

# Algoritmos e Complexidade

Época Especial, MIEI, LCC 2º ano

27 de Julho de 2019 – Duração: 2:30 h

1. Considere o problema de, dados dois arrays de inteiros **a** e **b** com **N** elementos, determinar o comprimento do maior prefixo comum.

(a) Apresente uma especificação (pré e pós- condições) da função.

(b) Apresente um invariante apropriado à prova da correcção parcial da função bem como a condição de verificação correspondente à prova da preservação do invariante.

```
int maiorPrefixo (int a[], int b [], int N) {  
    int i;  
    // PRE: ...  
    i = 0;  
    while (i<N && a[i] == b[i])  
        // Inv: ...  
        i++;  
    // POS: ...  
    return i;  
}
```

2. Relembre a função de partição de um array tal como necessário no algoritmo quick-sort de ordenação de um array.

De forma a analisar o comportamento desta função relativamente ao número de trocas (**swap**):

- Identifique o melhor e pior casos da execução desta função. Para cada um dos casos indique quantas trocas são feitas.
- Admitindo uma distribuição aleatória calcule o número médio de trocas que são efectuadas.

```
int partition (int v[], int N){  
    int i, j;  
    for (i=j=0; j<N-1; j++)  
        if (v[j] < v[N-1])  
            swap (v,i++,j);  
    swap (v,i,N-1);  
    return i;  
}
```

3. Considere o problema de, dada uma sequência (*array*) de números inteiros, e um número inteiro denominado **alvo**, determinar quantas sub-sequências (segmentos do array) têm uma soma exactamente igual ao alvo.

Por exemplo, para o array **v** = [1,2,3,4,1,3,2,5,8,1] e um alvo 10, existem 3 destas sub-sequências ([1,2,3,4], [4,1,3,2] e [3,2,5]).

```
int sum (int v[], int N){  
    int soma = 0, i;  
    for (i=0; i<N; i++)  
        soma += v[i];  
    return soma;  
}
```

```
int countsums (int v[], int N, int target){  
    int count = 0, u, l;  
    for (u=0; u<N; u++)  
        for (l=u; l<N; l++)  
            if (sum (v+u, l-u+1) == target)  
                count++;  
    return count;  
}
```

- (a) Admitindo que a invocação de **sum** (**v**,**N**) consulta **N** elementos do array **v**, determine o número de consultas que são feitas na invocação de **countsums** (**v**, **N**, **t**). Justifique a sua resposta.
- (b) Na execução desta função (**countsums**), a maioria das invocações da função **sum** repetem os cálculos feitos na invocação anterior. Por exemplo, quando **u==0** e **l==10** a função **sum** é usada para calcular a soma dos elementos de 0 até 10. Na invocação anterior (para **l==9**) tinha calculado a soma dos elementos de 0 até 9!

Usando esta observação apresente uma definição alternativa com uma complexidade inferior (justifique essa melhoria).

4. Relembre a implementação de *min-heaps* num *array*. Defina uma função `int quantosMenores (int h[], int N, int x)` que, dada uma min-heap de inteiros `h` com `N` elementos e um inteiro `x`, calcula quantos dos elementos dessa min-heap são menores do que o inteiro `x`. Certifique-se que a sua função toma partido da ordem pela qual os elementos estão armazenados no array.
5. Assuma que existe disponível uma função `int diskstraSP (Grafo g, int o, int d, int pais[], int pesos)` que calcula o caminho mais curto num grafo `g` desde o vértice `o` até ao vértice `d`. Defina uma função `int maiorEtapa (Grafo g, int o, int d)` que calcula o peso da aresta mais pesada do caminho mais curto entre `o` e `d`. A função deverá retornar `-1` se não existir qualquer caminho.
6. Considere que, usando uma lista ligada de inteiros, se implementa um tipo de dados com duas operações:
  - **insere** que acrescenta um número à lista (em tempo constante).
  - **soma** que transforma a lista numa lista singular com o somatório dos elementos da lista. Esta operação tem uma complexidade linear no comprimento da lista.

Mostre, usando o método do potencial, que ambas as operações têm um custo amortizado constante.