Trabalho Teórico 6

Unidade IV:
Ordenação Interna - Algoritmo de Inserção

Slide B.

1) Mostre todas as comparações entre elementos do array para os arrays abaixo.

1	2	3	4	5	6

Before Insertion Sort
Vector {1,2,3,4,5,6}

|Key|--------2--------|
1º) Compare -> 1<- ->2
Vector {1,2,3,4,5,6}

|Key|--------3--------|
2º) Compare -> 2<- ->3
Vector {1,2,3,4,5,6}

|Key|--------4--------|
3º) Compare -> 3<- ->4
Vector {1,2,3,4,5,6}

|Key|--------5--------|
4º) Compare -> 4<- ->5
Vector {1,2,3,4,5,6}

|Key|--------6--------|
5º) Compare -> 5<- ->6
Vector {1,2,3,4,5,6}

After Insertion Sort
Vector {1,2,3,4,5,6}

Observação: Ao se tratar de comparações temos:

Best Case f(x)= n-1

Worst Case f(x)= $\frac{n*(n-1)}{2}$

6	5	4	3	2	1

Before Insertion Sort
Vector {6,5,4,3,2,1}
|Key|--------5--------|
1º) Compare -> 6<- ->5
Vector {5,6,4,3,2,1}

|Key|--------4--------|
2º) Compare -> 6<- ->4
3º) Compare -> 5<- ->4
Vector {4,5,6,3,2,1}

|Key|--------3--------|
4º) Compare -> 6<- ->3
5º) Compare -> 5<- ->3
6º) Compare -> 4<- ->3
Vector {3,4,5,6,2,1}

|Key|--------2--------|
7º) Compare -> 6<- ->2
8º) Compare -> 5<- ->2
9º) Compare -> 4<- ->2
10º) Compare -> 3<- ->2
Vector {2,3,4,5,6,1}

|Key|--------1--------|
11º) Compare -> 6<- ->1
12º) Compare -> 5<- ->1
13º) Compare -> 4<- ->1
14º) Compare -> 3<- ->1
15º) Compare -> 2<- ->1
Vector {1,2,3,4,5,6}

After Insertion Sort
Vector {1,2,3,4,5,6}

2) Mostre todas as comparações e movimentações do algoritmo anterior para o array abaixo:

12	4	8	2	14	17	6	18	10	16	5	5	13	9	1	11	7	3

Before Insertion Sort
Vector
{12,4,8,2,14,17,6,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------4--------|
1º) Compare -> 12<- ->4
Vector
{4,12,8,2,14,17,6,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------8--------|
2º) Compare -> 12<- ->8
3º) Compare -> 4<- ->8
Vector
{4,8,12,2,14,17,6,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------2--------|
4º) Compare -> 12<- ->2
5º) Compare -> 8<- ->2
6º) Compare -> 4<- ->2
Vector
{2,4,8,12,14,17,6,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------14--------|
7º) Compare -> 12<- ->14
Vector
{2,4,8,12,14,17,6,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------17--------|
8º) Compare -> 14<- ->17
Vector
{2,4,8,12,14,17,6,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------6--------|
9º) Compare -> 17<- ->6
10º) Compare -> 14<- ->6
11º) Compare -> 12<- ->6
12º) Compare -> 8<- ->6
13º) Compare -> 4<- ->6
Vector
{2,4,6,8,12,14,17,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------18--------|
14º) Compare -> 17<- ->18
Vector
{2,4,6,8,12,14,17,18,10,16,15,5,13,9,1,11,7,3}

|Key|--------10--------|
15º) Compare -> 18<- ->10
16º) Compare -> 17<- ->10
17º) Compare -> 14<- ->10
18º) Compare -> 12<- ->10
19º) Compare -> 8<- ->10
Vector
{2,4,6,8,10,12,14,17,18,16,15,5,13,9,1,11,7,3}

|Key|--------16--------|
20º) Compare -> 18<- ->16
21º) Compare -> 17<- ->16
22º) Compare -> 14<- ->16
Vector
{2,4,6,8,10,12,14,16,17,18,15,5,13,9,1,11,7,3}

|Key|--------15--------|
23º) Compare -> 18<- ->15
24º) Compare -> 17<- ->15
25º) Compare -> 16<- ->15
26º) Compare -> 14<- ->15
Vector
{2,4,6,8,10,12,14,15,16,17,18,5,13,9,1,11,7,3}

