Universidade de São Paulo – ICMC
Departamento de Ciências da Computação
SCC601 – Introdução a Ciência da Computação II

Professora: Rosane Minghim
Estagiária P.A.E: Nathalie Portugal

Lista de Exercícios Nº1

Análise Assintótica de algoritmos

1. Expresse a função $n^3/1000 - 100n^2 - 100n + 3$ em termos da notação Θ.

2. Sejam $f(n)$ e $g(n)$ funções assintoticamente não negativas. Usando a definição básica da notação Θ, prove que $max(f(n), g(n)) = \Theta(f(n)+g(n))$.

3. É verdade que $2n^3 + 5\sqrt{n} = \Theta(n^3)$

4. É verdade que $2^{n+1} = O(2^n)$? É verdade que $2^{2n} = O(2^n)$?

5. Demonstre que para duas funções quaisquer $f(n)$ e $g(n)$, temos $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.

6. *Dois algoritmos A e B possuem complexidade n5 e 2n respectivamente. Você utilizaria o algoritmo B ao invés do A, em qual caso. Exemplifique*

7. *Considere as seguintes funções e coloque as funções em ordem de crescimento assintótico.*
   a. *log n*
   b. *log (n^2)*
   c. *1000(log n)^2*
   d. *n/logn*
   e. *loglogn*

8. *Mostre que:*
   - *2n + 10 é O(n)*
   - *1/2n(n+1) é O(n^2);*
   - *n + √n é O(n);*
   - *n/1000 não é O(1);*
   - *1/2n^2 não é O(n);*
   - *1/2n^2 -3n é O(n^2).*

Professora: Rosane Minghim
Estagiária P.A.E: Nathalie Portugal

---

9. Prove que 2^n é $O(3^n)$, mas 3^n não é 2^n

10. Calcular a complexidade dos seguintes algoritmos em termos de n
    a. Complexidade →

```
proc mult_mat (in A, B : Matriz; out Res : Matriz);
var
        i, j, k : Integer;
begin
   for i ← 1 in n do
      for j ← 1 in n do
         Res[ i, j ] ← 0;
         for k ← 1 in n do
            Res[i,j] ← Res[i,j] + A[i,k] * B[k,j]
         end for;
      end for;
   end for;
end;
```

    b. Complexidade →

```
proc proc1 (in n : Integer);
var
   i, j, k : Integer;
begin
   for i ← 1 in n - 1 do
      for j ← i + 1 in n do
         for k ← 1 in j do
            sum ← 1
         end for;
      end for;
   end for;
end;
```

Professora: Rosane Minghim
Estagiária P.A.E: Nathalie Portugal

---

c. Complexidade →

```
proc proc2 (in n : Integer);
var
   i, j, sum : Integer;
begin
   sum ← 0;
   for i ← 1 in n - 1 do
      for j ← 1 in 2*n do
        sum ← sum + 1
      end for;
   end for;
end;
```

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.