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.

9. Defina uma função `void merge (int r [], int a[], int na, int b[], int nb)` que, dados vectores ordenados `a` (com `na` elementos) e `b` (com `nb` elementos), preenche o array `r` (com `na+nb` elementos) com os elementos de `a` e `b` ordenados.
   Usando essa função podemos definir a função `mergesort` ao lado para ordenar um vector. Note que, se `v` é um vector, `v+m` é o vector (sufixo de `v`) que começa na posição `m`, i.e.,

   `v + m` = `&(v[m])`

```
void mergesort (int v[], int n,
                int aux[]) {
  int i, m;

  if (n>1) {
    m = n/2;
    mergesort (v, m, aux);
    mergesort (v+m, n-m, aux);
    merge (aux, v, m, v+m, n-m);
    for (i=0; (i<n); i++)
        v[i] = aux[i];
  }
}
```

10. O algoritmo de quick-sort para ordenação de um vector pode ser descrito da sequinte forma:

    (a) Começa-se por escolher um elemento do array (chamado pivot)

    (b) De seguida particiona-se o vector em duas partes:

    - os elementos nas posições `0..p-1` são todos menores do que o pivot e
    - os elementos nas posições `p+1..n-1` são maiores ou iguais ao pivot (que se encontra na posição `p`).

    (c) Aplicam-se os dois passos anteriores a ambas as partes do vector identificadas acima.

    Assumindo que existe uma função `int partition (int v[], int a, int b)` que realiza a partição do vector referida atrás (o passo (b)), a definição ao lado traduz este processo de ordenação.

    Apresente uma possível definição da função `partition`.

```
void qsort (int v[], int n){
  qsortAux (v,0,n-1);
}

void qsortAux (int v[], int a, int b) {
  int p;
  if (a<b) {
    p = partition (v, a, b);
    qsortAux (v, a, p-1);
    qsortAux (v, p+1, b);
  }
}
```