THE EVIL TRADER

You are a genius working for one of the most famous high-frequency trading firms in the world. But you are an evil genius and have found a backdoor that lets you add fake operations in the last second with all the daily stock information available. Having all the information, you can choose when to buy some shares and when to sell them and get maximum profit. Of course, you don't want to be greedy and draw attention to the operations and get caught, so you limit them to one buy and sell per day. Since it's a high-frequency trading system and you expect a big input, you design a program that reads all the stock information with 100ms resolution, gives the time when you want to add the buy and sell operations (in milliseconds) and tells you the expected gain. Of course, you want the maximum gain for that transaction.

Input

One line with an integer for each stock sample.

Output

The starting (when you buy) and ending (when you sell) point of the transaction in milliseconds and the expected gain.

Sample input

```
70
93
69
75
52
73
67
81
65
48
67
```

Sample output

```
400 700 29
```

El ejercicio en sí ha sido planteado para recibir cualquier tipo de cadena de entrada (Que era lo que se planteaba). Así mismo, se intenta resolver el problema de no tener que probar todos los números con todos.

La base de todo reside en la creación de 2 estructuras, una cola y un vector, ambas dinámicas.

La idea es la siguiente:

1. Creamos nuestra estructura en la clase Stock con parámetros número de tipo Integer que contiene el valor de nuestra entra y un segundo parámetro auxiliar, posición, que determina **la posición del elemento en la cola**

```
public class Stock {
    int numero;
    BigInteger posicion;
    public Stock(int n, BigInteger p){
        numero = n;
        posicion = p;
    }
}
```

2. Insertamos los elementos tanto en la cola como en el vector según entran por nuestro input. De esto se encarga nuestra función “insert”, que recibe un elemento de tipo stock.

```
ObjetoEvil.insert(new Stock(Integer.parseInt(line), interacciones));
```

```
public void insert(Stock elemento){
    cola.enqueue(elemento);
    vector.add(elemento);
}
```

3. Cuando hemos insertado todos los registros, ordenamos el vector con el método de la burbuja

```
ObjetoEvil.ordenar();
```

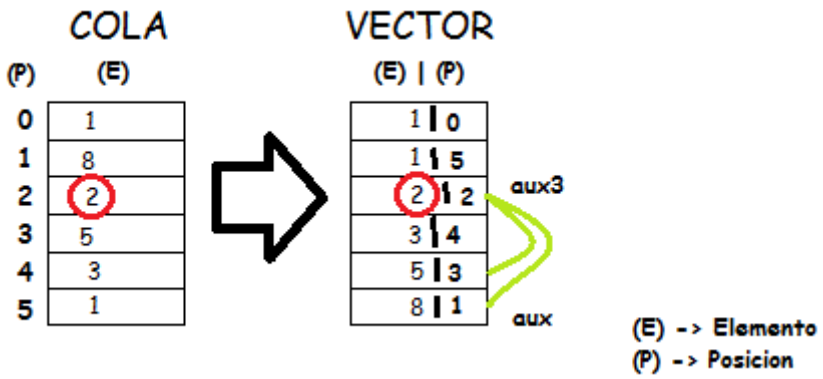
4. Una vez ordenado ya disponemos de las herramientas necesarias. Ahora llamamos al método search e inicializamos todas las variables.
    a. Inicializamos la variable **aux1** encargargada de contar guardar el elemento de compra y **aux2**, que se encargará de guardar el elemento de venta por defecto, el primer elemento de la cola y el último elemento de la lista que es el mayor.

```
public void search(){
    aux1 = cola.front();
    aux2 = vector.elementAt(vector.size()-1);
```

    b. Por cada elemento de la cola, sabemos en qué posición se encuentra en ese vector ordenado, luego los elementos que queden por encima ya

quedan descartados y de los que hay por debajo, empezamos a mirar a través de su elemento si está por debajo de la cola gracias al atributo posición del objeto Stock. Por último, si resulta que el elemento es mayor que el elemento obtenido de la cola y también está por debajo, evaluamos si la ganancia es mayor que la que ya tenemos en la variable gain, si es mayor, se cambia y se cambia la variable option a 1, que quiere decir que ya tenemos ganancia superior, luego no interesa seguir buscando más, pasamos al siguiente elemento.

Gráficamente la taréa es la siguiente



Hemos obtenido el valor del elemento de la cola E = 2 y nos vamos al vector. Como sabemos en qué posición del vector está, automáticamente los valores que quedan por encima quedan descartados por ser menores. Ahora miramos desde la posición de aux y evaluamos. En el caso de la comparación aux3 con aux, vemos que efectivamente 8 > 2, pero resulta que en la cola real aux = 1 y aux3 = 2; si restamos aux – aux3 sale negativo, luego está por debajo. Aux pues pasaría a valer 5, el siguiente elemento empezando por el final. Como aux > aux3 y (P)aux3 < (P)aux entonces evalúa la ganancia. Si es mayor que la existente, se pasa al siguiente elemento de la cola y se inicia la misma operación.

5. Imprimimos el resultado

```
/*
 * SOLUCION
 */
public void result(){
    System.out.println(aux1.posicion.multiply(BigInteger.valueOf(100)) + " " + aux2.posicion.multiply(BigIn
}
```