**DESAFIO LÓGICA**

**As questões abaixo podem ser solucionadas na linguagem de sua preferência. Tente**

**resolver o maior numero de questões que conseguir.**

**01 - QUESTÃO**

Dado um array de números inteiros, retorne os índices dos dois números de forma que eles se

somem a um alvo específico.

Você pode assumir que cada entrada teria exatamente uma solução, e você **não** pode usar o

mesmo elemento duas vezes.

**Exemplo:**

Dado nums = [2, 7, 11, 15], alvo = 9,

Como nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

List<Integer> nums = Arrays.asList(1,2,3,4,5,6,7,8,9,0);

Integer alvo=9;

for (Integer num : nums) {

for (Integer numi : nums) {

if((num+numi)==alvo) {

System.out.println(num);

System.out.println(numi);

System.exit(0);

}

}

}

**02 – QUESTÃO**

Um bracket é considerado qualquer um dos seguintes caracteres: (, ), {, }, [ ou ].

Dois brackets são considerados um par combinado se o bracket de abertura (isto é, (, [ou {) ocorre à esquerda de um

bracket de fechamento (ou seja,),] ou} do mesmo tipo exato. Existem três tipos de pares de brackets : [], {} e ().

Um par de brackets correspondente não é balanceado se o de abertura e o de fechamento não corresponderem entre

si. Por exemplo, {[(])} não é balanceado porque o conteúdo entre {e} não é balanceado. O primeiro bracket inclui o

de abertura, (, e o segundo inclui um bracket de fechamento desbalanceado,].

Dado sequencias de caracteres, determine se cada sequência de brackets é balanceada. Se uma string estiver

balanceada, retorne SIM. Caso contrário, retorne NAO.

**Exemplo:**

{[()]} SIM

{[(])} NAO

{{[[(())]]}} SIM

String basicBrackets = "{{[[(())]]}}";

List<String> brackets = new ArrayList<String>();

for (int i = 0; i < basicBrackets.length(); i++) {

brackets.add(basicBrackets.substring(i, i+1));

}

for (int i = 0; i < (brackets.size()/2); i++)

String bracket = brackets.get(i) + brackets.get((brackets.size()-1)-i) ;

if( !( "{}".equals(bracket) || "[]".equals(bracket) || "()".equals(bracket) ) ) {

System.out.println(false);

System.exit(0);

}

}

System.out.println(true);

**03 – QUESTÃO**

Digamos que você tenha um array para o qual o elemento i é o preço de uma determinada ação

no dia i.

Se você tivesse permissão para concluir no máximo uma transação (ou seja, comprar uma e

vender uma ação), crie um algoritmo para encontrar o lucro máximo.

**Note que você não pode vender uma ação antes de comprar**.

**Exemplo:**

**Input:** [7,1,5,3,6,4]

**Output:** 5 (Comprou no dia 2 (preço igual a 1) e vendeu no dia 5 (preço igual

a 6), lucro foi de 6 – 1 = 5

**Input:** [7,6,4,3,1]

**Output:** 0 (Nesse caso nenhuma transação deve ser feita, lucro máximo igual a

0)

var input = new List<int>();

int diaCompra = 0;

int diaVenda = 0;

int compra = 0;

int venda = 0;

WriteLine("Digite o tamanho do array: ");

var size = int.Parse(ReadLine());

WriteLine("Digite os valores do array: ");

for (int i = 0; i < size; i++)

{

input.Add(int.Parse(ReadLine()));

}

for (int i = 0; i < input.Count; i++)

{

if (input[i] == input.Min())

{

diaCompra = i + 1;

compra = input[i];

i = diaCompra;

}

else if (i + 1 < size)

{

if (input[i] > input[i + 1])

{

if (compra > 0)

{

venda = input[i];

diaVenda = i + 1;

}

}

}

}

if (venda - compra > 0)

Write($"{venda - compra}");

else

Write($"0");

**04 – QUESTÃO**

Dados n inteiros não negativos representando um mapa de elevação onde a largura de cada barra é 1, calcule quanta água é capaz de reter após a chuva.



**Exemplo:**

**Input:** [0,1,0,2,1,0,1,3,2,1,2,1]

**Output:** 6

List<Integer> nums = Arrays.asList(0,1,0,2,1,0,1,3,2,1,2,1);

Integer sum = 0;

Integer dezena = 10;

for (Integer num : nums) {

sum+=num;

}

while(dezena<sum) {

dezena+=10;

}

System.out.println(dezena-sum);