PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA

UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE

Bacharelado em Engenharia de Software

Ravi Antônio Gonçalves de Assis

4ª LISTA DE EXERCÍCIOS DE LABORATÓRIO DE COMPUTAÇÃO II

Belo Horizonte

2018

Lista Ordenação

1 – Gere um vetor de inteiros de tamanho 100 (aleatoriamente) e ordene-o em ordem crescente utilizando os seguintes métodos:

• Bubblesort clássico

```
public static int[] BBSortClassic(int[] vetO) {
	// TODO Auto-generated method stub
	int i, j, temp;
	int vet[] = vetO.clone();
	for ( i = 0; i < vet.length - 1; i++) {
		for (j = 0; j < vet.length - 1 ; j++) {
			if(vet[j] > vet[j+1]) {
				temp = vet[j];
				vet[j] = vet[j+1];
				vet[j+1] = temp;
			}
		}
	}
	return vet;
}
```

• Bubblesort Adap. 1

```
public static int[] BBSortAdap1(int[] vetO) {
	// TODO Auto-generated method stub
	int i, temp;
	int vet[] = vetO.clone();
	boolean trocou = true;
	while(trocou) {
		trocou = false;
		for (i = 0; i < vet.length -1; i++) {
			if(vet[i] > vet[i+1]) {
				temp = vet[i];
				vet[i] = vet[i+1];
				vet[i+1] = temp;
				trocou = true;
				}
		}
	}
	return vet;
}
```

• Bubblesort Adap. 2

```
public static int[] BBSortAdap2(int[] vet) {
	// TODO Auto-generated method stub
	int i, j, temp, cont;
	int vetor[] = vet.clone();
	cont = vetor.length - 1;
	for (i = 0; i < vetor.length - 1; i++) {
```

```
            for (j = 0; j < cont; j++)
                  if (vetor[j] > vetor[j + 1]) {
                        temp = vetor[j];
                        vetor[j] = vetor[j + 1];
                        vetor[j + 1] = temp;
                  }
            cont--;
      }
      return vetor;
}
```

• Bubblesort Adap. 3 (você deverá gerar uma versão híbrida das adaptações 1 e 2)

```
public static int[] BBSortAdap3(int[] vet) {
      // TODO Auto-generated method stub
      int i, temp, cont;
      boolean HouveTroca = true;
      int vetor[] = vet.clone();
      cont = vetor.length - 1;
      while (HouveTroca) {
            HouveTroca = false;
            for (i = 0; i < cont; i++)
                  if (vetor[i] > vetor[i + 1]) {
                        temp = vetor[i];
                        vetor[i] = vetor[i + 1];
                        vetor[i + 1] = temp;
                        HouveTroca = true;
                  }
            cont--;
      }
      return vetor;
}
```

• Selectionsort

```
public static int[] SelectionSort(int[] vet) {
      // TODO Auto-generated method stub
      int [] vetor = vet.clone();
      int i, j, min, aux;

      for (i = 0; i < vetor.length - 1; i++) {
            min = i;
            for (j = i + 1; j < vetor.length; j++)
                  if (vetor[j] < vetor[min] ) min = j ;

            aux = vetor[min];
            vetor[min] = vetor[i];
            vetor[i] = aux;

      }
```

```
        return vetor;
    }
```

• Insertionsort

```
    public static int[] insertionSort(int[] vet) {
        // TODO Auto-generated method stub
        int [] vetor = vet.clone();

        int i, j, v;
        for (i = 1; i < vetor.length; i++) {
            v = vetor[i];
            j = i;
            while ((j > 0) && (vetor[j - 1] > v)) {
                vetor[j] = vetor[j - 1];
                j--;
            }
            vetor[j] = v;
        }

        return vetor;
    }
```

2 – Gere um vetor de inteiros de tamanho 100 (aleatoriamente) e ordene-o em ordem decrescente utilizando os seguintes métodos:

• Bubblesort clássico

```
    public static int[] BBSortClassicDesc(int[] vetO) {
        // TODO Auto-generated method stub
        int i, j, temp;
        int vet[] = vetO.clone();
        for ( i = 0; i < vet.length - 1; i++) {
            for (j = 0; j < vet.length - 1 ; j++) {
                if(vet[j] < vet[j+1]) {
                    temp = vet[j];
                    vet[j] = vet[j+1];
                    vet[j+1] = temp;
                }
            }
        }
        return vet;
    }
```

• Bubblesort Adap. 1

```
    public static int[] BBSortAdap1Desc(int[] vetO) {
        // TODO Auto-generated method stub
        int i, temp;
        int vet[] = vetO.clone();
        boolean trocou = true;

        while(trocou) {
```

```
            trocou = false;
            for (i = 0; i < vet.length -1; i++) {
                  if(vet[i] < vet[i+1]) {
                        temp = vet[i];
                        vet[i] = vet[i+1];
                        vet[i+1] = temp;
                        trocou = true;
                  }
            }
      }
      return vet;
}
```

• Bubblesort Adap. 2

```
public static int[] BBSortAdap2Desc(int[] vet) {
      // TODO Auto-generated method stub
      int i, j, temp, cont;
      int vetor[] = vet.clone();
      cont = vetor.length - 1;
      for (i = 0; i < vetor.length - 1; i++) {
            for (j = 0; j < cont; j++)
                  if (vetor[j] < vetor[j + 1]) {
                        temp = vetor[j];
                        vetor[j] = vetor[j + 1];
                        vetor[j + 1] = temp;
                  }
            cont--;
      }
      return vetor;
}
```

