

# Algoritmos e Complexidade

## LEI/LCC (2º ano)

### 5ª Ficha Prática

O objectivo desta ficha é a análise do tempo de execução de algoritmos simples implementados com ciclos, sem a utilização de recursividade nem de estruturas de dados complexas.

1. Considere-se o problema de *pesquisa* numa sequência de números, e uma função (escrita em C) que resolve o problema.

Dada uma sequência de números inteiros  $v$  de dimensão  $n$  e um inteiro  $x$ , pretende-se obter como resultado o índice da primeira ocorrência de  $x$  em  $v$ , ou o valor -1 caso  $x$  não ocorra em  $v$ .

```
int search(int v[], int n, int x) {
    int i = 0, found = 0;
    while (i < n && !found) {
        if (v[i] == x) found = 1;
        i++;
    }
    if (!found) return -1;
    else return i-1;
}
```

- (a) Analise o tempo de execução da função no melhor e no pior caso.
  - (b) Se a sequência  $v$  estiver ordenada é possível melhorar o comportamento do algoritmo. Basta considerar a posição no meio da sequência e comparar o seu conteúdo com o elemento  $x$ , o que permite eliminar a inspecção de metade da sequência. O algoritmo de *pesquisa binária* repete este procedimento até encontrar  $x$ , ou a sub-sequência considerada ser vazia. Argumente que este algoritmo executa em tempo logarítmico no pior caso.
2. Considere o algoritmo de ordenação Bubble Sort:

```
void bubble_sort(int A[], int N) {
    for (i=1 ; i<N ; i++)
        for(j=N ; j>i ; j--)
            if (A[j] < A[j-1])
                swap(A,j,j-1);
}
```

```
void swap(int A[], int i, int j) {
    int aux=A[i];
    A[i]=A[j];
    A[j]=aux;
}
```

- (a) Caracterize o funcionamento deste algoritmo *no melhor e no pior caso* utilizando a notação  $\Theta$ .
- (b) Caracterize o funcionamento *global* deste algoritmo utilizando as notações  $\mathcal{O}$  e  $\Omega$ . Que relação existe entre a resposta a esta alínea e a resposta à alínea anterior?
- (c) Altere o ciclo exterior do algoritmo por forma a terminar quando detectar que a sequência já está ordenada.
- (d) Repita a análise do tempo de execução para o algoritmo optimizado.
- (e) Analise a seguinte versão alternativa do algoritmo:

```
void bubble_sort(int A[], int N) {
    for (i=1 ; i<N ; i++)
        for(j=N ; j>i ; j--)
            if (A[j] < A[i])
                swap(A,j,i);
}
```

3. Considere a seguinte descrição informal de um algoritmo de ordenação a que chamaremos *max sort*:

*A sequência a ordenar está em cada passo dividida em duas sub-sequências, uma não-ordenada seguida de uma ordenada (inicialmente a parte ordenada é vazia). Em cada passo o algoritmo selecciona o maior elemento na parte não-ordenada e troca-o com o último elemento dessa mesma parte. Neste momento esse elemento passa a fazer parte da sub-sequência ordenada.*

Exemplo de execução (o caracter | indica a fronteira entre as sub-sequências):

$$[3, 4, 1, 2 \mid] \longrightarrow [3, 2, 1 \mid 4] \longrightarrow [1, 2 \mid 3, 4] \longrightarrow [1 \mid 2, 3, 4] \longrightarrow [\mid 1, 2, 3, 4]$$

- (a) Efectue a análise assintótica do comportamento no pior caso de uma implementação *baseada em ciclos* (i.e. sem recursividade) deste algoritmo.
  - (b) Como poderá o algoritmo ser optimizado de forma a parar quando a sequência se encontra ordenada, analogamente ao que foi feito na questão anterior para o *bubble sort*?
4. Considere o seguinte algoritmo para o problema da avaliação do valor de um polinómio

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

num ponto  $x$  dado, sendo o polinómio representado por um vector de coeficientes:

```
float Poly (float a[], int n, float x)
{
    float p, xpotencia;
    int i;
    p = a[0] + a[1] * x;
    xpotencia = x;
    for (i=2 ; i<=n ; i++) {
        xpotencia = xpotencia * x;
        p = p + a[i] * xpotencia;
    }
    return p;
}
```

- (a) Quantas operações de soma e multiplicação efectua este algoritmo? Caracterize o comportamento assintótico do seu tempo de execução.
- (b) O *algoritmo de Horner* é uma alternativa mais eficiente para a resolução do mesmo problema. Trata-se de uma optimização do algoritmo anterior, que efectua uma *factorização* do polinómio [note-se que  $ab + ac$  pode ser calculado com apenas uma multiplicação como  $a(b + c)$ ]:

```
float HornerPoly (float a[], int n, float x)
{
    float p;
    int i;
    p = a[n];
    for (i=n-1 ; i>=0 ; i--)
        p = p*x + a[i];
    return p;
}
```

Quantas operações de soma e multiplicação efectua este algoritmo? Caracterize o comportamento assintótico no pior caso do seu tempo de execução.