519.400 | 89.756.200 | 56.166.300 | 7.714.800 | 405.000 | 4.481.600 | 701.000 |
| 30000 | 1.220.768.700 | 368.414.600 | 227.180.000 | 14.959.900 | 933.000 | 5.854.800 | 1.385.100 |

Tabela 3 Vetor Decrescente

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tamanho do Vetor | BublleSort | SelectionSort | InsertSort | Mergesort | Quicksort | Shellsort | HeapSort |
| 10000 | 27.621.700 | 40.685.100 | 40.424.100 | 4.739.400 | 597.800 | 2.207.800 | 476.100 |
| 15000 | 56.586.000 | 87.616.100 | 96.535.100 | 6.697.400 | 810.300 | 2.384.900 | 681.100 |
| 30000 | 218.579.600 | 362.886.700 | 379.839.000 | 12.458.100 | 1.533.400 | 3.870.300 | 1.394.600 |

3) Faça uma análise a respeito dos resultados obtidos no seu benchmark das tabela anteriores.

Através da análise do algoritmo podemos perceber que na primeira tabela, o método de ordenação com menor tempo de execução foi o Insertion Sort, enquanto o que demorou mais tempo foi o Bubble Sort. Na segunda tabela, o método que executou no menor tempo foi o Quick Sort, já o que demorou mais para executar foi o Bubble Sort. Por fim na tabela 3, o método que executou em menor tempo foi o Heap Sort, enquanto o que levou mais tempo de execução foi o Insertion Sort.

**Informações sobre cópias**

As questões são individuais. Em caso de cópias de trabalho a pontuação será zero para os autores originais e copiadores. Não serão aceitas justificativas como: “Fizemos o trabalho juntos, por isso estão idênticos”.