1. Algoritmo simples para multiplicar dois números naturais:

```
=======================================
Algoritmo: mult_somas
Entrada: naturais "y" e "z".
Saída: um natural.
---------------------------------------
1. SE y <= z
2. | menor := y , maior := z
3. SENÃO
4. | menor := z , maior := y
5. resp := 0
6. PARA i DE 1 A menor
7. | resp := resp + maior
8. RETORNE resp.
=======================================
```

2. COMPLEXIDADE: $\theta(\text{máx}\{ 1 , \text{mín}\{y,z\} \})$, no modelo de computação que costumamos utilizar, no qual cada soma e cada atribuição de inteiros custa tempo $\theta(1)$.

3. MODELO DE COMPUTAÇÃO SOBRE BITS: considere que um inteiro está armazenado num vetor de bits, e que operações elementares sobre um bit custam tempo $\theta(1)$: comparar um bit com uma constante (0 ou 1), comparar dois bits quanto à igualdade, atribuir um valor a um bit, etc.

Operações sobre vários bits podem custar mais que $\theta(1)$. Por exemplo, saber se um inteiro de "n" bits é zero envolve percorrer os "n" bits e verificar se todos são zero, e por isso custa $O(n)$ operações de bit NO PIOR CASO (no melhor caso, primeiro bit é descoberto valer 1 e então não é necessário considerar os demais bits). De forma semelhante, somar dois inteiros de "n" bits cada envolve percorrer os dois vetores de bits, e custa $\theta(n)$ operações de bit.

IMPORTANTE: neste modelo, realizar, sobre um inteiro de "n" bits, um deslocamento de "k" bits à esquerda ou à direita custa $\theta(n)$, INDEPENDENTEMENTE DO VALOR DE "k", como se pode ver pelo algoritmo abaixo:

```
=========================================================================
Algoritmo: desloc_esq
Entrada: (1) vetor de bits bit[1..n]; (2) inteiro "k".
Saída: nenhuma.
-------------------------------------------------------------------------
1. i = 1
2. ENQUANTO i+k <= n
3. | bit[i] := bit[i+k]
4. | ++i
5. ENQUANTO i <= n
6. | bit[i] := 0
7. | ++i
=========================================================================
```

4. No modelo de computação sobre bits, o algoritmo "mult_somas" acima tem o custo descrito a seguir, supondo que os inteiros "y" e "z" dados como entrada estão armazenados em "n" bits cada:

   - Linha 1: $O(n)$.
   - Linhas 2 e 4: $\theta(n)$.
   - Linha 3: $O(1)$.
   - Linha 5: $\theta(n)$.

- Linha 6: O(2^n), no total. (Veja a observação abaixo.)
- Linha 7: O(2^n * n), no total (pois são O(2^n) iterações, cada uma ao custo de θ(n) operações de bit).
- Linha 8: O(n).

Logo, o algoritmo custa O(2^n * n) operações de bit.

5. OBSERVAÇÃO: incrementar um inteiro de "n" bits pode custar ω(1) operações de bit: por exemplo, para incrementar o inteiro 00001111, precisamos fazer 5 atribuições, resultando em 00010000. Entretanto, como demonstrado na seção 17.1 do nosso livro-texto, incrementar um inteiro "k" vezes custa O(k), desde que ele comece valendo zero. De forma semelhante, é possível mostrar que "k" decrementos num de um inteiro de "n" bits custa O(máx{n,k}). Portanto, como nós podemos modificar o laço da linha 6 do algoritmo "mult_somas" para "ENQUANTO menor > 0" e decrementar "menor" em cada iteração, e como nós podemos combinar o teste sobre a positividade de "menor" com o decremento da variável, então o custo do controle do laço é O(máx{n,k}), sendo "k" o valor de "menor" no início do algoritmo. Como "y" e "z" estão armazenadas em "n" bits, então "menor" vale, no máximo, 2^n - 1, e portanto o custo do controle do laço é O(2^n) operações de bit.

6. Algoritmo baseado em deslocamentos:

```
========================================
Algoritmo: mult_desloc
Entrada: naturais "y" e "z".
Saída: um natural.
----------------------------------------
01. SE y <= z
02. | m := y , M := z
03. SENÃO
04. | m := z , M := y
05. R := 0
06. ENQUANTO m > 0
07. | SE m % 2 = 1
08. | | R := R + M
09. | M := M << 1 // equivale a M := M*2
10. | m := m >> 1 // equivale a m := piso(m/2)
11. RETORNE R.
========================================
```

7. COMPLEXIDADE: no modelo de computação habitual, sendo k = mín{y,z}, o algoritmo acima executa em tempo θ(1) se k = 0. Se k > 0, esse tempo é θ(lg k), pois as linhas 1 a 5 e 11 executam em θ(1) e o laço da linha 6 executa 1+piso(lg k) iterações de custo θ(1) cada.

