1. PROBLEMA DE SELEÇÃO: dado um vetor A[1..n] de números distintos,
   e dado um índice "i" de 1 a n, encontrar o i-ésimo menor elemento
   de "A".

2. OBSERVAÇÃO: maneiras intuitivas de resolver o problema:

a) Descobrir o menor elemento do vetor e colocá-lo no início;
   depois, percorrer o restante do vetor, descobrir o menor
   elemento desse trecho restante e colocá-lo na posição 2 do vetor;
   repetir o procedimento até se encontrar o i-ésimo menor elemento.
   Essa estratégia leva tempo θ(n^2) no pior caso.
   Como observado em sala, trata-se de uma variação da ordenação por
   seleção.

b) Ordenar o vetor e então selecionar o elemento que estiver na
   posição "i". Custa O(n*lg n), mesmo no pior caso: basta utilizar
   um algoritmo de ordenação que execute nesta complexidade de tempo,
   como a ordenação por entrelaçamento (merge sort) ou a ordenação
   por monte (heapsort).

c) Montar um monte ("heap") de mínimo e fazer "i" extrações do menor
   elemento -- ou, caso i > n/2, montar um monte de máximo e fazer
   n-i+1 extrações do máximo. Custa θ(n) + O(i*lg n), o que é melhor
   que o custo da estratégia "b" caso i seja bem menor que "n", ou
   caso "i" seja bem próximo de "n"; entretanto, caso "i" seja "n/2",
   ainda custa θ(n*lg n) no pior caso.

3. Algoritmo de seleção por particionamento:

```
============================================================
Algoritmo: seleção
Entrada: vetor A, índices "a", "i", "b" // a <= i <= b
Saída: um elemento do vetor
------------------------------------------------------------
1. SE a = b
2. | RETORNE A[a] // = A[b] = A[i]
3. k := particionamento(A,a,i,b) /* no lugar de i, qualquer
                outro valor de "a" a "b" também serviria */
4. SE k = i
5. | RETORNE A[k] // = A[i]
6. SE k > i
7. | RETORNE seleção(A, a, i, k-1)
8. RETORNE seleção(A, k+1, i-k, b)
============================================================
```

4. Tempo de execução do algoritmo, NO PIOR CASO, sendo n = b -a +1:

T(n) = θ(1), se n = 1
T(n) = θ(n) + máx{T(q) : para todo q < n}, se n > 1

Não é difícil observar que T é uma função crescente.
Nesse caso, podemos simplificar a 2a equação, obtendo:

T(n) = θ(n) + T(n-1)

5. EXERCÍCIO: mostre que a função T acima é θ(n^2).

6. OBSERVAÇÃO: apesar do resultado anterior, o algoritmo acima tem bom
   desempenho médio na prática, pelo mesmo motivo que o quicksort: uma
   execução na qual os particionamentos são repetidamente ruins é
   muito improvável no caso de entradas "aleatórias".

7. EXERCÍCIO: qual é o tempo de execução do algoritmo acima no MELHOR
   caso?

8. Algoritmo de seleção por particionamento, com garantia de qualidade do particionamento: dados "A" e "i", realize os seguintes passos:

* Passo 1: divida "A" em piso(n/5) grupos de 5 elementos contíguos e, se n mod 5 != 0, um grupo com os n mod 5 últimos elementos do vetor.

* Passo 2: encontre a mediana de cada um dos grupos, ordenando cada grupo e então selecionando o elemento "do meio".

* Passo 3: utilize o presente algoritmo recursivamente para encontrar a mediana "x" das teto(n/5) medianas obtidas no passo anterior.

* Passo 4: particione "A" com relação a "x", ficando então este elemento numa certa posição "k" do vetor particionado.

* Passo 5: se i = k, retorne k. Se i < k, chame o presente algoritmo recursivamente para encontrar o i-ésimo menor do trecho A[1..k-1]. Se i > k, encontre, recursivamente, o (i-k)-ésimo menor elemento do trecho A[k+1..n].

9. Tempo de execução do algoritmo acima:

a) Observe que os passos 1, 2 e 4 custam, ao todo, $\theta(n)$.

b) Como discutido em sala, das ~ n/5 medianas obtidas no passo 2, ~ n/10 são menores que x, e, no grupo de cada tal mediana y, há 2 elementos menores que y, os quais são portanto também menores que x. Logo, há pelo menos ~ 3n/10 elementos de A que são menores que x. Além disso, por argumento análogo, há pelo menos ~ 3n/10 elementos de "A" que são maiores que x.
De forma precisa, ambas as quantidades são >= 3n/10 - 6, pois:

   1) São teto(n/5) grupos, e portanto teto(n/5) medianas.

   2) 2 grupos não contam: o de "x" e o último.
      Logo, os grupos restantes são teto(n/5) - 2.

   3) Dos grupos restantes, certamente pelo menos o piso da metade está à esquerda de "x", e a mesma quantidade à direita de x. Logo, há pelo menos piso((teto(n/5) - 2) / 2) medianas menores que "x", e o mesmo sobre as maiores.

   4) Logo, a quantidade de elementos garantidamente menores/maiores que "x" é:

$$\begin{aligned} 3 * \text{piso}((\text{teto}(n/5) - 2) / 2) &>= 3 * ((\text{teto}(n/5) - 2)/2 - 1) \\ &= 3/2 * (\text{teto}(n/5) - 4) \\ &>= 3/2 * (n/5 - 4) \\ &= 3n/10 - 6. \end{aligned}$$

c) Logo, a chamada recursiva do passo 5 é feita para no máximo n - (3n/10 - 6) = 7n/10 + 6 elementos.

d) O seguinte vale então sobre o tempo de execução do algoritmo:

$$\begin{aligned} T(n) &= O(1), && \text{se } 1 <= n < 140 \\ T(n) &<= T(\text{teto}(n/5)) + T(7n/10 + 6) + O(n), && \text{se } n >= 140 \end{aligned}$$

A constante 140 acima é, como já nos é familiar, escolhida em função do passo da indução da solução da recorrência.

10. EXERCÍCIO: mostre que T(n) = O(n).

11. EXERCÍCIO: implemente o algoritmo de seleção O(n) acima em C.