```php
<?
    // 1. Considerando o algoritmo de ordenação por inserção:

    // a) Na função insertionSort, troque a comparação A[i]>x por A[i]>=x. A nova função continua produzindo uma ordenação crescente de v[0..n?1]?
        function insertionSort1 ( $A, $n )
        {
            for ( $j = 1; $j < $n; $j ++ )
            {
                $x = $A [ $j ];
                $i = $j - 1;

                while ( ( $i >= 0 ) and ( $A [ $i ] >= $x ) )
                {
                    $A [ $i + 1 ] = $A [ $i ];
                    $i = $i - 1;
                }

                $A [ $i + 1 ] = $x;
            }
        }
    //  R: sim, continua produzindo uma ordenação crescente.

    // b) O que acontece se trocarmos "for (j=1..." por "for (j=0..." no código da função insertionSort?
        function insertionSort2 ( $A, $n )
        {
            for ( $j = 0; $j < $n; $j ++ )
            {
                $x = $A [ $j ];
                $i = $j - 1;

                while ( ( $i >= 0 ) and ( $A [ $i ] > $x ) )
                {
                    $A [ $i + 1 ] = $A [ $i ];
                    $i = $i - 1;
                }

                $A [ $i + 1 ] = $x;
            }
        }
    //  R: ordenará a partir do primeiro elemento do vetor, e não do segundo.

    // c) Que acontece se trocarmos A[i+1] = x por A[i] = x no código da função insertionSort?
        function insertionSort3 ( $A, $n )
        {
            for ( $j = 1; $j < $n; $j ++ )
            {
                $x = $A [ $j ];
                $i = $j - 1;

                while ( ( $i >= 0 ) and ( $A [ $i ] > $x ) )
                {
                    $A [ $i + 1 ] = $A [ $i ];
                    $i = $i - 1;
                }

                $A [ $i ] = $x;
            }
        }
    //  R: última posição fica -1, eliminando o último elemento.

    // d) Altere o algoritmo de ordenação por inserção para permutar os elementos de um vetor inteiro A[0..n?1] de modo que eles fiquem em ordem decrescente.
        function insertionSort4 ( $A, $n )
        {
            for ( $j = 1; $j < $n; $j ++ )
            {
                $x = $A [ $j ];
                $i = $j - 1;

```

```
                while ( ( $i >= 0 ) and ( $A [ $i ] < $x ) ) // <- troca aqui o sinal
                {
                    $A [ $i + 1 ] = $A [ $i ];
                    $i = $i - 1;
                }

                $A [ $i + 1 ] = $x;
            }
        }

    // e) MOVIMENTAÇÃO DE DADOS: Quantas vezes, no pior caso, o algoritmo de
    inserção copia um elemento do vetor de um lugar para outro? Quantas vezes isso
    ocorre no melhor caso?
    // R: O pior caso é n-1. No melhor caso, 0.

    // 2. Considerando o algoritmo de ordenação por seleção:
    // a1) Na função selecao, o que acontece se trocarmos "for (i=0" para "for (i=1"?
        function selectionSort1 ( $n, $v )
        {
            for ( $i = 1; $i < ( $n - 1 ); $i ++ )
            {
                $min = $i;

                for ( $j = ( $i + 1 ); $j < $n; $j ++ )
                {
                    if ( $v [ $j ] < $v [ $min ] )
                        $min = $j;
                }

                $x = $v [ $i ];
                $v [ $i ] = $v [ $min ];
                $v [ $min ] = $x;
            }
        }
    // R: ordena apenas a partir da segunda posição do vetor.

    // a2) O que acontece se trocarmos "for (i=0; i<n-1" por "for (i=0; i<n"?
        function selectionSort2 ( $n, $v )
        {
            for ( $i = 0; $i < $n; $i ++ )
            {
                $min = $i;

                for ( $j = ( $i + 1 ); $j < $n; $j ++ )
                {
                    if ( $v [ $j ] < $v [ $min ] )
                        $min = $j;
                }

                $x = $v [ $i ];
                $v [ $i ] = $v [ $min ];
                $v [ $min ] = $x;
            }
        }
    // R: ordena até a penúltima posição do vetor.

