Lista 3 - Projeto e Análise de Algoritmos

Matrícula: 201700030154

Aluno: Edilson leite Santos Junior

Turma: 02

```
1)
    T(n) = 2T(n/2) + n, T(1) = 0

    Supondo n=2^k  ⇒ T(2^k) = 2T(2^k-1) + 2^k, T(2^0) = 0

    T(2^k-1) = 2T(2^k-2) + 2^(k-1)
    T(2^k-2) = 2T(2^k-3) = 2^(k-2)

    T(2^k) = 2T(2^(k-1)) + 2^k =

            = 2 * [ 2T(2^(k-2)) + 2^(k-1) ] + 2^k =

            = 2 * { 2 * [2T(2^k-3) + 2^(k-2)] + 2^(k-1)} + 2^k =

            = 2^3 * T(2^(k-3)) + 3 * 2^k =
             ...
            = 2^k * T(2^0) + k * 2^k = k * 2^k

            2^k = n ⇒ k = log(n)

            T(n) = n*log(n) = O(n*log(n))
```

```
2) Obs: são considerados grandes valores de n;

    Solução s: T(n) = 7T(n/3) + n^2 T(1) = 1

        Teorema mestre: a = 7; b = 3; f(n) = n^2

        log b(a) = log 3(7) ~= 1.77

        f(n) = n^2 > n^(1.77124374916) -- caso 3 do Teorema

        ϵ = 0.23

        f(n)= Ω(n^(1.77 + 0.23)) = Ω(n^2)

        analisando a condição de regularidade:

            a * f(n/b) ⩽ c * f(n)
            7*(n^2/9) ⩽ c * n^2
            (7/9) * n^2 ⩽ c * n^2
            7/9 ⩽ c < 1  -- condição satisfeita, então T(n) = O(f(n)) = O(n^2)

    Solução t: T(n) = T(n/2) + sqrt(n)

        Teorema mestre: a = 1; b = 2; f(n) = sqrt(n)

        log2(1) = 0

        f(n) = sqrt(n) > n^0 -- caso 3 do Teorema

        ϵ = 1/2

        f(n)= Ω(n^(0 + 1/2)) = Ω(sqrt(n))

        analisando a condição de regularidade:

            a * f(n/b) ⩽ c * f(n)
            sqrt(n/2) ⩽ c * sqrt(n)
            sqrt(1/2) ⩽ c
            1/1.4 ⩽ c  -- condição satisteita, então T(n) = O(f(n)) = O(sqrt(n))

    A solução t é mais eficiente
```

```
3)
    a)

        Caso base: n = 1; z é comparado com o único elemento do vetor. Se existe, é retornado
            1 caso exista ou 0 caso exista;
        Hipótese de indução: para qualquer vetor de tamanho i, 1 ⩽ i < k, é possível verificar
            se o elemento z existe ou não dentro desse vetor por meio de busca ternária;
        Caso geral: Supondo a HI, desejamos provar que se a propriedade é válida para 1 ⩽ i < k,
            então é válida para k;
```

```
b)

    função BuscaTernaria(X, primeiro, último, z): inteiro

    {-
        primeiro indica a primeira posição da seção do array a ser analisada. Valor inicial: 1
        último indica a última posição da seção do array a ser analisada. Valor inicial: n
    -}

        início
            se (último = primeiro)

            então se (X[último] = z)

                então retorne último

                senão retorne 0

            senão se (X[ultimo - primeiro)/3 + primeiro = z)

                então retorne (ultimo - primeiro)/3 + primeiro)

                senão se (X[ultimo - primeiro)/3 + primeiro] > z)

                    então retorne BuscaTernaria(X, primeiro, (ultimo - primeiro)/3 + primeiro, z)

                    senão se (X[2*(ultimo - primeiro)/3 + primeiro] = z)

                        então retorne 2*(ultimo - primeiro)/3 + primeiro

                        senão se (X[2*(ultimo - primeiro)/3 + primeiro] > z)

                            então retorne BuscaTernaria(X, (ultimo - primeiro)/3 + primeiro, 2*(ultimo - primeiro)/3 + primeiro, z)

                            senão retorne BuscaTernaria(X, 2*(ultimo - primeiro)/3 + primeiro, último, z)
        fim
```

```
c) Algoritmo recursivo: T(n) = T(n/3) + c*(n^0)

d) Teorema Mestre: a = 1; b = 3; f(n) = n^0

    logb(a) = log3(1) = 0

    f(n) = n^0 = n^(logb(a)) -- caso 2 do Teorema

    T(n) = O(n^(log b(a)) * log (n))
    T(n) = O(log (n)) = O(log (n))

e) A eficiência do algoritmo de busca ternária é a mesma da de busca binária: O(log(n))
```