ALGORITMOS AVANÇADOS

Lista de Exercícios – 01

Luiz Leão – luizleao@gmail.com

http://www.luizleao.com

Exercício 01

Qual a importância de estudarmos a complexidade dos algoritmos?

Exercício 01 – Resposta

Qual a importância de estudarmos a complexidade dos algoritmos?

Tem como objetivo analisar o consumo de recursos necessários para a execução dos algoritmos e, assim, buscar formas de otimização do uso desses recursos

Exercício 02

Conceitue a Notação *O* e o seu significado para a análise da complexidade de um algoritmo.

Exercício 02– Resposta

Conceitue a Notação *O* e o seu significado para a análise da complexidade de um algoritmo.

Trata-se da notação que define o maior tempo de execução de um algoritmo sobre todas as entradas de tamanho n. Significa que ao avaliarmos a complexidade de algoritmo, devemos fazê-la esperando o cenário mais pessimista de sua execução

Exercício 03

Defina a análise da complexidade de algoritmos do melhor caso, a do caso médio e a do pior caso.

Exercício 03 – Resposta

Defina a análise da complexidade de algoritmos do melhor caso, a do caso médio e a do pior caso.

Melhor caso: Previsão do melhor cenário, executando o mínimo de instruções possíveis, tendo como reflexo o uso otimizado dos recursos computacionais,

Médio caso: Previsão de um consumo mediano de recursos computacionais

Pior caso: Previsão do pior cenário para a execução do algoritmo, prevendo que ele execute o máximo de instruções possíveis

Exercício 04

Por que a análise da complexidade, na maioria das vezes, é feita sob a ótica do pior caso ou mesmo do caso médio, tornando-se pouco prática a análise do melhor caso?

Exercício 04 – Resposta

Por que a análise da complexidade, na maioria das vezes, é feita sob a ótica do pior caso ou mesmo do caso médio, tornando-se pouco prática a análise do melhor caso?

É importante prever o cenário de execução mais pessimista do algoritmo, para que não geremos falsas expectativas quanto a eficiência do programa

Exercício 05

Qual a diferença entre levantar a complexidade de um algoritmo iterativo e levantar a complexidade de um algoritmo recursivo?

Exercício 05 – Resposta

Qual a diferença entre levantar a complexidade de um algoritmo iterativo e levantar a complexidade de um algoritmo recursivo?

O algoritmo iterativo é calculado de forma linear, somando as instruções que possivelmente serão executadas

O algoritmo recursivo é um pouco mais imprevisível pois depende dos componentes: Complexidade da base, Complexidade do Núcleo e Profundidade da Recursão

Exercício 06

Dado o algoritmo abaixo, determine:

a) A ordem de complexidade das linhas 06, 07 e 08.

b) A ordem de complexidade das linhas 10, 11 e 12.

c) A ordem de complexidade do algoritmo completo utilizando uma operação com a Notação *O*, envolvendo os resultados obtidos nos itens **a** e **b**.

Exercício 06 – Resposta

Dado o algoritmo abaixo, determine:

a) A ordem de complexidade das linhas 06, 07 e 08.

b) A ordem de complexidade das linhas 10, 11 e 12.

c) A ordem de complexidade do algoritmo completo utilizando uma operação com a Notação *O*, envolvendo os resultados obtidos nos itens **a** e **b**.

```
01 inicio
02      i, j: inteiro
03      A: vetor inteiro de n posicoes
04       i = 1
05
06      enquanto (i < n) faca
07              A[i] = 0
08              i = i + 1
09
10      para i = 1 ate n faca
11              para j = 1 ate n faca
12                      A[i] = A[i] + (i*j)
13  fim
```

Exercício 07

Calcule a complexidade de tempo:

a) Para o melhor caso

b) Para o pior caso, do o algoritmo “maiorElementoNoArray” apresentado (Obs.: A disposição dos dados apresentada pelo array A[ ] poderá ser modificada).

Exercício 07

```
1   #include <iostream>
2
3   int maiorElementoNoArray(int A[],int n)
4   {
5     int tmpMax = A[0];
6     int i;
7
8     for(i =1; i<n;i++){
9       if(tmpMax < A[i]){
10        tmpMax = A[i];
11      }
12    }
13    return tmpMax;
14  }
15
16  int main(int argc, char** argv) {
17      int A[]={10,1,5,9,3,12,0,11,19,6,7};
18      printf("O maior e:%d", maiorElementoNoArray(A,11));
19      return 0;
20  }
```

Exercício 08

Dados os seguintes trechos de código, encontre a complexidade para cada um deles: (**Obs.**: Considere o bloco “sequência de comandos” com ordem de complexidade constante, ou seja: ***O*(1)**).