    // b) Na função selecao, troque a comparação "v[j] < v[min]" por "v[j] <=
    v[min]". A nova função continua correta?
        function selectionSort3 ( $n, $v )
        {
            for ( $i = 0; $i < ( $n - 1 ); $i ++ )
            {
                $min = $i;

                for ( $j = ( $i + 1 ); $j < $n; $j ++ )
                {
                    if ( $v [ $j ] <= $v [ $min ] )
                        $min = $j;
                }

                $x = $v [ $i ];
                $v [ $i ] = $v [ $min ];
```

```
138                 $v [ $min ] = $x;
139             }
140         }
141     // R: sim (mudando apenas a "estabilidade").
142
143     // c) MOVIMENTAÇÃO DE DADOS: Quantas vezes, no pior caso, o algoritmo de seleção
        copia um elemento do vetor de um lugar para outro? Quantas vezes isso ocorre no
        melhor caso?
144     // R: no pior caso, (n - 1). No melhor caso, 1.
145
146     // d) Escreva uma função que permute os elementos de um vetor inteiro v[0..n-1]
        de modo que eles fiquem em ordem decrescente. Inspire-se no algoritmo de seleção.
147         function selectionSort4 ( $n, $v )
148         {
149             for ( $i = 0; $i < ( $n - 1 ); $i ++ )
150             {
151                 $min = $i;
152
153                 for ( $j = ( $i + 1 ); $j < $n; $j ++ )
154                 {
155                     if ( $v [ $j ] > $v [ $min ] ) // <- trocar aqui
156                         $min = $j;
157                 }
158
159                 $x = $v [ $i ];
160                 $v [ $i ] = $v [ $min ];
161                 $v [ $min ] = $x;
162             }
163         }
164
165     // 3. Considerando o algoritmo de ordenação por flutuação:
166
167     // a) MOVIMENTAÇÃO DE DADOS: Quantas vezes, no pior caso, o algoritmo de
        flutuação copia um elemento do vetor de um lugar para outro? Quantas vezes isso
        ocorre no melhor caso?
168     // R: no pior caso, (n - 1). No melhor, 0.
169
170     // b) Escreva uma função que permute os elementos de um vetor inteiro v[0..n-1]
        de modo que eles fiquem em ordem decrescente. Inspire-se no algoritmo de
        flutuação.
171         function bubble ( $v, $n )
172         {
173             $k = ( $n - 1 );
174
175             for ( $i = 0; $i < $n; $i ++ )
176             {
177                 for ( $j = 0; $j < $k; $j ++ )
178                 {
179                     if ( $v [ $j ] < $v [ $j + 1 ] ) // <- só trocar aqui pra ficar
                        um "bubble decrescente"...
180                     {
181                         $aux = $v [ $j ];
182                         $v [ $j ] = $v [ $j + 1 ];
183                         $v [ $j + 1 ] = $aux;
184                     }
185                 }
186
187                 $k --;
188             }
189         }
190
191     // Faça a análise do tempo de execução para o pior e melhor caso da seguinte
        implementação do bubblesort:
192     void bubbleSort ( int arr[], int n )
193     {
194         int i, j;
195         int swapped;
196
197         for(i=0; i<n-1; i++)
198         {
199             swapped=0;
200
```

```
201             for(j =0; j<n-i-1; j++)
202             {
203                 if(arr[j]>arr[j +1])
204                 {
205                     swap(&arr[j], &arr[j+1]);
206                     swapped=1;
207                 }
208             }
209 
210             if(swapped==0)
211                 break;
212         }
213     }
214     // R: o pior caso vai percorrer todos os elementos apenas uma vez. No melhor
        caso, ele percorre o vetor por duas vezes.
215 
216     // 4. O algoritmo de seleção é estável? Não.
217     // 5. O algoritmo de inserção é estável? Sim.
218     // 6. O algoritmo de flutuação é estável? Sim.
219 
220     // Obs.: dos algoritmos que a gente estudou até agora, mesmo os que não são
        estáveis, qualquer um deles pode "ser tornado" estável com uma comparação extra
        ali no meio, pra não perder o vínculo das chaves.
221 ?>
```