INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

REDES DE COMPUTADORAS

PROF. AXEL ERNESTO MORENO CERVANTES

GRUPO: 3CM7

PRÁCTICA 5 – BUCKET SORT

ALEJANDRO DE JESÚS ZEPEDA FLORES

11 de junio de 2020

Prácticas: https://drive.google.com/drive/folders/12lUUJImgU72oXJfukVcbrB8OjSsMSkb2?usp=sharing

Práctica 5: https://drive.google.com/drive/folders/1MtQCZDRdKCxPQn-J4oVRMxzYBl9cR62y

OBJETIVOS

Implementar el algoritmo de ordenamiento bucket sort en un entorno distribuido haciendo uso de hilos, así como sockets de flujo. Desarrolla aplicaciones en red, con base en el modelo cliente-servidor y utilizando de sockets de flujo, así como hilos para el envío de datos a cada uno de los servidores.

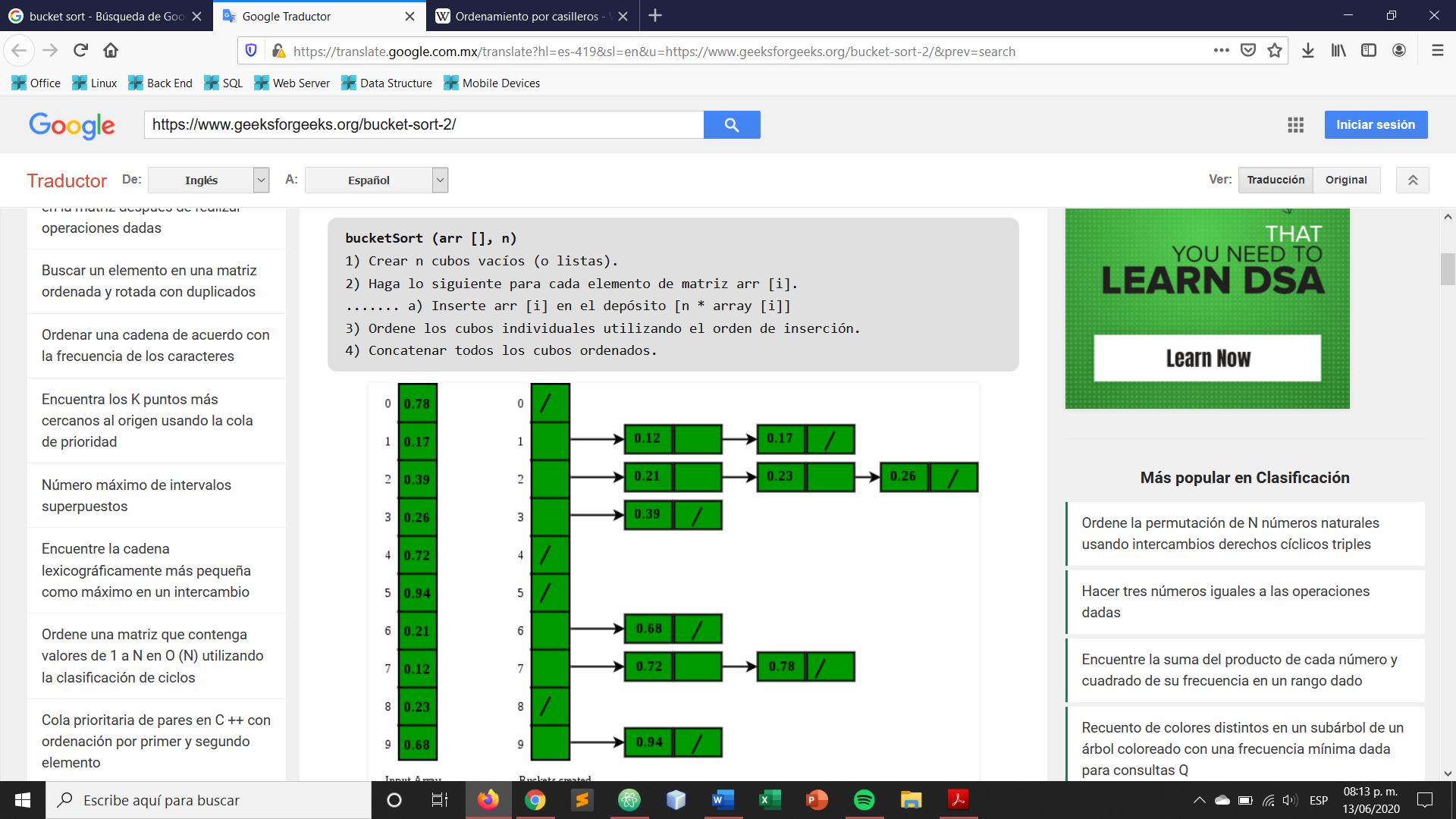
INTRODUCCIÓN

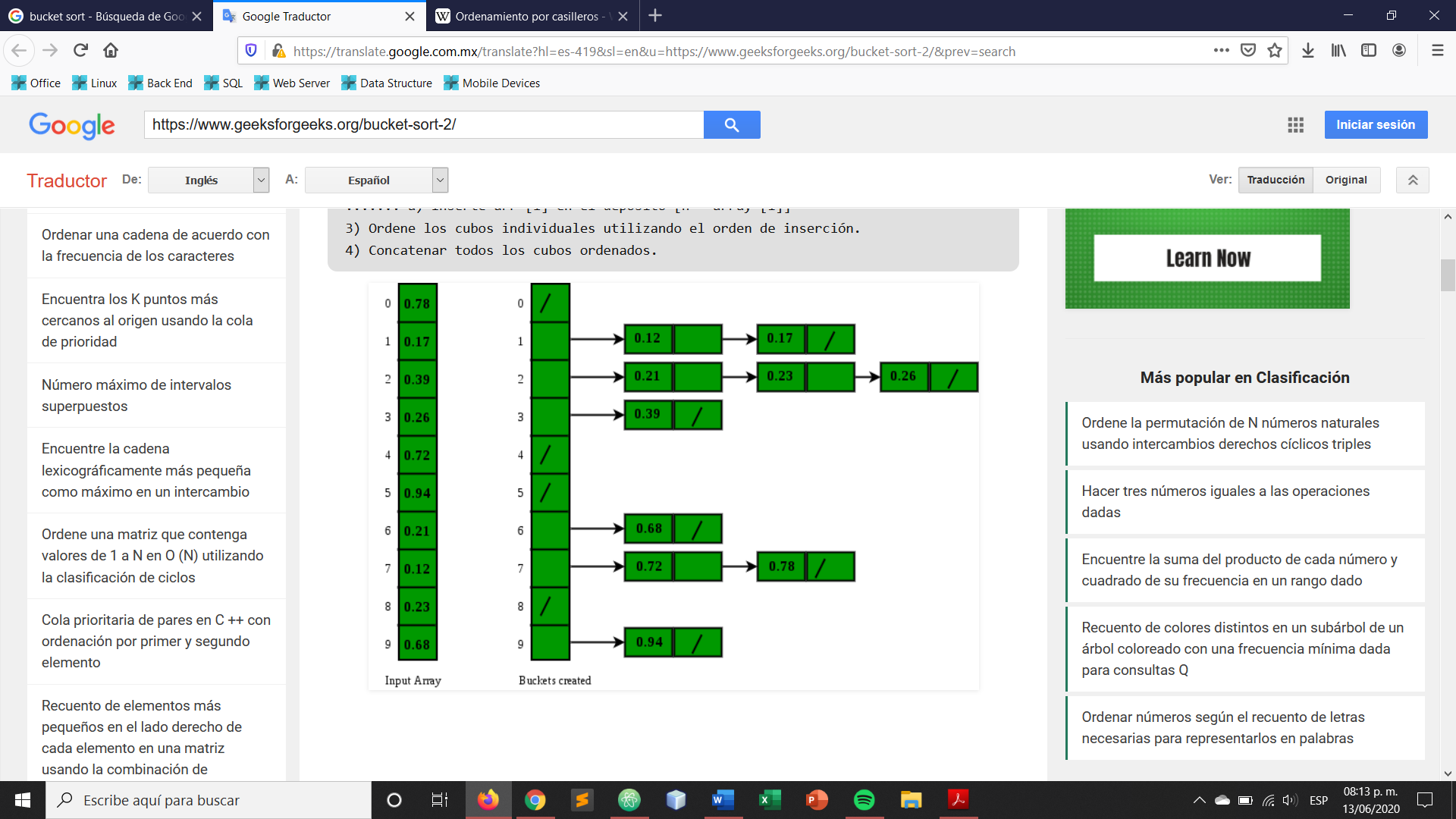
Es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número finito de casilleros o cubetas, cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones. Este método es eficiente y rápido, en cuanto a ordenamiento interno se refiere, es decir, se lleva a cabo completamente en la memoria principal.

La clasificación de cubetas es principalmente útil cuando la entrada se distribuye uniformemente en un rango. Por ejemplo, considere el siguiente problema. *Ordene un conjunto grande de números de coma flotante que estén en el rango de 0.0 a 1.0 y estén distribuidos uniformemente en todo el rango.* ¿Cómo clasificamos los números de manera eficiente?

Una manera simple es aplicar un algoritmo de clasificación basado en comparación. El límite inferior para el algoritmo de clasificación basado en la comparación (clasificación de combinación, clasificación de montón, clasificación rápida, etc.) es Ω (n Log n), es decir, no pueden funcionar mejor que nLogn.

