INSTITUTO FEDERAL DE MINAS GERAIS-IFMG

Disciplina: Algoritmos e Estrutura de Dados I

Prof. Geraldo Pereira de Souza

Prof. Roteiro Prático

Objetivo: praticar o uso de algoritmos de ordenação para tipos simples e objetos em Java

Parte 1 – Revisão dos conceitos básicos relacionados à ordenação e pesquisa

Etapa 1 (5 minutos): baixe os slides (do moodle) da aula teórica e reveja os conceitos relacionados a esse tópico.

Etapa 2 (20 minutos): Responda as questões abaixo:

a) O quê você entende por algoritmos de ordenação e por quê eles são importantes?
b) Quais requisitos são importantes na escolha de um algoritmo de ordenação?
c) O quê é um método de ordenação "**in situ**"?
d) O quê é um método de ordenação **estável**?
e) Qual a diferença entre um algoritmo de ordenação interna e externa?
f) Sendo n o número de elementos a serem ordenados o quê você entende por C(n) e M(n) conforme aula teórica?
g) Quando dizemos que um algoritmo tem comportamento conforme notação O(n), $O(n^2)$ ou O(log n) o que isso representa? Explique e desenhe as respectivas curvas.

Etapa 3 (10 minutos): Crie um projeto com nome TesteOrdenacao e faça o seguinte:

a) O programa deve declarar um vetor de 10 posições de inteiros;
b) Solicite que o usuário digite 10 números de modo aleatório e guarde em cada posição do vetor;
c) Dentro do programa invoque o método de ordenação bolha fornecido abaixo:

```
public static void bolha (int v[], int n){
  for (int i=n-1; i>=1; i--){
    for (int j=0; j<i; j++){
      if (v[j]>v[j+1]) { /* troca */
        int temp = v[j];
        v[j] = v[j+1];
        v[j+1] = temp;
      }
    }
  }
}
```

d) Imprima e conteúdo do vetor antes e depois de ordená-lo;
e) Mude o programa para considerar um vetor de tamanho 10, 50, 100, 500 e 1000. Para isso crie uma função chamada carregarVetor que gera vetores com números aleatórios de acordo com o tamanho requerido.

A função carregarVetor deve ter a seguinte assinatura:

public static void carregarVetor (int v[], int n){} // v é o vetor a ser carregado com números aleatórios e n é o número de elementos a serem considerados no vetor.

Etapa 4: Veja a implementação do método de ordenação quickSort. Primeiro crie um projeto com nome ProjetoQuickSort e incorpore o código fonte abaixo no seu projeto:

```
public static void swap(int a[], int i, int j)
{
     int T;
     T = a[i];
     a[i] = a[j];
     a[j] = T;

 }

public static void QuickSort(int a[], int indiceEsquerdo, int indiceDireito)
{
     int i = indiceEsquerdo;
     int j = indiceDireito;
     int mid;

     if ( indiceDireito > indiceEsquerdo)
     {

        /* Pega o pivô.
         */
        mid = a[ ( indiceEsquerdo + indiceDireito ) / 2 ];

        // repete até que as extremidades se cruzem
        while( i <= j )
        {
           /* procura o primeiro elemento que seja maior ou igual
            * ao pivê começando do lado esquerdo.
            */

            while( ( i < indiceDireito ) && ( a[i] < mid ))
               ++i;

           /* procura o primeiro elemento que seja menor ou igual
            * ao pivô começando do lado direito.
            */
            while( ( j > indiceEsquerdo ) && ( a[j] > mid ))
               --j;

           // se os índices não se cruzaram, efetua a troca
           if( i <= j )
           {
              swap(a, i, j);
              ++i;
              --j;
           }
        }

        /* Ordenar a parte indiceEsquerdo.
         */
        if( indiceEsquerdo < j )
           QuickSort( a, indiceEsquerdo, j );

        /* Se os índices não se cruzaram,
         * ordenar a parte indiceDireito.
```

```
        */
        if( i < indiceDireito )
            QuickSort( a, i, indiceDireito );
    }
}

 public static void sort(int a[], int n)
 {
     QuickSort(a, 0, n - 1);
 }

public static void main main(String a[])
{
   int v[] = {25,48,37,12,57,86,33,92};
   System.out.println("\nVetor desordenado: ");
   for (int i=0; i<8; i++){
          System.out.println(v[i] +  "  ");
          }
   sort(v, 8);
   System.our.println("\nVetor ordenado: ");
   for (int i=0; i<8; i++){
                   System.out.println(v[i] +  "  ");
   }
   System.out.println("\nFim da impressão: ");
}
```

