1. Algoritmo de ordenação por dígitos ("radix sort"):

```
======================================================================
Algoritmo: ord_díg
Entrada: vetor A[1..n], cada chave sendo uma tupla de "d" "dígitos",
         cada dígito assumindo valores de 1 a "k".
Saída: nenhuma.
----------------------------------------------------------------------
1. PARA i de d a 1
2. | ordene A com relação ao dígito i usando ordenação por contagem.
======================================================================
```

2. COMPLEXIDADE: θ(d(n + k)), o que é θ(dn) se k = O(n).
   Observe que essa complexidade é melhor que a complexidade de θ(d*n*lg n) que obteríamos aplicando diretamente um algoritmo de ordenação baseado em comparações (que utilizasse uma função O(d) para comparar chaves).

3. EXERCÍCIO: reescreva o algoritmo acima, substituindo a linha 2 por um código equivalente ao da ordenação por contagem. O k-ésimo dígito de A[i] pode ser denotado por A[i].díg[k], ou A[i][k], ou como lhe convenha.

4. Como visto em sala, o algoritmo ordenação por dígitos também pode ser utilizado para ordenar um vetor de inteiros.
Para tanto, seja "b" o números de bits utilizado para se armazenar um inteiro na máquina em questão, e considere um natural 1 <= r <= b.
A ideia é interpretar cada inteiro como possuindo teto(b/r) dígitos, cada um assumindo valores de 0 a 2^r - 1 (que são os valores possíveis para um conjunto de "r" bits).
Em outras palavras, nós utilizaremos a base 2^r para obter os dígitos dos elementos do vetor em questão.
Como o algoritmo de ordenação por dígitos executa em tempo θ(d(n+k)), e como, no caso em questão, temos d = teto(b/r) e k = 2^r, então o algoritmo executará em tempo θ((b/r)(n + 2^r)).

5. LEMA: assintoticamente, o valor ótimo para r é mín {b, piso(lg n)}.

```
======================================================================
------------------------------ PROVA -------------------------------
```
Nós mostraremos o resultado por análise dos casos possíveis.

Suponha primeiramente que b < piso(lg n). Nesse caso, como r <= b, então n + 2^r = θ(n), independentemente do valor de r. Por outro lado, o termo b/r diminui com o aumento de r. Logo, o melhor valor para r é o maior possível, que é b = mín {b, piso(lg n)}, CQD.

Suponha agora que b >= piso(lg n). Nesse caso, observe primeiramente que, se r = piso(lg n), então o tempo de execução do algoritmo é θ((b/lg n)*n). Além disso, observe que, para valores maiores de r, o termo 2^r cresce mais rapidamente do que decresce o termo 1/r, levando a função a assumir valores maiores. Por fim, valores de r menores que piso(lg n) tornam o termo 1/r maior que 1/piso(lg n), mas não diminuem o termo "n + 2^r" assintoticamente, que continua valendo θ(n). Logo, o melhor valor para "r" é piso(lg n) = mín {b, piso(lg n)}, CQD.

```
------------------------------ PROVA -------------------------------
======================================================================
```

6. OBSERVAÇÃO: como visto acima, se b <= piso(lg n), então pode-se fazer r = b e o algoritmo ordenação por dígitos ordena um vetor de inteiros em tempo θ(n). Caso b > piso(lg n), então, fazendo-se r = piso(lg n), o algoritmo executa em tempo θ((b/piso(lg n))*n).

7. EXERCÍCIO: como observado em sala, é sempre possível utilizar a
   ordenação por dígitos para ordenar um vetor de inteiros segundo
uma base d >= 2 qualquer; na prática, entretanto, é conveniente
utilizar como base uma potência de 2, para aproveitar a representação
nativa dos inteiros nos computadores. MOSTRE ENTÃO que esse é, de
fato, o caso, implementando em C uma função que ordene um vetor de
inteiros pelo algoritmo de ordenação por dígitos. O valor de "r",
que determina a base 2^r utilizada, deve ser recebido como argumento.
Utilize deslocamento de bits para obter rapidamente os dígitos de cada
inteiro na base 2^r.

8. Algoritmo para o problema das Torres de Hanói:

```
============================================================
Algoritmo: hanoi
Entrada: naturais d (número de discos), i (pino inicial),
                  a (pino auxiliar),    f (pino final).
Saída: nenhuma.
------------------------------------------------------------
1. SE d = 1
2. | IMPRIMA "i -> f"
3. SENÃO
4. | hanoi(d-1,i,f,a)
5. | IMPRIMA "i -> f"
6. | hanoi(d-1,a,i,f)
============================================================
```

9. COMPLEXIDADE:

t(1) = a,   para algum a > 0.   (= θ(1))
t(d) = b + 2*t(d-1), para d > 1, para algum b > 0.

10. EXERCÍCIO: mostre que t = θ(2^d).

11. EXERCÍCIO: qual é o número exato de jogadas realizadas pela
    solução gerada pelo algoritmo acima para "d" discos? Esse número
    é ótimo?

12. EXERCÍCIO: escreva uma variação do algoritmo acima, na qual a
    resposta é dada pelo algoritmo não por meio da impressão na tela,
    mas sim do preenchimento de dois vetores "O" e "D", O[k] devendo
    armazenar o número do pino de origem da k-ésima jogada e D[k] o
    número do pino de destino da mesma jogada. Além disso, o algoritmo
    deve passar a retornar o número da última jogada (de forma que
    quem chamar o algoritmo saiba até que posição dos vetores "O" e
    "D" está registrada a sequência de jogadas). Os vetores "O" e "D"
    devem ser recebidos como entrada, e o algoritmo deve supor que
    eles são grandes o suficiente. Caso seja conveniente, você pode
    adicionar parâmetros ao algoritmo.