(a) Dois laços seguidos:

```
for (i = 0; i < N; i++) {
        <<sequência de comandos>>
}
for (j = 0; j < M; j++) {
        <<sequência de comandos>>
}
```

O que acontecerá se for trocada a complexidade do segundo laço por N no lugar de M?

Exercício 09

Dados os seguintes trechos de código, encontre a complexidade para cada um deles: (**Obs.**: Considere o bloco “sequência de comandos” com ordem de complexidade constante, ou seja: ***O*(1)**).

(b) Primeiro um laço aninhado e, em seguida, um laço não aninhado:

```
for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
                <<sequência de comandos;>>
        }
}
for (k = 0; k < N; k++) {
        <<sequência de comandos;>>
}
```

Exercício 10

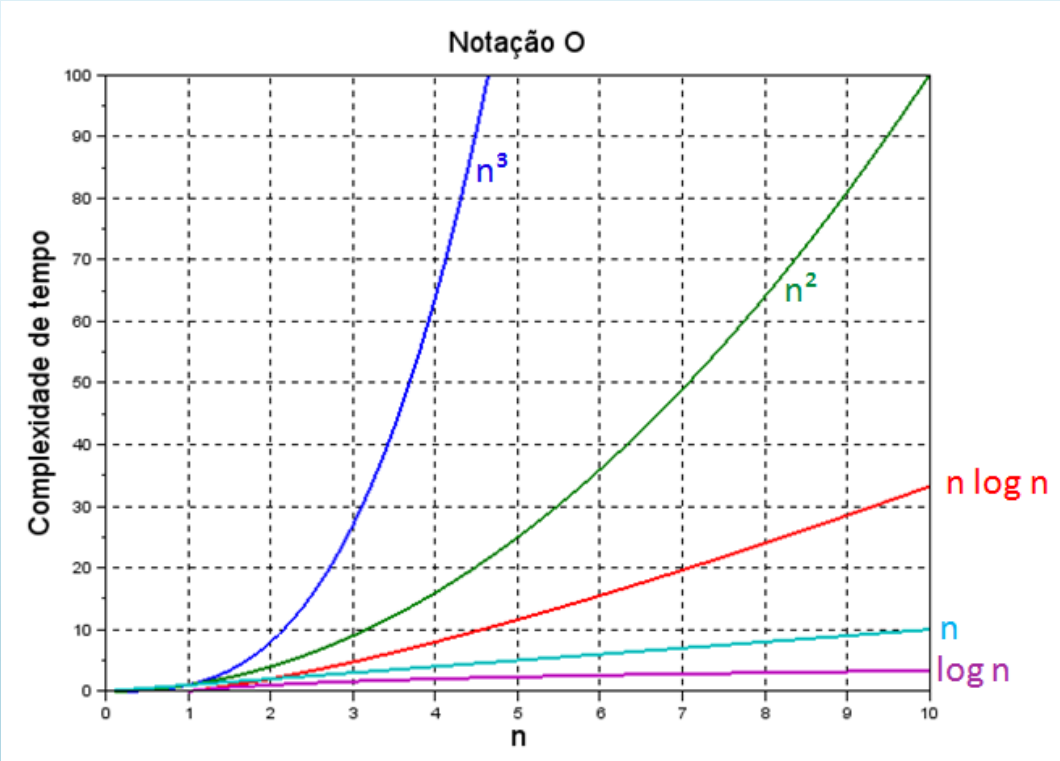
Dados os seguintes trechos de código, encontre a complexidade para cada um deles: (**Obs.**: Considere o bloco “sequência de comandos” com ordem de complexidade constante, ou seja: ***O*(1)**).