|Key|--------5--------|
27º) Compare -> 18<- ->5
28º) Compare -> 17<- ->5
29º) Compare -> 16<- ->5
30º) Compare -> 15<- ->5
31º) Compare -> 14<- ->5
32º) Compare -> 12<- ->5
33º) Compare -> 10<- ->5
34º) Compare -> 8<- ->5
35º) Compare -> 6<- ->5
36º) Compare -> 4<- ->5
Vector
{2,4,5,6,8,10,12,14,15,16,17,18,13,9,1,11,7,3}
|Key|--------13--------|
37º) Compare -> 18<- ->13
38º) Compare -> 17<- ->13
39º) Compare -> 16<- ->13
40º) Compare -> 15<- ->13
41º) Compare -> 14<- ->13
42º) Compare -> 12<- ->13
Vector
{2,4,5,6,8,10,12,13,14,15,16,17,18,9,1,11,7,3}

|Key|--------9--------|
43º) Compare -> 18<- ->9
44º) Compare -> 17<- ->9
45º) Compare -> 16<- ->9
46º) Compare -> 15<- ->9
47º) Compare -> 14<- ->9
48º) Compare -> 13<- ->9
49º) Compare -> 12<- ->9
50º) Compare -> 10<- ->9
51º) Compare -> 8<- ->9
Vector
{2,4,5,6,8,9,10,12,13,14,15,16,17,18,1,11,7,3}

|Key|--------1--------|
52º) Compare -> 18<- ->1
53º) Compare -> 17<- ->1
54º) Compare -> 16<- ->1
55º) Compare -> 15<- ->1
56º) Compare -> 14<- ->1
57º) Compare -> 13<- ->1
58º) Compare -> 12<- ->1
59º) Compare -> 10<- ->1
60º) Compare -> 9<- ->1
61º) Compare -> 8<- ->1
62º) Compare -> 6<- ->1
63º) Compare -> 5<- ->1
64º) Compare -> 4<- ->1
65º) Compare -> 2<- ->1
Vector
{1,2,4,5,6,8,9,10,12,13,14,15,16,17,18,11,7,3}

|Key|--------11--------|
66º) Compare -> 18<- ->11
67º) Compare -> 17<- ->11
68º) Compare -> 16<- ->11
69º) Compare -> 15<- ->11
70º) Compare -> 14<- ->11
71º) Compare -> 13<- ->11
72º) Compare -> 12<- ->11
73º) Compare -> 10<- ->11
Vector
{1,2,4,5,6,8,9,10,11,12,13,14,15,16,17,18,7,3}

|Key|--------7--------|/
74º) Compare -> 18<- ->7/
75º) Compare -> 17<- ->7
76º) Compare -> 16<- ->7
77º) Compare -> 15<- ->7
78º) Compare -> 14<- ->7
79º) Compare -> 13<- ->7
80º) Compare -> 12<- ->7
81º) Compare -> 11<- ->7
82º) Compare -> 10<- ->7
83º) Compare -> 9<- ->7
84º) Compare -> 8<- ->7
85º) Compare -> 6<- ->7
Vector
{1,2,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,3}

|Key|--------3--------|
86º) Compare -> 18<- ->3
87º) Compare -> 17<- ->3
88º) Compare -> 16<- ->3
89º) Compare -> 15<- ->3
90º) Compare -> 14<- ->3
91º) Compare -> 13<- ->3
92º) Compare -> 12<- ->3
93º) Compare -> 11<- ->3
94º) Compare -> 10<- ->3
95º) Compare -> 9<- ->3
96º) Compare -> 8<- ->3
97º) Compare -> 7<- ->3
98º) Compare -> 6<- ->3
99º) Compare -> 5<- ->3
100º) Compare -> 4<- ->3
101º) Compare -> 2<- ->3
Vector
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18}

After Insertion Sort
Vector
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18}

4)Uma forma de melhorar o Algoritmo de Inserção é considerar a pesquisa binária para procurar a posição em que o novo elemento será inserido na lista ordenada. Nesse caso, realizamos O (lg m) comparações, onde m é o tamanho da lista ordenada, para encontrar a posição de inserção. Em seguida, em uma estrutura de repetição, movemos em uma unidade todos os elementos já ordenados e cuja posição é maior ou igual a de inserção. Implemente o Algoritmo de Inserção com Pesquisa Binária.

