



Programação 2

2º Semestre

Folha de Exercícios 3

Algoritmos de Ordenação e Complexidade

Efectue as tarefas de programação descritas abaixo, usando a linguagem C++ em ambiente Linux.

1. Escreva um programa para ordenar uma lista de nomes introduzidos pelo teclado (um nome em cada linha). Fazer a mesma coisa eliminando repetidos.

(Sugestão: usar as funções **InsertionSort**, **InsertSorted** e **BinarySearch** apresentadas nas aulas teóricas).

2. Partindo dos dois vectores de inteiros **a** e **b** deverá identificar quais os valores repetidos que existem no vector **a** e trocá-los por valores que existam no vector **b** (tendo em atenção que não pode trocar valores já existentes por outros já existentes). No final deverá ordenar o vector **a** utilizando o algoritmo MergeSort.

Vector **A** – 45, 32, 3, 4, 6, 8, 32, 34, 23, 6, 4

Vector **B** – 35, 2, 23, 33, 60, 45, 2, 3, 70, 10, 9

3. Analise e calcule a complexidade dos seguintes pedaços de código.

a)

```
void imprime_matriz(int largura, int altura, int ntabs)
{
    num = 1;
    for(int a = 1 ; a <= altura ; a++)
    {
        cout << "[";
        for(int l = 1 ; l <= largura ; l++)
        {
            cout << num++;
            for(int t = 1 ; t <= ntabs ; t++)
                cout << "\t";
        }
        cout << "]" << endl;
    }
    cout << endl;
}
```

b)

```
void sort(int vec[], int size )
{
    for( int i = size - 1 ; i > 0 ; i--)
    {
        bool troca = false;
        for( int j = 0 ; j < i; j++)
        {
            if ( vec[j] > vec[j+1] )
            {
                int tmp = vec[j+1];
```

```

        vec[j+1] = vec[j];
        vec[j] = tmp);
        troca = true;
    }
}
if (!troca)
    return;
}
}

```

c)

```

int pesquisa (int v[], int size, int x)
{
    left = 0;
    right = size-1;
    while (left <= right)
    {
        int middle = (left + right) / 2;
        if (x == v[middle])
            return middle; // encontrou
        else if (x > v[middle])
            left = middle + 1;
        else
            right = middle - 1;
    }
    return -1;
}

```

4. Outro método de ordenação muito conhecido é o método de ordenação por selecção. Neste método, selecciona-se e passa-se para a última posição o maior elemento do array, e procede-se da mesma forma com os restantes elementos do array, até só restar 1 elemento. Uma rotina em C++ é apresentada de seguida.

```

template <class T>
void SelectionSort(T v[], int n)
{
    /*1*/   for( ; n > 1; n--)
    /*2*/   {
        // determina a posição (pos_max) do maior elemento do array
    /*3*/       int pos_max = 0;
    /*4*/       for (int i = 1; i < n; i++)
    /*5*/           if (v[i] > v[pos_max])
    /*6*/               pos_max = i;
        // troca v[pos_max] com v[n-1]
    /*7*/       T val_max = v[pos_max];
    /*8*/       v[pos_max] = v[n - 1] ;
    /*9*/       v[n - 1] = val_max;
    }
}

```

a) Determine o número de vezes que cada instrução da função **SelectionSort** é executada (em função de **n**), no pior caso, no caso médio e no melhor caso. No caso dos ciclos **for**, distinga as três partes do cabeçalho do ciclo. As instruções foram numeradas para mais fácil referência.

b) Determine a complexidade temporal $T(n)$ e a complexidade espacial $S(n)$ da função **SelectionSort**, no pior caso, no caso médio e no melhor caso, usando a notação de O grande. Suponha que $n > 0$.

c) Compare este método de ordenação com os métodos estudados nas aulas teóricas (ordenação por inserção e ordenação por partição), em termos da sua eficiência temporal.

5. Imagine que tem a chave do último euromilhões não ordenada e que os últimos dois números correspondem ao número de estrelas. Ordene o vector utilizando os algoritmos indicados tendo em atenção que deverá aparecer primeiramente o número da chave e só depois o número das estrelas. Exemplo: o vector com os elementos:

43 | 3 | 27 | 33 | 15 | 5 | 2

deverá tornar-se em

3 | 15 | 27 | 33 | 43 | 2 | 5

a) ShellSort

b) MergeSort