(c) Laço aninhado onde o número de vezes do laço interno é função do valor do índice no laço externo:

```
for (i = 0; i < N; i++) {
        for (j = i; j < N; j++) {
                <<sequência de comandos>>
        }
}
```

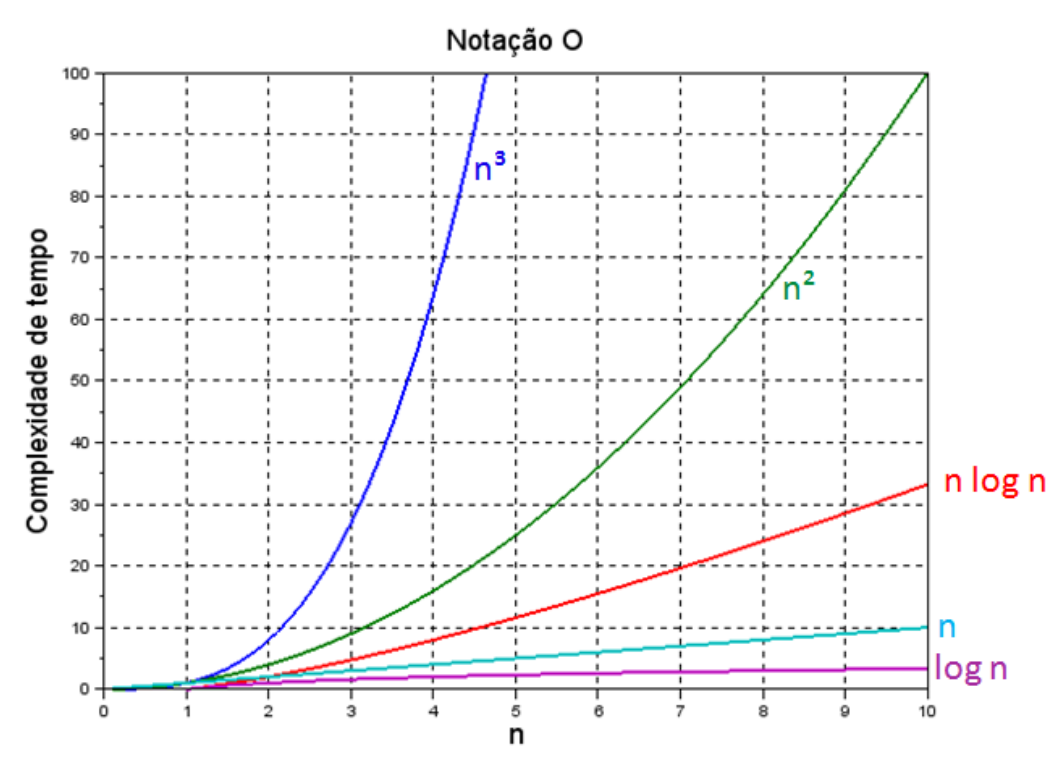
Exercício 11

Considerando o gráfico abaixo, podemos perceber que ele exibe uma comparação entre as grandezas da Notação O para alguns tipos de algoritmos. Pesquise e comente a eficiência de cada tipo de algoritmo, no tratamento de entradas para grandes valores de **n**.



Exercício 11 – Resposta

Considerando o gráfico abaixo, podemos perceber que ele exibe uma comparação entre as grandezas da Notação O para alguns tipos de algoritmos. Pesquise e comente a eficiência de cada tipo de algoritmo, no tratamento de entradas para grandes valores de **n**.



$O(n^3)$ e $O(n^2)$: Custo computacional elevado a medida que o n aumenta

$O(n\log n)$ e $O(n)$: Aumento linear do custo, sendo $O(n\log n)$ mais caro

$O(\log n)$: É o algoritmo de menor custo, pois chega num ponto que o custo estabiliza

Exercício 12

Obtenha a função de complexidade ***f (n)*** dos algoritmos abaixo. Defina a ordem de complexidade para o pior caso e para o melhor caso

a)

```
1  void Procedimento1(int n) {
2     int i, j, x, y;
3     x = y = 0;
4     for(i = 1; i <= n; i++) {
5        for(j = i; j <= n; j++)
6           x = x + 1;
7        for(j = 1; j < i; j++)
8           y = y + 1;
9     }
10 }
```

Exercício 13

Obtenha a função de complexidade ***f* (*n*)** dos algoritmos abaixo. Defina a ordem de complexidade para o pior caso e para o melhor caso

b)

```
1  void Procedimento2() {
2     int i, j, k, x;
3     x = 0;
4     for(i = 1; i <= n; i++) {
5        for(j = 1; j <= n; j++)
6           for(k = 1; k <= j; k++)
7              x = x + j + k;
8     x = i;
9  }
```