5. Eficiência de algoritmos

Matemática como ferramenta para análise de algoritmos

Função constante

f(n) = c . g(n)

A função constante, então, costuma representar a quantidade de passos que um algoritmo executa em uma determinada aplicação.

Função logarítmica

f(n)* = log*b n

O log é a quantidade de vezes que se pode dividir um determinado número pela base até que o resultado das divisões subsequentes seja próximo ou igual a 1.

Função linear

f(n) = n

Isso significa que toda entrada n informada à função resultará em uma saída igual à entrada. Essa função é facilmente encontrada em diversos cenários da computação. Toda vez que é executada uma operação para n elementos, por exemplo, no caso de operações que percorrem um vetor por completo, a função que descreve essa operação é justamente a função linear

Função *n* log *n*

f(n) = n .* log *n

Ela possui uma taxa de crescimento superior à função linear, mas é bem menor quando comparada a funções polinomiais de grau 2 ou mais. A função n log n é o alvo para pesquisadores que querem otimizar um problema de ordem quadrática

Função quadrática

$f(n) = n^2$

Isso ocorre, dentro do cenário da computação, geralmente em algoritmos que possuem laços de repetição aninhados. Em outras palavras, isso quer dizer que existe um laço de repetição externo que executa *n* vezes, linearmente, e um outro laço de repetição interno que também executa linearmente *n* vezes

Função cúbica e demais polinomiais

$$f(n) = n^3$$

Assim como a função quadrática, a função cúbica atribui o produto de uma entrada *n* pelo próprio *n*, só que, nesse caso, três vezes. A função cúbica é menos comum no cenário de análise de algoritmos

Função exponencial

$$f(n) = b^n$$

O resultado de *f(n)* é obtido através da multiplicação da base por ela mesma, o número de vezes informado pela entrada *n*. Na análise de algoritmos, a base 2 é tida como a mais comum. Portanto, em geral, quando se fala de função expoente ou exponencial, subentende-se que se trata da função $f(n) = 2^n$

Comparativo das taxas de crescimento

Quadro 1 – Tabela representando as sete funções para a análise de algoritmos, sendo da mais rápida até a mais demorada, da esquerda para a direita

Constante	Logarítmica	Linear	*N log N*	Quadrática	Cúbica	Exponencial
1	log *n*	*n*	*n* log *n*	n^2	n^3	2^n

Identificando a função do algoritmo

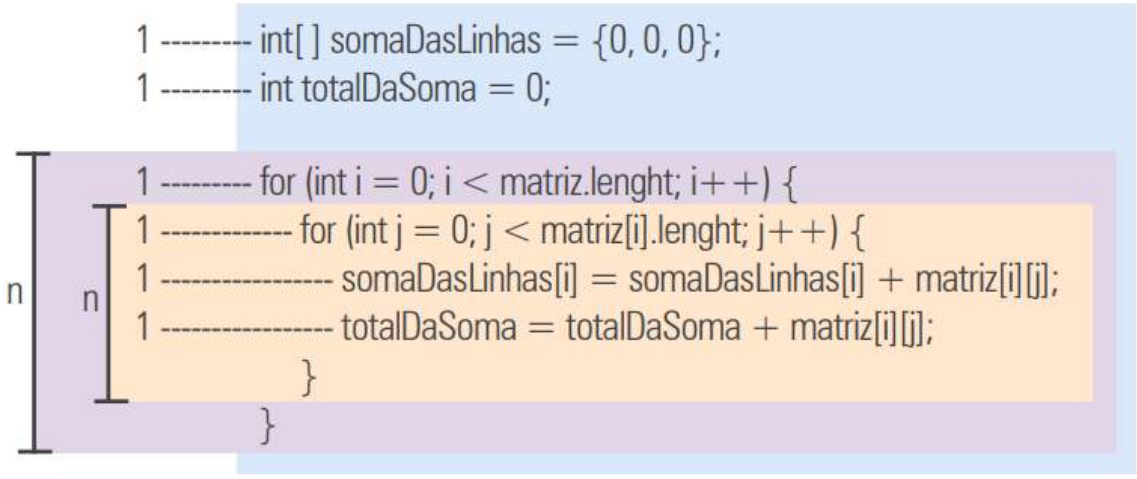
```
public class SomatorioMatriz {
   public static void main(String[] args) {
     int[][] matriz = {
       {1, 2, 3, 4, 5},
       {6, 7, 8, 9, 10},
       {11, 12, 13, 14, 15}
     };

     int[] somaDasLinhas = {0, 0, 0};
     int totalDaSoma = 0;

     for(int i = 0; i < matriz.length; i++) {
       for(int j = 0; j < matriz[i].length; j++) {
          somaDasLinhas[i] = somaDasLinhas[i] + matriz[i][j];
          totalDaSoma = totalDaSoma + matriz[i][j];
```

```
        }
      }

      System.out.printf("O soma total eh: %d\n", totalDaSoma);
   }
}
```



$2n^2 + 2$

Notação Big-O

- Quando uma análise assintótica é realizada em um algoritmo, o objetivo principal é encontrar a função que determine o limite superior do tempo de processamento do algoritmo. O limite superior também é conhecido como o cenário de pior caso de um algoritmo. Pense que, se o algoritmo tiver um bom desempenho dentro do pior cenário possível, ele irá consegui processar as informações dos cenários menos complexos de uma maneira muito mais tranquila
- A notação Big-O representa justamente o limite superior, baseando-se no grau mais alto encontrado para n