Teste de Lógica de Programação – Hexacta – Remoto

Instruções: Este teste remoto consiste de quatro problemas para serem codificados em Java. Todos são testes de lógica, não demandando conhecimento aprofundado da linguagem. Não é preciso ter pressa para entregar as soluções, faça no seu tempo. Tente codificar as soluções da forma mais clara e simples possível e teste bem o seu código antes de fazer a entrega. Caso haja alguma dúvida em relação aos enunciados, por favor mande suas dúvidas por escrito que responderemos o mais breve possível. As respostas poderão ser enviadas por e-mail ou disponibilizadas para consulta no seu GitHub pessoal.

Boa sorte!

Questão 1) Complete o código Java abaixo:

```
// você pode usar imports, por exemplo:
// import java.util.*;

// você pode escrever em stdout para fins de debug, ex.
// System.out.println("esta e uma mensagem de debug");

public class Solution1 {
    public int solution(int[] A) {
        // escreva o seu codigo em Java
    }
}
```

de forma que, ao receber um array A com N inteiros, o método `public int solution(int[] A)` retorne o menor número inteiro positivo (maior que 0) que não esteja presente em A. Por exemplo,

- dado A = [1, 3, 6, 4, 1, 2], a função deve retornar 5;
- dado A = [1, 2, 3], a função deve retornar 4;
- dado A = [-1, -3], a função deve retornar 1.

Escreva um algoritmo eficiente para o problema descrito acima respeitando as seguintes premissas:

- N é um inteiro dentro do intervalo [1, 100.000];
- cada elemento do array A é um inteiro dentro do intervalo [-1.000.000, 1.000.000].

Questão 2) Escreva um método Java que receba um array de números ordenados de forma ascendente e uma variável numérica x e calcule, da forma mais eficiente possível, todas as combinações de pares de números deste array cuja a soma é igual a x.

Exemplo: Dado um array = [-2, -1, 0, 2, 4, 7, 8, 9, 9] e x = 8 como entrada, a resposta do seu algoritmo deverá ser [[-1, 9], [0, 8], [4, 4]] (não precisando ser necessariamente nesta ordem)

Questão 3) Complete o código Java abaixo:

```
// você pode usar imports, por exemplo:
// import java.util.*;

// você pode escrever em stdout para fins de debug, ex.
// System.out.println("esta e uma mensagem de debug");

public class Solution3 {
    public double calcPM(int m, double p) {
        // escreva o seu codigo em Java
    }
}
```

de forma que o método `public double calcPM(int m, double p)`, ao receber um inteiro m (sendo 1 <= m <= 10^9) e um valor double p, retorne o valor de p elevado a m (ou p^m) sem utilizar bibliotecas matemáticas ou a operação de exponenciação nativa do Java e que seja computacionalmente eficiente. Por exemplo,

- calcPM(m = 1000, p = 0.999999999999D): 0,99999999900
- calcPM(m = 1000000, p = 0.999999999999D): 0,99999900002
- calcPM(m = 1000000000, p = 0.999999999999D): 0,99900052193

Questão 4) Reescreva o trecho de código destacado em amarelo abaixo para fazer com que o método `public void printIntersection(int[] setA, int[] setB)` tenha um processamento mais eficiente:

```
import java.util.ArrayList;
import java.util.Arrays;

public class Solution4 {

    public void printIntersection(int[] setA, int[] setB) {
      if (setA != null && setB != null) {
            Arrays.sort(setA);
            Arrays.sort(setB);
            ArrayList<Integer> commonSet = new ArrayList<Integer>();

            for (int i = 0; i < setA.length; i++) {
                  for (int j = 0; j < setB.length; j++) {
                        if (setA[i] == setB[j] && !commonSet.contains(setA[i])) {
                              commonSet.add(setA[i]);
                        }
                  }
            }

            for (int k = 0; k < commonSet.size(); k++) {
                  System.out.print(commonSet.get(k) + " ");
            }
      }
    }
}
```