Ficha 5

Programação Imperativa

Ordenação de vectores

1. Defina uma função `void insere (int v[], int N, int x)` que insere um elemento (`x`) num vector ordenado. Assuma que as `N` primeiras posições do vector estão ordenadas e que por isso, após a inserção o vector terá as primeiras `N+1` posições ordenadas.

2. A função ao lado usa a função insere para ordenar um vector.
   Apresente uma definição alternativa deste algoritmo sem usar a função `insert`.

```
void iSort (int v[], int N) {
  int i;
  for (i=1; (i<N); i++)
    insere (v, i, v[i]);
}
```

3. Defina uma função `int maxInd (int v[], int N)` que, dado um array com `N` inteiros, calcula o índice onde está o maior desses inteiros.

4. Use a função anterior na definição de uma função de ordenação de arrays de inteiros, que vai repetidamente calculando os maiores elementos e trocando-os com o elemento que está na última posição.

5. Apresente uma definição alternativa do algoritmo da alínea anterior sem usar a função `maxInd`.

6. Considere a definição ao lado da função `bubble`.
   Ilustre a execução da função com um pequeno exemplo. Verifique que após terminar, o maior elemento do vector se encontra na última posição.

```
void bubble (int v[], int N) {
  int i;
  for (i=1;(i<n); i++)
    if (v[i-1] > v[i])
       swap (v,i-1, i);
}
```

7. Use a função `bubble` na definição de uma função `void bubbleSort (int v[], int N)` que ordena o array `v` por sucessivas invocações da função `bubble`.

8. Uma optimização frequente da função `bubbleSort` consiste em detectar se o array já está ordenado. Para isso basta que uma das passagens pelo array não efectue nenhuma troca. Nesse caso podemos concluír que o array já está ordenado.

   Incorpore essa optimização na função anterior.