Etapa 5: Compile e execute o código.

Etapa 6: Na função main do seu programa, simule a execução do programa com vetores de tamanho 100, 200, 500, 1000, 10.000, etc; Faça medições de tempo para as execuções para cada tamanho e para cada método de ordenação. Pesquise na internet sobre “Como medir tempo de execução de uma função em Java”.
Dica: Pesquise sobre o método ***System.currentTimeMillis();***

Parte II – Algoritmos de ordenação e Objetos

Exercício 1 (10 minutos): Incorpore o seguinte método de ordenação na classe Agencia do seu projeto Banco:

```
/* Ordenação bolha */
public static void bolha (Conta lista[], int n){

  for (int i=n-1; i>=1; i--){
    for (int j=0; j<i; j++){
      if (lista[j].getNumero() > lista[j+1].getNumero()) { /* troca */
        Conta temp = lista[j];
        lista[j] = lista[j+1];
        lista[j+1] = temp;
      }
    }
  }
}

public static void main(String a[])
{
  Conta lista[];

  lista[0] = new Conta (3, "Maria", 5);          // Construtor para número, titular e saldo.
  lista[1] = new Conta (2, "José", 10);
  lista[2] = new Conta (1, "Carlos", 8);
  lista[3] = new Conta (5, "João", 4);
  lista[4] = new Conta (4, "Raul", 7);

  bolha(lista, 5);

  for (int i=0; i<5; i++){
    lista[i].imprimeDadosConta();
  }

}
```

Etapa 6 (30 minutos): Implemente uma versão do método quickSort para ordenar objetos do tipo Conta na classe Agencia, mas sem ser um método estático. A função deve ter a seguinte assinatura:

```
public void quickSort (Conta listaContas[]);
```

Incorpore a função no seu projeto para testá-la.

Parte III – Collection em Java

Etapa 1) Revise os conceitos relativos a Collection em Java.

Etapa 2) Faça um programa que permita ao usuário digitar uma sequência de números inteiros positivos. Os números devem ser armazenados em uma variável do tipo "**Vector**". Posteriormente o programa deve imprimir os valores digitados.
Obs: Qualquer número negativo digitado indica o final da sequência.

Etapa 3) Inclua no programa feito na etapa 2 uma função que receba uma variável do tipo Vector de tamanho qualquer e imprima seu conteúdo na tela.

Etapa 4) Mude a função definida na etapa 4 para receber uma variável do tipo Vector que pode armazena valores de qualquer tipo T. Teste a função criada.

Etapa 5) Crie um projeto chamado ProjetoPilha e inclua no mesmo uma classe chamada Pilha que encapsula as funcionalidades de uma estrutura de dados do tipo “pilha”.

Etapa 6) Inclua na classe Pilha a função empilhar deve receber uma variável de um tipo qualquer e deve “empilhar” o valor da variável.

Etapa 7) A função desempilhar deve retornar o último elemento empilhado.

Etapa 8) A função imprimirConteudo da Pilha deve mostrar todos elementos já empilhados.

Etapa 9) A função esvaziar deve eliminar todos os elementos da Pilha.

Etapa 10) No main do seu projeto faça testes com a classe Pilha criada.