Primeira Lista de Exercícios: Recursividade e Complexidade de Algoritmos.

1. Um problema típico em ciência da computação consiste em converter um número da sua forma decimal para a forma binária. Por exemplo, o número 12 tem a sua representação binária igual a 1100. A forma mais simples de fazer isso é dividir o número sucessivamente por 2, onde o resto da *i-ésima* divisão vai ser o dígito *i* do número binário (da direita para a esquerda). Desenvolva algoritmos recursivos para impressão de um número natural em base binária.
2. Considere um sistema numérico que não tenha a operação de adição implementada e que você disponha somente dos operadores (funções) `sucessor` e `predecessor`. Então, pede-se para escrever uma função recursiva que calcule a soma de dois números x e y através desses dois operadores.
3. Desenvolva algoritmos recursivos para os seguintes problemas:
    a. Multiplicação de dois números naturais, através de somas sucessivas (Ex.: 6 $*$ 4 = 4+4+4+4+4+4).
    b. Multiplicação de dois números naturais, através de incrementos sucessivos.
    c. Inversão de uma string.
    d. A partir de um vetor de números inteiros, calcule a soma e o produto dos elementos do vetor.
    e. Verifique se uma palavra é palíndromo
4. O problema da Torre de Hanoi é comumente utilizado como exemplo de um algoritmo recursivo. Neste link vocês encontram uma implementação visual deste problema. O objetivo do problema é mover todos os discos do pino de origem para o pino destino (o pino da esquerda) para o pino destino (o pino da direita). Você deve seguir as regras básicas que são: Somente pode mover um disco de cada vez e um disco maior nunca pode ficar em cima de um disco menor.
    a. Discuta a forma de resolver o problema;
    b. Faça uma implementação em C que permita resolver o problema para qualquer quantidade de discos;
    c. Determine a complexidade do problema

5. A **sequência de Fibonacci** é uma sequência de elementos $f_1, \ldots, f_n$, definida do seguinte modo: $f_1 = 0; f_2 = 1; f_j = f_{j-1} + f_{j-2}\ para\ j > 2$.
    a. Elaborar um algoritmo recursivo para calcular a **sequência de Fibonacci** para qualquer valor de n;
    b. Determinar o número de chamadas recursivas e a complexidade do algoritmo implementado;
    c. construir um algoritmo não recursivo para calcular o elemento n da sequência, cuja complexidade seja linear com n;
6. Implemente uma função recursiva soma(n) que calcula o somatório dos n primeiros números inteiros. Qual é a ordem de complexidade da sua função? Qual seria a ordem de complexidade dessa mesma função implementada sem utilizar recursividade? O que você conclui?
7. Considere a seguinte sequência definida a partir de uma equação de diferenças de segunda ordem: $y_n = 2y_{n-1} + y_{n-2} + n,\ com\ y_1 = 0\ \ e\ y_2 = 0$
    a. Elaborar um algoritmo recursivo para calcular esta sequência para qualquer valor de n;
    b. Determinar o número de chamadas recursivas e a complexidade do algoritmo implementado;
    c. construir um algoritmo não recursivo para calcular o elemento n da sequência, cuja complexidade seja linear com n;