Francisco Braz

1-)

$T(n) = 2T(n/2) + n,\ T(1) = 0$

Supondo $n = 2^K \rightarrow T(2^K) = 2T(2^{K-1}) + 2^K,\ T(1) = 0$

$T(2^K) = 2T(2^{K-1}) + 2^K$

$= 2 * [2T(2^{K-2}) + 2^{K-1}] + 2^K$

$= 2^2T(2^{K-2}) + 2^1 * 2^{K-1} + 2^K = 2^2T(2^{K-2}) + 2^K + 2^K$

$= 2^2 * [2T(2^{K-3}) + 2^{K-2}] + 2^K + 2^K$

$= 2^3T(2^{K-3}) + 2^1 * 2^{K-2} + 2^K + 2^K = 2^3T(2^{K-3}) + 2^{K-1} + 2^K + 2^K$

$= 2^3 * [T(2^{K-4}) + 2^{K-3}] + 2^{K-1} + 2^K + 2^K$

$= 2^4T(2^{K-4}) + 2^{K-2} + 2^{K-1} + 2^K + 2^K$

…

$= 2^KT(2^0) + 2^2 + 2^3 + \ldots + 2^{K-1} + 2^K + 2^K$, como $T(1) = 0$:

$2^2 + 2^3 + 2^4 + \ldots + 2^{K-1} + 2^K + 2^K$

Podemos perceber que a partir do segundo termo até o primeiro 2^K, um termo é sempre o anterior multiplicado por 2, logo estamos diante de uma PG com razão 2. Então:

$\frac{4(2^K - 1)}{2-1} + 2^K = 4(2^K - 1) + 2^K$, desfazendo a troca de variável ($n = 2^K$):

$4(n - 1) + n = 4n - 4 + n = 5n - 4$

Como na notação Big O, as constantes são relativamente insignificantes para valores grandes de n, poderíamos ignorar as constantes, ficando somente com o termo n. Assim, nossa complexidade seria: O(n)

2-)

Solução S: T(n) = 7T($\frac{n}{3}$) + n^2

Pelo teorema mestre:

a = 7, b = 3, f(n) = n^2

OBS: (n^$\log_3$7 - ε) = $n^{log_b a-\varepsilon}$

n^1,8-ε = $n^{1,8-\varepsilon}$

1º Caso: Se f(n)∈O(n^$\log_b$a - ε) então T(n) = θ(n^$\log_b$a), ε > 0

n^2 ∈ O(n^$\log_3$7 - ε)

n^2 ≤ ∈ (n^1,8-ε)

n^2 ≤ c * (n^1,8-ε)

Podemos substituir ε por 0.1 e verificar que tal condição é falsa

2º Caso: Se f(n)∈ θ(n^$\log_b$a) então T(n) = θ(n^$\log_b$a * log n)

n^2 ∈ θ(n^$\log_3$7)

c2 * (n^1,8-ε) ≤ n^2 ≤ c2 * (n^1,8-ε)

A primeira desigualdade podemos verificar que seria verdadeiro, porém na segunda desigualdade não teríamos o mesmo resultado, já que nosso termo de maior ordem na direita da desigualdade é n^1,8, enquanto na esquerda é n^2. Portanto o segundo caso também será falso

3º Caso: f(n) Ω(n^$\log_b$a + ε) e a*f($\frac{n}{3}$) ≤ c*f(n) então T(n) = f(n)

Primeira condição:

$n^2 \in$ Ω(n^1,8+ε)
Podemos substituir ε por 0.1 e verificar que tal condição é verdadeira

Segunda condição:

Substituindo o $\frac{n}{3}$ na nossa f(n), temos $\frac{n^2}{9}$, logo:

7 * $\frac{n^2}{9} \leq$ c * n^2

Os termos de maior ordem possuem o mesmo grau, porém o termo da esquerda está sendo dividido.

Como nosso terceiro caso é verdadeiro nosso T(n) = θ(f(n)), então:

T(n) = θ(n^2).

Solução T: T(n) = T(n/2) + $n^{1/2}$
Pelo teorema mestre:

a = 1, b = 2, f(n) = $n^{1/2}$

1ºCaso: Se f(n)∈O(n^$\log_b$a - ε) então T(n) = θ(n^$\log_b$a), ε > 0

$n^{1/2}$ ∈ O(n^$\log_2$1 - ε)

$n^{1/2}$ ≤ c * n^$\log_2$1 - ε

$n^{1/2}$ ≤ c * (n^0-ε)

Como ε precisa ser maior do que 0, nosso n ficará elevado à um número negativo, portanto essa desigualdade não será verdadeira.

2º Caso: Se f(n)∈ θ(n^$\log_b$a) então T(n) = θ(n^$\log_b$a * log n)

$n^{1/2}$ ∈ θ(n^$\log_2$1)

c2 * n^$\log_2$1 ≤ $n^{1/2}$ ≤ c2 * n^$\log_2$1 c2 * (n^0) ≤ $n^{1/2}$ ≤ c2 * (n^0)

A primeira desigualdade podemos verificar que seria verdadeiro, porém na segunda desigualdade não teríamos o mesmo resultado, já que nosso termo de maior ordem na direita da desigualdade é n^0, enquanto na esquerda é $n^{1/2}$. Portanto o segundo caso também será falso

3º Caso: f(n) Ω(n^log$_b$a + ε) e a*f($\frac{n}{3}$) ≤ c*f(n) então T(n) = f(n)

Primeira condição:

$n^{1/2}$∈ Ω(n^0+ε)

Podemos substituir ε por 0.2 (1/5) e verificar que tal condição é verdadeira

Segunda condição:

Substituindo o $\frac{n}{2}$ na nossa f(n), temos $\frac{n^{1/2}}{2^{1/2}}$, logo:

$1 * \frac{n^{1/2}}{2^{1/2}} \leq c * n^{1/2}$

Os termos de maior ordem possuem o mesmo grau, porém o termo da esquerda está sendo dividido.

