```php
<?

    // include para poder medir o tempo de execução
    include ( "exec_time.php" );

    // 1. Implemente o quicksort em qualquer linguagem de programação.
    function quicksort ( &$vetor )
    {
        // cria os "subarrays" à esquerda e à direita do pivô
        $sub_esquerda = array ();
        $sub_direita = array ();

        // mata se chegar no caso base
        if ( count ( $vetor ) < 2 )
            return ( $vetor );

        // pega o índice do atual (por causa da recursividade...) e tira o primeiro elemento
        $indice_pivo = key ( $vetor );
        $pivo = array_shift ( $vetor );

        // "formata" o pivô como se fosse um array, pra poder "fundir" ele com os menores (esq) e com os maiores (dir)
        $pivo_formatado = array (
            $indice_pivo => $pivo
        );

        // varre o vetor atual, dividindo entre esquerda e direita a partir do pivô
        foreach ( $vetor as $elemento )
        {
            // se for menor, esquerda... senão, direita
            if ( $elemento < $pivo )
                $sub_esquerda [] = $elemento;
            elseif ( $elemento > $pivo )
                $sub_direita [] = $elemento;
        }

        // "funde" os 3 vetores usando recursividade: esquerda, pivô (como se fosse um array) e direita.
        $organizado = array_merge ( quicksort ( $sub_esquerda ), $pivo_formatado, quicksort ( $sub_direita ) );

        return ( $organizado );
    }

    // a) Faça a análise do tempo de execução do pior e melhor caso.
        startExec ();
        echo "Organizando... " . quicksort ( $VETOR ) . "<br>";
        echo "O tempo de execução foi de " . endExec () . "<br>";
    // R: o pior caso é quando o pivô (que divide os subs da esq e dir) é o maior ou o menor elemento do vetor. Já o melhor caso é quando o pivô divide os subs (esq e dir) exatamente no meio.

    // 2. Reescreva a função partition fazendo com que o quicksort usando outro critério para a escolha do pivô.
        int partition ( int *A, int p, int r )
        {
            int pivot = A [ p ]; // <- contanto que p < r, apenas substituí o "r" por "p".
            int i = ( p - 1 ), temp;

            for ( int j = p; j <= r-1; j ++ )
            {
                if ( A [ j ] <= pivot )
                {
                    i ++;
                    temp = A [ i ];
                    A [ i ] = A [ j ];
                    A [ j ] = temp;
                }
            }

```

```
65                temp = A [ i + 1 ];
66                A [ i + 1 ] = A [ r ];
67                A [ r ] = temp;
68
69                return ( i + 1 );
70            }
71    ?>
```