   No modelo de computação sobre bits, se "y" e "z" possuem "n" bits cada, então as linhas 1 a 5 e 6 executam θ(n) operações de bit e o laço da linha 6 executa O(n) iterações, em cada uma das quais são realizadas θ(n) operações de bit. Logo, o algoritmo executa O(n^2) operações de bit.

   OBSERVAÇÃO: a complexidade do algoritmo acima é muito melhor que a do algoritmo simples baseado em somas. Crucial para essa melhora no tempo de execução é o fato de que o algoritmo mais rápido tira proveito da representação dos números no computador (o que não é explorado pelo primeiro algoritmo).

8. Algoritmo de divisão e conquista para multiplicação de inteiros de "n" bits cada (por simplicidade, suponha que n é potência de dois): se

    y = a*2^(n/2) + b   ,   z = c*2^(n/2) + d   ,   x = yz

    então x = ac*2^n + (ad + bc)*2^(n/2) + bd .

9. COMPLEXIDADE: acima, x é calculado por meio de:

- 4 produtos de números de n/2 bits (ac, ad, bc, bd)
- 2 deslocamentos à esquerda (" * 2^n" e " * 2^(n/2)")
- 3 somas

Logo, o custo do algoritmo é dado por:

  T(n) = 4*T(n/2) + O(n) --> operações de bit

e, pelo caso 1 do teorema mestre, T(n) = θ(n^2). Portanto, esse algoritmo não tem performance melhor que o algoritmo "mult_desloc" acima.

10. Um algoritmo assintoticamente mais rápido: observe que o produto "x" de "y" e "z" também pode ser calculado assim:

- u = (a+b)(c+d)
- v = ac
- w = bd
- x = v*2^n + (u-v-w)*2^(n/2) + w

Observe que, além de somas e deslocamentos, o algoritmo acima possui apenas 3 produtos, dois dos quais ("v" e "w") entre números de n/2 bits e um entre números de n/2 + 1 bits (a soma de 2 números de n/2 bits tem até n/2 + 1 bits). Este último produto pode ser calculado da seguinte forma conveniente: sejam

  a + b = p*2^(n/2) + q --> "p" é somente o dígito mais significativo
  c + d = r*2^(n/2) + s --> "q" é somente o dígito mais significativo

Logo,

  (a+b)(c+d) = pr*2^n + (ps + qr)*2^(n/2) + qs .

Além disso, o custo de calcular os termos acima é:

- pr: θ(1) (é o produto de dois bits)
- ps: O(n/2), pois ps = s se p = 1 e ps = 0 se p = 0.
- qr: idem.
- qs: produto de dois números de n/2 bits cada.
- restante: são deslocamentos e somas.

Logo, o termo (a+b)(c+d) pode ser calculado em tempo T(n/2) + O(n), e portanto o custo de calcular x pelo presente algoritmo é

  T(n) = 3*T(n/2) + O(n)

e, pelo teorema mestre, T(n) = θ(n^lg 3) = o(n^2). Logo, o presente algoritmo é assintoticamente mais rápido que os demais no modelo de operações sobre bits em questão.

--> OBSERVAÇÃO: o modelo de operações de bit NÃO É utilizado abaixo.

11. EXERCÍCIO: ideia semelhante pode ser utilizada para se obter um algoritmo o(n^3) para o problema da multiplicação de duas matrizes n x n, conforme explicação abaixo:

a) Dadas duas matrizes n x n "Y" e "Z", o objetivo é calcular a matriz X = Y*Z.

b) Recapitule que o algoritmo direto para a obtenção de X executa em tempo θ(n^3) (no modelo de computação habitual).

c) Aplicando a ideia de divisão-e-conquista, divida as matrizes em blocos (n/2)x(n/2):

```
Y = | A B |, Z = | E F |,  X = | I J |
    | C D |      | G H |       | K L |
```

Logo: I = AE + BG
       J = AF + BH
       K = CE + DG
       L = CF + DH

d) Computar X pelas 4 equações acima leva a um algoritmo que executa em tempo

T(n) = 8*T(n/2) + O(n^2)

e, pelo teorema mestre, T(n) = θ(n^3), o que não é melhor que o tempo do algoritmo direto.

e) A SUA TAREFA NESTA QUESTÃO É mostrar que calcular X pela maneira descrita abaixo é correto e leva tempo O(n^(lg 7)) = o(n^3):

- M1 = (A+C)(E+F)
- M2 = (B+D)(G+H)
- M3 = (A-D)(E+H)
- M4 = A(F-H)
- M5 = (C+D)E
- M6 = (A+B)H
- M7 = D(G-E)

- I = M2 + M3 - M6 - M7
- J = M4 + M6
- K = M5 + M7
- L = M1 - M3 - M4 - M5.