• Bubblesort Adap. 3 (você deverá gerar uma versão híbrida das adaptações 1 e 2)

```
public static int[] BBSortAdap3Desc(int[] vet) {
      // TODO Auto-generated method stub
      int i, temp, cont;
      boolean HouveTroca = true;
      int vetor[] = vet.clone();
      cont = vetor.length - 1;
      while (HouveTroca) {
            HouveTroca = false;
            for (i = 0; i < cont; i++)
                  if (vetor[i] < vetor[i + 1]) {
                        temp = vetor[i];
                        vetor[i] = vetor[i + 1];
                        vetor[i + 1] = temp;
                        HouveTroca = true;
                  }
            cont--;
```

```
        }
        return vetor;
    }
```

• Selectionsort

```
    public static int[] SelectionSortDesc(int[] vet) {
        // TODO Auto-generated method stub
        int [] vetor = vet.clone();
        int i, j, min, aux;

        for (i = 0; i < vetor.length - 1; i++) {
            min = i;
            for (j = i + 1; j < vetor.length; j++)
                if (vetor[j] > vetor[min] ) min = j ;

            aux = vetor[min];
            vetor[min] = vetor[i];
            vetor[i] = aux;

        }
        return vetor;
    }
```

• Insertionsort

```
    public static int[] insertionSortDesc(int[] vet) {
        // TODO Auto-generated method stub
        int [] vetor = vet.clone();

        int i, j, v;
        for (i = 1; i < vetor.length; i++) {
            v = vetor[i];
            j = i;
            while ((j > 0) && (vetor[j - 1] < v)) {
                vetor[j] = vetor[j - 1];
                j--;
            }
            vetor[j] = v;
        }

        return vetor;
    }
```

3 – Crie um vetor com o nome de 10 pessoas (o vetor não precisa ser lido). Ordene-o em ordem crescente usando os mesmos algoritmos da questão 1.

```
public static String[] BBSortClassic(String[] vetO) {
    // TODO Auto-generated method stub
    int i, j;
    String temp;
    String vet[] = vetO.clone();
```

```
	for ( i = 0; i < vet.length - 1; i++) {
		for (j = 0; j < vet.length - 1 ; j++) {
			if(vet[j].compareToIgnoreCase(vet[j+1]) > 0){
				temp = vet[j];
				vet[j] = vet[j+1];
				vet[j+1] = temp;
			}
		}
	}
	return vet;
}

public static String[] BBSortAdap1(String[] vetO) {
	// TODO Auto-generated method stub
	int i;
	String temp;
	String vet[] = vetO.clone();
	boolean trocou = true;
	while(trocou) {
		trocou = false;
		for (i = 0; i < vet.length -1; i++) {
			if(vet[i].compareToIgnoreCase(vet[i+1]) > 0 ) {
				temp = vet[i];
				vet[i] = vet[i+1];
				vet[i+1] = temp;
				trocou = true;
			}
		}
	}
	return vet;
}
public static String[] BBSortAdap2(String[] vet) {
	// TODO Auto-generated method stub
	int i, j, cont;
	String temp;
	String vetor[] = vet.clone();
	cont = vetor.length - 1;
	for (i = 0; i < vetor.length - 1; i++) {
		for (j = 0; j < cont; j++)
			if(vetor[j].compareToIgnoreCase(vetor[j+1])>0){
				temp = vetor[j];
				vetor[j] = vetor[j + 1];
				vetor[j + 1] = temp;
			}
		cont--;
	}
	return vetor;
}
```

```
public static String[] BBSortAdap3(String[] vet) {
	// TODO Auto-generated method stub
	int i, cont;
	String temp;
	boolean HouveTroca = true;
	String vetor[] = vet.clone();
	cont = vetor.length - 1;
	while (HouveTroca) {
		HouveTroca = false;
		for (i = 0; i < cont; i++)
			if(vetor[i].compareToIgnoreCase(vetor[i+1])>0){
				temp = vetor[i];
				vetor[i] = vetor[i + 1];
				vetor[i + 1] = temp;
				HouveTroca = true;
			}
		cont--;
	}
	return vetor;
}

public static String[] SelectionSort(String[] vet) {
	// TODO Auto-generated method stub
	String [] vetor = vet.clone();
	int i, j, min;
	String aux;
	for (i = 0; i < vetor.length - 1; i++) {
		min = i;
		for (j = i + 1; j < vetor.length; j++)
			if(vetor[j].compareToIgnoreCase(vetor[min])<0)
				min = j ;
		aux = vetor[min];
		vetor[min] = vetor[i];
		vetor[i] = aux;
	}
	return vetor;
}

public static String[] insertionSort(String[] vet) {
	// TODO Auto-generated method stub
	String [] vetor = vet.clone();
	String v;
	int i, j;
	for (i = 1; i < vetor.length; i++) {
		v = vetor[i];
		j = i;
		while((j>0)&&(vetor[j-1].compareToIgnoreCase(v)>0)){
			vetor[j] = vetor[j - 1];
```

```
                j--;
            }
            vetor[j] = v;
        }

        return vetor;
    }
```