MCTA003-17 - Análise de Algoritmos
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Prof. Maycon Sambinelli

Análise de Tempo de Execução: Algoritmos Iterativos

1. Para cada um dos Algoritmos a seguir, forneça duas expressões, em notação assintótica, de descrição de tempo de execução, sendo uma para o melhor caso e outra para o pior caso. Nos dois primeiros algoritmos, ambos os valores n e m descrevem o tamanho da entrada, por isso suas expressões devem ser uma função sobre ambos. Use a notação Θ sempre que possível. Justifique sua resposta.

Algorithm 1 BubbleSort

```
1: Função BubbleSort(A, n)
2:     Para i = n até 2, decrementando faça
3:         Para j = 1 até i − 1, incrementando faça
4:             Se A[j] > A[j + 1] então
5:                 troque A[j] com A[j + 1]
```

Algorithm 2 Busca em dois vetores.

```
1: Função BuscaVetores(A, n, B, m, k)
2:     Para i = 1 até n faça
3:         Se A[i] == k então
4:             Devolve verdadeiro
5:     Para i = 1 até m faça
6:         Se B[i] == k então
7:             Devolve verdadeiro
8:     Devolve falso
```

Algorithm 3 Abacate

```
1: Função ABACATE(A, B, n)
2:     k ← 0
3:     i ← 1
4:     Enquanto i ≤ n faça
5:         k ← k + 1
6:         j ← i + 1
7:         Enquanto j ≤ n e A[j] < B[i] faça
8:             j ← j + 1
9:         i ← j
10:    Devolve k
```

Algorithm 4 Busca por elemento em comum.

```
1: Função BUSCAINTERSECAO(A, n, B, m)
2:     Para i = 1 até n faça
3:         Para j = 1 até m faça
4:             Se A[i] == B[j] então
5:                 Devolve verdadeiro
6:     Devolve falso
```

Algorithm 5 Busca por elementos duplicados no mesmo vetor.

```
1: Função BUSCADUPLICADOS(A, n)
2:     Para i = 1 até n faça
3:         Para j = i + 1 até n faça
4:             Se A[i] == A[j] então
5:                 Devolve verdadeiro
6:     Devolve falso
```

2. Forneça um limitante superior justo, usando notação assintótica, para os seguintes algoritmos

(a)
```
int isPrime (int n) {
    for (int i = 2; i * i <= n; i++)
        if (0 == n % i)
            return 0;
    return 1;
}
```

(b)
```
double exp (double a, int n) {
        if (n == 0) return 1.0;
        if (n == 1) return a;
        if (0 == n % 2)
            return exp(a*a, n/2);
        else
            return a * exp(a, n-1);
}
```

(c)
```
int PowersOfTwo(int n) {
        int i;
        i = 0;
        while (n%2 == 0) {
            n = n/2;
            i++;
        }
        return i;
}
```

(d)
```
for (i = 0; i < n-1; i++) {
        small = i;
```

```
        for (j = i+1; j < n; j++)
            if (A[j] < A[small])
                small = j;
        temp = A[small];
        A[small] = A[i];
        A[i] = temp;
    }
```

(e)
```
int gcd(int a, int b) {
        while (a !=  b) {
            if (a >= b)
                a = a - b;
            else
                b = b - a;
        }
        return a;
    }
```

(f)
```
int gcd(int a, int b) {
        while (b !=  0) {
            int t = b;
            b = a % b;
            a = t;
        }
        return a;
    }
```

3. Nesse exercício, um número inteiro x com n dígitos será representado em um vetor com n posições, uma para cada dígito, de forma invertida. Assim, por exemplo, o número 38549 é representado pelo vetor $A = (9, 4, 5, 8, 3)$.

Considere o problema de adicionar dois inteiros de n dígitos cada que estão representados em dois vetores A e B, de tamanho, portanto, n. A soma dos dois inteiros deve ser representada em um vetor C de tamanho $n + 1$.

Escreva um algoritmo que resolva esse problema e escreva uma expressão para seu tempo de execução no pior e no melhor caso, usando notação assintótica.

4. Seja $A[1..n]$ um vetor ordenado com elementos distintos. Mostre um algoritmo que decide se existe índice i, com $1 \leq i \leq n$, tal que $A[i] = i$ em tempo $O(\log n)$.

5. Sejam $X[1..n]$ e $Y[1..n]$ dois vetores, cada um contendo n números ordenados. Dê um algoritmo $O(\log n)$ para encontrar a mediana de todos os $2n$ elementos dos vetores X e Y.

6. Escreva um algoritmo que rearranje um vetor $A[ini..fim]$ de inteiros de modo que tenhamos $A[ini..j-1] \leq 0$ e $A[j..fim] > 0$ para algum j em $ini..fim+1$. Faz sentido exigir que j esteja em $ini..fim$? Procure fazer um algoritmo rápido que não use vetor auxiliar.

7. Neste problema, considere que fotos e imagens são representadas por matrizes binárias (matrizes que contêm apenas 0's e 1's). Dada a imagem de um elemento X que está sendo

procurado, que é uma matriz binária de tamanho $p \times q$, deseja-se procurá-la em uma foto F de tamanho $m \times n$. Cada posição de uma matriz é também chamada de pixel.

(a) Escreva em pseudocódigo uma função de nome PROCURA-SE, especificada de forma a receber parâmetros X, p, q, F, m e n como descritos acima. A função deve devolver `Verdadeiro` se existir uma submatriz de F delimitada por p linhas consecutivas e q colunas consecutivas que seja idêntica à matriz X, e `Falso` caso contrário.

(b) Em função dos parâmetros fornecidos à função PROCURA-SE, calcule o número de comparações de pixels feitas, no pior caso.

(c) Suponha que haja $N = 10$ milhões de fotos em um certo banco de dados. Adote os valores $p = q = 100$, $m = n = 1000$, e suponha que cada comparação de pixels (elementos das matrizes) gasta ao menos 10^{-13} dias (8.64 nanosegundos). Calcule, em dias, o tempo que um programa que chame a sua função PROCURA-SE N vezes levará, no pior caso, para executar todas as comparações de pixels necessárias na procura de uma imagem X nas N fotos desse banco de dados.

8. Dado um vetor A com n elementos ordenados de forma crescente e um valor k, escreva um algoritmo que encontre a quantidade de ocorrências de k em A em tempo $O(\log n)$. Por exemplo, se $A = (50, 50, 68, 68, 68, 68, 76, 79, 90)$ e $k = 68$, o algoritmo deve devolver 4. Você pode considerar que k ocorre em A ao menos uma vez.