UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO - UFRPE

DEPARTAMENTO DE COMPUTAÇÃO

Algoritmos e Estruturas de Dados

Prof. Filipe Cordeiro

1ª Lista de Exercícios – Complexidade

1. Por muitas vezes damos atenção apenas ao pior caso dos algoritmos, explique o por quê.
   A análise do pior caso dos algoritmos é importante para avaliar a viabilidade da solução em cenários com grandes entradas de dados. O estudo do pior caso nos permite analisar se o algoritmo irá completar sua execução baseado nos tamanhos de entrada do domínio do problema e verificar se será eficiente para sua tarefa. A análise do pior caso permite analisar o limite do algoritmo de acordo com o tamanho de entrada, possibilitando entender sua viabilidade mesmo no pior cenário possível.
2. Que tipo de crescimento melhor caracteriza cada uma dessas funções? (Constante, Linear, Polinomial, Exponencial)
   a) $\left(\frac{3}{2}\right)^n$ exponencial
   b) 1. constante
   c) (3/2)n. linear
   d) $2n^3$ cúbica
   e) $2n^2$ quadrática
   f) $3n^2$ quadrática
   g) 1000. constante
   h) $3n$. linear
3. Classifique as funções de acordo com o crescimento, do crescimento mais lento (na parte de cima) para o crescimento mais rápido (na parte de baixo)
   a. n
   b. n^3
   c. 1
   d. $\left(\frac{3}{2}\right)^n$
   e. n^2
   f. 2^n

   Ordem de crescimento: 1, n, n^2, n^3, $\left(\frac{3}{2}\right)^n$, 2^n

4. Classifique as funções de acordo com o crescimento, do crescimento mais lento para o mais rápido.
   a. 4n
   b. $8n^2$

c. $6n^3$
d. 64
e. $n\,log_2 n$
f. $n \log_6 n$
g. $\log_2 n$
h. $\log_8 n$
i. 8^{2n}

Ordem de crescimento: 64, $\log_8 n$, $\log_2 n$, 4n, $n \log_6 n$, $n\,log_2 n$, $8n^2$, $6n^3$

5. Seja um algoritmo com complexidade de tempo $a(n) = n^2 - n + 549$ e B um algoritmo com complexidade de tempo $b(n) = 49n + 49$. Qual algoritmo é melhor?
   O algoritmo b é mais rápido, pois possui complexidade assintótica linear, enquanto o algoritmo b possui complexidade quadrática. Isso indica que o tempo de execução de b cresce na mesma proporção de n, enquanto que no algoritmo *a*, ele dobra a cada aumento de *n*.

6. Considere um algoritmo de força bruta para calcular $a^n, onde\ n \in \mathbb{N}$. Pergunta-se:
   a. Qual a complexidade desse algoritmo?
      Complexidade exponencial
   b. Construa um algoritmo iterativo que calcule o valor em tempo $O(\log n)$
      Para construção do algoritmo, você pode se basear na seguinte fórmula de exponenciação:

$$yx^n = \begin{cases} (yx)(x^2)^{\frac{n-1}{2}}, se\ n\ é\ ímpar \\ y(x^2)^{\frac{n}{2}}, se\ n\ é\ par \end{cases}$$

      Pode fazer em qualquer linguagem, basta seguir a fórmula.

7. Estime a complexidade assintótica de cada um dos algoritmos abaixo. Obs: Não é tão simples quanto parece.

a.
```
int sum = 0;
for (int n = N; n > 0; n /= 2)
   for (int i = 0; i < n; i++)
      sum++;
```

b.
```
int sum = 0;
for (int i = 1; i < N; i *= 2)
   for(int j = 0; j < i; j++)
      sum++;
```

c.
```
int sum = 0;
for (int i = 1; i < N; i *= 2)
   for (int j = 0; j < N; j++)
      sum++;
```

a. linear (N + N/2 + N/4 + ...)
b. linear (1 + 2 + 4 + 8 + ...)

c. n logn (o primeiro for executa logn vezes, o segundo for executa sempre n vezes)

8. Diga a complexidade assintótica dos seguintes algoritmos:

```
(1)  sum = 0;
     for( i = 0; i < n; i++ )
         sum++;
(2)  sum = 0;
     for( i = 0; i < n; i++ )
         for( j = 0; j < n; j++ )
             sum++;
(3)  sum = 0;
     for( i = 0; i < n; i++ )
         for( j = 0; j < n * n; j++ )
             sum++;
(4)  sum = 0;
     for( i = 0; i < n; i++ )
         for( j = 0; j < i; j++ )
             sum++;
(5)  sum = 0;
     for( i = 0; i < n; i++ )
         for( j = 0; j < i * i; j++ )
             for( k = 0; k < j; k++ )
                 sum++;
```

9.

(1) O(n)

(2) $O(n^2)$

(3) $O(n^3)$

(4) $O(n^2)$

(5) $O(n^5)$