Como nosso terceiro caso é verdadeiro nosso T(n) = θ(f(n)), então:

T(n) = θ($n^{1/2}$).

Nossa solução T, seria superior pois está elevado a um expoente menor

3-)

a-)

Caso base: O caso base será quando n = 1, ou seja, temos somente um elemento. Então se esse único elemento for o procurado retornamos o index, senão retornamos 0 (-1 no caso da implementação em C).

Hipótese de indução: Estou supondo que sei resolver o problema para um vetor n ≤ ⌊ n/3 ⌋. Ou seja, caso estando nesse caso se o número procurado exista, sei determinar seu index e caso não exista retorno 0.

Caso geral: Resolver o problema para um vetor de n dados. Primeiro, deve-se calcular os índices que irão dividir o vetor em três partes, p1 e p2. Considerando que n = limDir - limEsq, teremos:

P1 = limEsq + (n/3), P2 = limEsq + (2*n/3);

Se nosso elemento for menor que a posição P1 do vetor, significa que ele pode estar na primeira parte e aplicamos a chamada da função nessa parte. Se for maior que o elemento na posição P1 e menor que o elemento da posição P2 do vetor ele pode estar na segunda parte e aplicamos a chamada recursiva nessa parte. Se for maior que P2 ele somente pode estar na terceira parte.

b-)

```
função Encontre(X, limEsq, LimDir, z)
início
        se limEsq = limDir então
                se X[limEsq] = z então retorne limEsq
                senão retorne 0;
        senão
             n := limDir - limEsq
             p1 := limEsq + ⌊ n/3 ⌋
             p2 := limEsq + ⌊ 2n/3 ⌋

             se ( z <= X[p1]) então Encontre(X, limEsq, p1, z)
             senão se (z > X[p2]) então retorne Encontre(X, p2+1, limDir, z)
             senão retorne Encontre(X, p1+1, p2, z)
fim
```

c-) T(n) = T(n/3) + c2

d-) Usando o método de derivação por substituição visto nos slides

$T(n) = T(n/3) + c, T(1) = c1$

Supondo $n = 3^K \rightarrow T(3^K) = T(3^{K-1}) + c, T(1) = c1$

$$
\begin{aligned}
T(n) &= T(3^{K-1}) + c_2 \\
&= (T(3^{K-2}) + c_2) + c_2 \\
&= ((T(3^{K-3}) + c_2) + c_2) + c_2 \\
&= T(3^0) + K * c_2 \\
&= c_1 + K * c_2
\end{aligned}
$$

Como $n = 3^K$, então $K = \log_3 n$. Portanto, $T(n) = \log_3 n * c_2$

Logo, $O(\log_3 n)$

e-) A ternária teria que fazer duas comparações, enquanto a binária só uma, então em determinados momentos essas comparações fariam diferença.

f-) Implementação no arquivo buscaTernaria.c

4-)

a-)

Caso base: O caso base será quando n = 1, ou seja, temos somente um index. Então se o elemento presente nesse index for maior que o elemento na posição anterior à esse index no vetor original, retornamos ele.

Hipótese indutiva: Estou supondo que sei resolver o problema para um vetor $n \leq \lceil n/2 \rceil$. Ou seja, caso estando nesse caso se o elemento do index atual for maior que seu anterior retorno ele.

Caso geral: Resolver o problema para um vetor de n dados. Primeiro, deve-se calcular o índice que irá dividir o vetor em duas partes, chamarei essa variável de “meio”.

meio = (limEsq + limDir)/2;

Se o elemento anterior for maior que o elemento na posição “meio” faremos uma chamada recursiva passando o limEsq e o meio- 1, pois isso significa que que nosso pico estará para esquerda. Caso contrário, faremos a chamada recursiva passando meio e limDir, pois nosso pico estará na direita.

b-)

```
função Encontre(X, limEsq, LimDir, z)
início
        se limEsq = limDir então
                se X[limEsq] > X[limEsq-1] então retorne limEsq
        senão
                meio:= (limEsq + limDir)/2
                se (X[meio-1] > X[meio] então
                        retorne encontre(X, limEsq, meio-1)

                senão retorne encontre(X, meio, limDir)

fim
```

c-) T(n) = T(n/2) + c

d-)
Usando o método de derivação por substituição visto nos slides
$T(n) = T(n/2) + c$, $T(1) = c_1$

Supondo $n = 2^{K} \rightarrow T(2^{K}) = T(2^{K-1}) + c$, $T(1) = c_1$

$T(n) = T(2^{K-1}) + c_2$
$= (T(2^{K-2}) + c_2) + c_2$
$= ((T(2^{K-3}) + c_2) + c_2) + c_2$
$= T(2^{0}) + K * c_2$

$= c_1 + K * c_2$

Como $n = 2^K$, então $K = \log_2 n$. Portanto, $T(n) = \log_2 n * c_2$
Logo, $O(\log_2 n)$

e-) Implementação no arquivo picoEnergia.c