Le code ci-dessous calcul la distance de Levenstein entre deux chaînes de caractères - qui sont lues dans deux fichiers passés en paramètre. Une étape du calcul est faite sur le GPU via l'utilisation du framework JCuda. Le kernel (méthode s'effectuant sur le GPU) est dans un fichier à part.
Après avoir pris connaissance du code, répondez aux 3 questions ci-dessous en cochant les cases.
***Note** : il n'est pas nécessaire de comprendre l'algorithme utilisé, mais plutôt l'utilisation du GPU.*

1. Quel est votre niveau d'expérience en programmation ?

Très grande expérience en programmation	1	2	3	4	5	Aucune expérience en programmation

2. Quel est notre niveau d'expérience en programmation GPU ?

Très grande expérience en programmation GPU	1	2	3	4	5	Aucune expérience en programmation GPU

3. Comment jugez-vous la difficulté à comprendre l'utilisation du GPU dans le code ?

Très difficile à comprendre	1	2	3	4	5	Très facile à comprendre

```
 1 import java.io.FileReader;
 2 import java.io.BufferedReader;
 3 import java.io.IOException;
 4 import jcuda.*;
 5 import jcuda.runtime.*;
 6 import jcuda.driver.*;
 7 import static jcuda.driver.JCudaDriver.*;
 8
 9 public class GPLevenshtein {
10
11   public static void main(String [] args) {
12
13     if(args.length < 2) {
14       System.err.println("Needs two arguments: <file a> <file b>");
15       System.exit(0);
16     }
17     String filea = args[0];
18     String fileb = args[1];
19     char[] a = null;
20     char[] b = null;
21
22     try {
23       FileReader fra = new FileReader(filea);
24       BufferedReader bra = new BufferedReader(fra);
25       a = bra.readLine().toCharArray();
26       FileReader frb = new FileReader(fileb);
27       BufferedReader brb = new BufferedReader(frb);
28       b = brb.readLine().toCharArray();
29     } catch (IOException e) {
30       e.printStackTrace();
31       System.exit(0);
32     }
33
34     // Initialize the costs array
35     int[] costs = new int[a.length+1];
36
37     // Enable exception
38     JCudaDriver.setExceptionsEnabled(true);
39
40     // Initialize the driver and create a context for the first device
41     cuInit(0);
42     CUdevice device = new CUdevice();
43     cuDeviceGet(device, 0);
44     CUcontext context = new CUcontext();
45     cuCtxCreate(context, 0, device);
46
```

```
47      // Load the CUDA kernel ptx file
48      CUmodule module = new CUmodule();
49      cuModuleLoad(module, "../JCudaLevenstein.ptx");
50
51      // Obtain a function pointer to the kernel function
52      CUfunction function = new CUfunction();
53      cuModuleGetFunction(function, module, "leven");
54
55      // Allocate the device input data, and copy the host input data
56      // to the device
57      int ptrSize = a.length * Sizeof.CHAR;
58      CUdeviceptr deviceInputA = new CUdeviceptr();
59      cuMemAlloc(deviceInputA, ptrSize);
60      cuMemcpyHtoD(deviceInputA, Pointer.to(a), ptrSize);
61
62      CUdeviceptr deviceInputB = new CUdeviceptr();
63      cuMemAlloc(deviceInputB, ptrSize);
64      cuMemcpyHtoD(deviceInputB, Pointer.to(b), ptrSize);
65
66      CUdeviceptr deviceInputCosts = new CUdeviceptr();
67      cuMemAlloc(deviceInputCosts, a.length * Sizeof.INT + 1);
68      cuMemcpyHtoD(deviceInputCosts, Pointer.to(costs), a.length * Sizeof.INT + 1);
69
70      // Set up the kernel parameters: A pointer to an array
71      // of pointers which points to the actual values
72      Pointer kernelParameters = Pointer.to(
73        Pointer.to(deviceInputA),
74        Pointer.to(deviceInputB),
75        Pointer.to(deviceInputCosts),
76        Pointer.to(new int[]{a.length})
77      );
78
79      // Kernel call parameters
80      int blockSizeX = 256;
81      int gridSizeX = (int)Math.ceil((double)a.length / blockSizeX);
82
83      // --------------------------------
84      long startTime = System.currentTimeMillis();
85      // --- Start of benchmark zone --->
86      for(int i = 0; i < costs.length; i++) costs[i] = i;
87      cuLaunchKernel(function,
88        gridSizeX, 1, 1,       // Grid dimension
89        blockSizeX, 1, 1,      // Block dimension
90        0, null,               // Shared memory size and stream
91        kernelParameters, null // Kernel and extra parameters
92      );
93      cuCtxSynchronize();
94      // <--- End of benchmark zone -----
95      long stopTime = System.currentTimeMillis();
96      // --------------------------------
97
98      // Benchmark time elapsed computation
99      long elapsedTime = stopTime - startTime;
100
101     // Output
102     System.out.println(elapsedTime/1000);
103 }
104
```

Fichier contenant le kernel (méthode s'effectuant sur le GPU).

```
extern "C"

__global__ void leven(char* a, char* b, char* costs, int size) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if(i > 0 && i < size) {

    costs[0] = i;
    int nw = i - 1;
    for(int j = 1; j <= size; j++) {
      int firstMin = costs[j] < costs[j-1] ? costs[j] : costs[j-1];
      // This line is hard to read due to the lack of min() function
      int secondMin = 1 + firstMin < a[i - 1] == b[j - 1] ?
        nw : nw + 1 ? 1 + firstMin : a[i - 1] == b[j - 1] ? nw : nw + 1;
      int cj = secondMin;
      nw = costs[j];
      costs[j] = cj;
    }
  }

}
```