1. Algoritmo direto para o n-ésimo número de Fibonacci:

```
================================================
Algoritmo: fib
Entrada: inteiro positivo n.
Saída: inteiro.
------------------------------------------------
1. SE n < 3
2. | RETORNE 1.
3. RETORNE fib(n-2) + fib(n-1).
================================================
```

2. EXERCÍCIO: mostre que o algoritmo acima executa em tempo Ω(2^(n/2)).

3. Algoritmo iterativo, mais rápido:

```
================================================
Algoritmo: fib_it
Entrada: inteiro positivo n.
Saída: inteiro.
------------------------------------------------
1. SE n < 3
2. | RETORNE 1.
3. fibim2 := 1, fibim1 := 1,
4. PARA i DE 3 A n
5. | fibi := fibim2 + fibim1, fibim2 := fibim1 , fibim1 := fibi
6. RETORNE fibi.
================================================
```

Observe que esse algoritmo executa em O(n), o que é muito melhor que Ω(2^(n/2)).

4. EXERCÍCIO (opcional): como você argumentaria que esse algoritmo é correto,
   por meio de variantes e invariantes de laço?

5. Versão "memoizada" do algoritmo de Fibonacci:

```
================================================
Algoritmo: fib_memo
Entrada: inteiro positivo n.
Saída: inteiro.
------------------------------------------------
1. SE n < 3
2. | RETORNE 1.
3. Obtenha vetor auxiliar A[1..n] de inteiros
4. A[1] := A[2] := 1
5. PARA i DE 3 A n
6. | A[i] := 0
7. RETORNE fib_rec(n,A).
================================================

================================================
Algoritmo: fib_rec
Entrada: inteiro positivo n, vetor "A" de inteiros.
Saída: inteiro.
------------------------------------------------
1. SE A[n] = 0
2. | A[n] := fib_rec(n-2,A) + fib_rec(n-1,A)
3. RETORNE A[n].
================================================
```

6. EXERCÍCIO (opcional): como você argumentaria que esse algoritmo também
   executa em tempo O(n)?

7. RESUMO DA ESTRATÉGIA DE PROGRAMAÇÃO DINÂMICA, EXEMPLIFICADA ACIMA:

a) Solução de instâncias grandes por meio da solução de instâncias menores, como em divisão-e-conquista.

b) Se as soluções das instâncias menores compartilham cálculos, então aplicar divisão-e-conquista diretamente leva a cálculos repetidos; a ideia, portanto, é contornar essa repetição, resolvendo do pequeno para o grande, ao invés de do grande para o pequeno.

c) Caso desejemos, nós ainda podemos manter a forma recursiva do algoritmo; para tanto, porém, nós devemos manter um registro dos cálculos que já foram feitos, e só fazer um cálculo se ele não tiver sido feito ainda: MEMOIZAÇÃO.

8. PROBLEMA DA LINHA DE MONTAGEM: (Cormen, 2ª edição, seção 15.1:

http://books.google.com.br/books?id=NLngYyWFl_YC&pg=PA324&lpg=PA324&dq=assembly+line+problem+dynamic+programming&source=bl&ots=ByOmHH1jJa&sig=-u4NpvBcN4DXCegSh8PE-5oM1qo&hl=pt-BR&sa=X&ei=JPwWVL_8NeHFigKmuYGIBQ&sqi=2&ved=0CFgQ6AEwBw

)

Dados de entrada:

- n: número de setores de cada uma das duas linhas de montagem.

- a_{i j}: tempo gasto no j-ésimo setor da i-ésima linha de montagem (a: "assembly").

- e_i: tempo gasto na entrada da i-ésima linha de montagem (e: "entry").

- t_{i j}: tempo gasto para transferir a partir do j-ésimo setor da linha de montagem "i" (t: "transfer").

- x_i: tempo gasto na saída da i-ésima linha de montagem (x: "exit").

Além disso, nós vamos calcular:

- M_{i j}: melhor tempo para se chegar ao j-ésimo setor da i-ésima linha de montagem.

```
============================================================
Algoritmo: LM
Entrada: os dados "a", "e", "t" e "x" acima, e o número
         "n" de setores de cada uma das duas linhas de montagem.
Saída: um número real (o menor tempo de montagem completa)
------------------------------------------------------------
1. M_{1 1} := e_1 , M_{2 1} := e_2
2. PARA j DE 2 A n
3. |  M_{1 j} := mín{ M_{1 j-1} + a_{1 j-1} ,
   |                 M_{2 j-1} + a_{2 j-1} + t_{2 j-1} }
4. |  M_{2 j} := mín{ M_{2 j-1} + a_{2 j-1} ,
   |                 M_{1 j-1} + a_{1 j-1} + t_{1 j-1} }
5. RETORNE mín{ M_{1 n} + a_{1 n} + x_1 , M_{2 n} + a_{2 n} + x_2 }.
============================================================
```

Claramente, o algoritmo acima executa em tempo O(n), que é o melhor possível, pois qualquer algoritmo que resolva corretamente o problema precisa ao menos ler os dados de entrada, os quais possuem tamanho θ(n).

9. EXERCÍCIO (VARIAÇÕES DO ALGORITMO ACIMA):

a) Como argumentado em sala, aplicar divisão-e-conquista de forma direta ao problema acima leva a um algoritmo que considera cada uma das 2^n maneiras de um carro passar pelos "n" setores de montagem. Escreva um algoritmo que implementa essa ideia, e se convença de que a complexidade do tempo de execução dele é muito maior que a do algoritmo acima.

b) O algoritmo acima utiliza uma matriz auxiliar M de 2 linhas e n colunas, ou seja, utiliza $\theta(n)$ de memória auxiliar. É possível diminuir esse uso de memória para O(1)?

c) Escreva uma variação do algoritmo acima que retorne, além do tempo de montagem mínimo, também um vetor l[1..n] que informa, para cada setor j, a linha de montagem l[j] cuja estação de montagem é utilizada no setor j da solução ótima (logo, para todo j, l[j] vale 1 ou 2).

d) Generalize o algoritmo acima para o caso de "m" linhas de montagem, ao invés de apenas duas.

e) Escreva uma versão memoizada do algoritmo acima.