O PROBLEMA DO CORTE DE HASTE (Cormen, 3ª ed., §15.1)

O problema do corte de haste é o de encontrar uma maneira de cortar em pedaços uma haste de tamanho "n" maximizando a soma dos preços dos pedaços. Tanto o tamanho da haste original quanto o de cada pedaço é um número natural, e, para todo "i" de 1 a "n", um pedaço de tamanho "i" tem preço P[i].

Dado um vetor P[1..n] de números reais tal que n >= 1 e P[i] >= 0 ∀i, e dado um natural m ∈ [1..n], seja lmáx(m) o maior valor de SOMATÓRIO_{i=1}^k P[t[i]] tal que:

1. k ∈ [1..m].
2. t[i] ∈ [1..m], ∀i ∈ [1..k].
3. m = SOMATÓRIO_{i=1}^k t[i].

Assim sendo, o seguinte algoritmo resolve o problema por meio de uma aplicação direta da técnica de divisão-e-conquista:

```
==============================================================================
Algoritmo:    corte_de_haste_força_bruta
Entrada:      vetor P[1..n] de números reais.
Pré-condição: n >= 1 e P[i] >= 0 ∀i.
Saída:        um número real "l"  // o maior "lucro" possível
Pós-condição: l = lmáx(l).
------------------------------------------------------------------------------
1. SE n = 1
2. | RETORNE P[1].
3. lucro := P[n]
4. PARA i DE 1 A n-1
5. | lucro := máx { lucro, P[i] + corte_de_haste_força_bruta(P[1..n-i]) }.
6. RETORNE lucro.
==============================================================================
```

É fácil verificar que o algoritmo acima faz várias chamadas recursivas repetidas. Isso é evitado na versão memoizada a seguir:

```
==============================================================================
Algoritmo:    corte_de_haste_memo
Entrada:      vetor P[1..n] de números reais.
Pré-condição: n >= 1 e P[i] >= 0 ∀i.
Saída:        um número real "l"  // o maior "lucro" possível
Pós-condição: l = lmáx(l).
------------------------------------------------------------------------------
1. SE n = 1
2. | RETORNE P[1].
3. solucoes[1] := P[1]
4. PARA i DE 2 A n
5. | solucoes[i] := -1
6. RETORNE corte_de_haste_rec(n, P, solucoes).
==============================================================================
```

```
===============================================================================
Algoritmo: corte_de_haste_rec
Entrada:   um inteiro "t" e vetores P[1..n] e solucoes[1..n] de números reais.
Saída:     um número real  // o maior "lucro" possível
// Por simplicidade, omitimos as pré- e pós-condições.
-------------------------------------------------------------------------------
1. SE solucoes[t] = -1
2. | lucro := P[t]
3. | PARA i DE 1 A t-1
4. | | lucro := máx { lucro, P[i] + corte_de_haste_rec(t-i, P, solucoes) }.
5. | solucoes[t] := lucro
6. RETORNE solucoes[t].
===============================================================================
```

Observando que o algoritmo acima essencialmente preenche o vetor "solucoes" do início para o fim, podemos escrever uma versão (de programação dinâmica) mais direta do algoritmo:

```
===============================================================================
Algoritmo:    corte_de_haste_PD
Entrada:      vetor P[1..n] de números reais.
Pré-condição: n >= 1 e P[i] >= 0 ∀i.
Saída:        um número real "l".
Pós-condição: l = lmáx(l).
-------------------------------------------------------------------------------
1. PARA t DE 1 A n
2. | lucro := P[t]
3. | PARA i DE 1 A t-1
4. | | lucro := máx { lucro, P[i] + solucoes[t-i] }.
5. | solucoes[t] := lucro
6. RETORNE solucoes[n].
===============================================================================
```

O algoritmo abaixo retorna uma solução completa para o problema:

```
===============================================================================
Algoritmo:    corte_de_haste_PD_completo
Entrada:      vetor P[1..n] de números reais.
Pré-condição: n >= 1 e P[i] >= 0 ∀i.
Saída:        um número real "l" e um vetor de números naturais.
Pós-condição: l = lmáx(l). // por simplicidade, omitida a parte sobre o vetor.
-------------------------------------------------------------------------------
1. PARA t DE 1 A n
2. | lucro := P[t], tam[t] := t
3. | PARA i DE 1 A t-1
4. | | lucro_i := P[i] + solucoes[t-i]
5. | | SE lucro_i > lucro
6. | | | lucro := lucro_i
7. | | | tam[t] := i
8. | solucoes[t] := lucro
9. RETORNE (solucoes[n], tam).
===============================================================================
```

1. EXERCÍCIO: com relação ao algoritmo acima:

   a) É correto substituir t-1 por $\lfloor t/2 \rfloor$ na linha 3 do algoritmo?

   b) O algoritmo executa em tempo $o(n^2)$ ou $\theta(n^2)$?