Resp: Arquivo.java → IsertionSort.java

```
class InsertionSort{
    public static void insertionSort(int array[]) {
        int n = array.length,
        aux=1;
        for (int j = 1; j < n; j++) {
            int key = array[j];
            int i = j-1;
            System.out.println("|Key|--------"+key+"--------|");
            System.out.println(aux +"º) Compare -> "+ array[i] + "<- -
>" + key);
            while ( (i >= 0) && ( array [i] > key ) ) {

                array [i+1] = array [i];
                i--;
                if(i>=0){
                    aux++;
                    System.out.println(aux +"º) Compare -
> "+ array[i] + "<- ->" + key);
                }
            }
            aux++;

            array[i+1] = key;
            printVector(array);
        }
    }
    public static void printVector(int array[]){
        System.out.print("Vector {");
            for(int l=0; l<array.length-1;l++){
                System.out.print(array[l]+",");
            }
            System.out.print(array[array.length-1]+"}");
            System.out.println(" ");
            System.out.println(" ");
    }

    public static int binarySearch(int array[],int numberSearch){
        int resp = -1,
            right = array.length - 1,
            left = 0,
```

```
            mid =-1;
        while (left <= right) {
                mid = (left + right) / 2;
                if (numberSearch == array[mid]){
                    resp = mid;
                    left = array.length;
                } else if (numberSearch > array[mid]){
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }

                if(left==right)
                    if(numberSearch<=array[left])
                        resp = left;
                    else
                        resp = right++;
         }
         return resp;
    }
    public static int[] addNumberSelection(int array[],int positionSelect
ion, int numberSelection){
        int[] array2 = new int[array.length+1];
        for(int i =0; i<positionSelection; i++)
            array2[i]=array[i];

        array2[positionSelection]=numberSelection;

        for(int i =positionSelection+1; i<array2.length; i++)
            array2[i]=array[i-1];

        return array2;
    }
    public static void main(String[] args) {
        int number= 0;
        int[] array = {0,2,4,6,10,8};
        System.out.println("Before Insertion Sort");
        printVector(array);
        insertionSort(array);
        System.out.println("After Insertion Sort");
        printVector(array);
        System.out.println("Enter a number");
        number=MyIO.readInt();
        array = addNumberSelection(array, binarySearch(array,number), num
ber);
        System.out.println("After Binary Insert");
        printVector(array);
    }
}
```

4) Quando os elementos estão ordenados de forma decrescente tanto o Seleção como o Inserção realizam comparações nesse caso, qual dos dois algoritmos executará mais rápido? Justifique sua resposta.

Comparações →
Vamos analisar o número de Comparações que os algoritmos de Inserção e de seleção realizam no pior caso:
Seleção → $\frac{n^2}{2} - \frac{n}{2}$
Inserção→ $\frac{n*(n-1)}{2}$
Sabemos que ao simplificar a equação referente ao número comparações no algoritmo de inserção obtemos a equação do algoritmo de Seleção.

Movimentações→
Vamos analisar o número de Comparações que os algoritmos de Inserção e de seleção realizam no pior caso:
Seleção → $n * 3$
Inserção→ $\frac{n*(n-1)}{2} + 1$
Agora vamos plotar os resultados em uma tabela

	Inserção	Seleção
Tamanho do Vetor	Num de Movimentos	Num de Movimentos
2	2	3
3	4	6
4	7	9
5	11	12
6	16	15
7	22	19

Percebemos que neste caso o número de Movimentações nos algoritmos de inserção compensa até um vetor de tamanho 5, quando trabalharmos com vetores de tamanho maiores que 5 o algoritmo de Seleção é o mais indicado.

Tempo de execução→

Sabemos que o tempo de execução varia de acordo com número de movimentações e comparações. Neste caso o número de comparações e a mesma em ambos, porem o número de Movimentações é diferente. A partir destes dados concluímos que o tempo de execução será menor nos algoritmos de inserção para um vetor de ate tamanho 5 e menor nos algoritmos de seleção para um vetor de tamanho maior que 5.

Exercise Resolves.

1) Uma dúvida básica sobre o operador AND pode prejudicar a compreensão do nosso algoritmo. Assim, o que será escrito na tela pelo programa abaixo?

```
class ExercicioDuvidaAND {
    public static boolean m1() {
        System.out.println("m1");
        return false;
    }
    public static boolean m2() {
        System.out.println("m2");
        return true;
    }
    public static void main(String[] args) {
        System.out.println("inicio");
        boolean and = m1() && m2();
        System.out.println("fim: " + and);
    }
}
```

Inicio
m1
fim: false