Lista número 1

1) O que é eficiência assintótica?

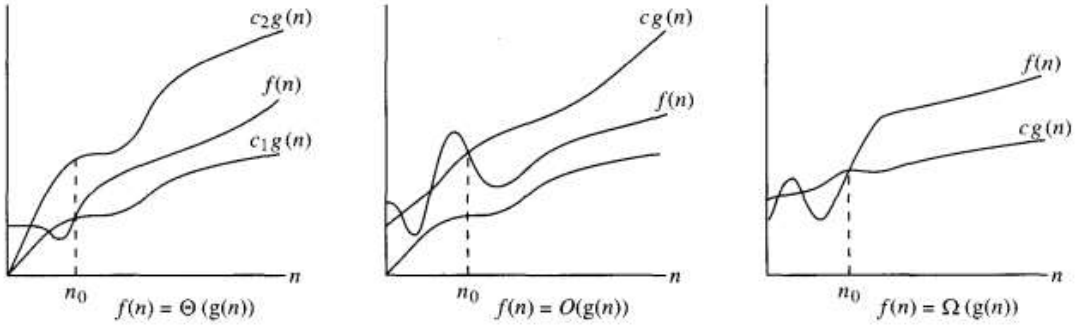
   Quando observamos tamanhos de entrada grandes o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução, estamos estudando a eficiência assintótica dos algoritmos. Ou seja, estamos preocupados com a maneira como o tempo de execução de um algoritmo aumenta com o tamanho da entrada no limite, a medida que o tamanho da entrada aumenta indefinidamente (sem limitação). Em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha para todas as entradas, exceto as muito pequenas.

2) Descreva as notações O, Ω, Θ

A notação Θ limita uma função ao intervalo entre fatores constantes. Escrevemos f(n) = Θ (g(n)) se existem constantes positivas n_o, c, e c, tais que, a direita de n_o, o valor de f(n) sempre reside entre c_1g(n) e c_2g(n) inclusive.

A notação **O** dá um limite superior para uma função dentro de um fator constante. Escrevemos f(n) = O(g(n)) se existem constantes positivas n_o e c tais que, a direita de n_o, o valor de f(n) sempre reside em ou abaixo de cg(n)

A notação Ω dá um limite inferior para uma função dentro de um fator constante. Escrevemos f(n) = Ω (g(n)) se existem constantes positivas n_O e c tais que, à direita de n_o, o valor de f(n) sempre reside em ou acima de cg(n)



3) Seja O(n ^{e}) = X, onde X representa o conjunto de funções que satisfaz a notação ***O*** para a função n^{e} . O conjunto $\{n^2, n\log n^2, n^{\frac{\pi}{2}}\log n^2, \frac{1}{n^e}\} \subset X?$ onde ***e*** é a constante de Euler. Essa afirmação é verdadeira ou falsa? Justifique!

Resposta:
A afirmação é verdadeira.
O conjunto X possui funções que têm uma taxa de crescimento no máximo da ordem de n^e.
Ao examinarmos cada elemento desse conjunto, temos:

$$\begin{aligned} n^2 &\in X \\ n\log n^2 &\in X \\ n^{\frac{\pi}{2}}\log n^2 \approx n^{1,67}\log n^2 &\in X \\ \frac{1}{n^e} &\in X \end{aligned}$$

4) Esse algoritmo procura o maior valor presente em um array “A” com n elementos e o armazena na variável M.

```
Int M= A[0];
For (i=0; i< n; i++){
        If(A[i]>= M){
                M=A[i]
‘       }
}
```

Faça a contagem das instruções e monte a a f(n) para o pior caso.

5) O que significa um algoritmo ser O(2) ou O(5)?

Resposta:
Significa que independentemente do tamanho da entrada n, o tempo de execução do algoritmo será constante.
Observe que se $f(n) = O(k)$ para uma função constante k, temos que:

$$f(n) \leq c \cdot k.$$

Se a constante k é 1 ou 2 ou 5 isso é indiferente pois a função $f(n)$ sempre será limitada por uma constante c vezes uma outra constante k.

6) Usando a definição formal de Θ prove que $6n^3 \neq \Theta(n^2)$.

Resposta:
Suponha que não, ou seja, suponha que $6n^3 = \Theta(n^2)$. Assim, pela definição formal da notação Θ, existem constantes positivas c_1, c_2 e n_0 tais que

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

para todo $n \geq n_0$. Neste caso, temos que $f(n) = 6n^3$, $g(n) = n^2$ e

$$c_1 n^2 \leq 6n^3 \leq c_2 n^2.$$

Ao dividirmos cada termo dessa inequação por n^2, temos:

$$c_1 \leq 6n \leq c_2.$$

Não existem constantes positivas $c_2 > 0$ e n_0 tais que $6n \leq c_2$ para todo $n \geq n_0$. Assim, a suposição original que é verdadeira, ou seja, $6n^3 \neq \Theta(n^2)$.

7) *Mostre que:*

a) 2n + 10 é O(n)

Resposta Se escolhermos c=3 e n_0=10
Teremos 0 ≤ 30 ≤ 30

b) 1/2n (n+1) é O(n^2)

Se escolhermos c=1 n=1

Termos ½ 1(1+1) ≤ 1.1^2*

c) n + √n é O(n);

Se escolhermos c=2 n_o=1
*Teremos 0 ≤ 1+√1 ≤ 2*1*

d) *n/1000 não é O(1)*

Não é, pois O(1) Não importa o tamanho da entrada, a complexidade permanece a mesma

8) *Considere as seguintes funções e coloque as funções em ordem de crescimento assintótico.*
a. log n
b. 2^n
c. n^2
*d. n * log n*
e. n^3
f 1
g n

As principais classes são: 1< log n < n< n* log< n^2< n^3<...2^n

9) Dada a função $n^2 + 1$. Demonstre que $f(n) \in O(n^2)$

$0 \leq n^2 + 1 \leq c * n^2, \forall n \geq n0$

Se escolhermos c=2 e n_0=1 , temos:

$0 \leq 2 \leq 2$

10) Dada a função $n^2 + 1$. Demonstre que $f(n) \in \Omega(n2)$

$0 \leq c * n^2 \leq n^2 + 1, \forall n \geq n_0$

Se escolhermos c=1 e n_0 , temos:

$1 \leq 2$

Não é difícil perceber que essa inequação é verdadeira para todo n_0 maior do que 1.

11)

Podemos definir o seguinte algoritmo para calcular a ordem de complexidade de algoritmos não recursivos:

- Escolher o parâmetro que indica o tamanho da entrada.
- Identificar a operação básica (comparação, atribuição)
- Estabeleça uma soma que indique quantas vezes sua operação básica foi executada.
- Utilize regras para manipulação de soma e formulas definindo uma função de complexidade.
- Encontre a ordem de complexidade.

a. Baseando-se no algoritmo acima determine a ordem de complexidade do seguinte algoritmo:

```
MaxMin(vetor v)
        max=v[1];
        min=v[1];
        para i=2 até n faça
                se v[1]> max então max=v[1]; fimse
                se v[1]< min então  min=v[1]; fimse
        fimpara;
fim.
```

b. Podemos dizer que o algoritmo de acima é O (n^2). Justifique.