¿Podemos ordenar la matriz en tiempo lineal? El ordenamiento de conteo no se puede aplicar aquí ya que usamos claves como índice en el ordenamiento de conteo. Aquí las teclas son números de coma flotante. La idea es usar el tipo de cubo. El siguiente es el algoritmo de cubo.





**Complejidad del tiempo:** si suponemos que la inserción en un cubo toma tiempo O (1), entonces los pasos 1 y 2 del algoritmo anterior claramente toman tiempo O (n). El O (1) es fácilmente posible si usamos una lista vinculada para representar un. El paso 4 también toma tiempo O (n) ya que habrá n elementos en todos los cubos.

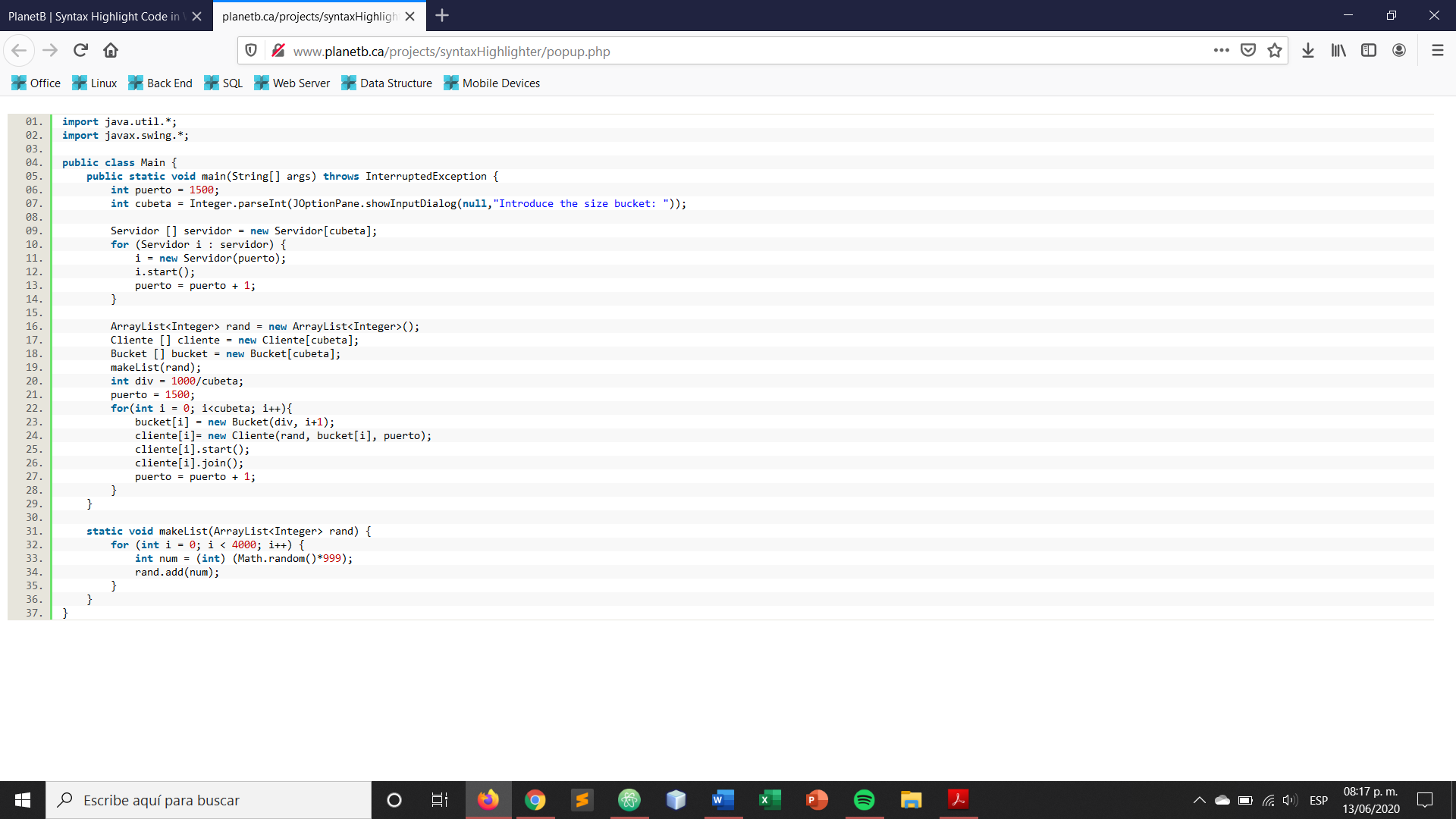
El paso principal para analizar es el paso 3. Este paso también toma un tiempo O (n) en promedio si todos los números están distribuidos uniformemente

DESARROLLO

Iniciamos con el código del Main, ya que este es el que se va a ejecutar e instanciará las demás clases. Lo primero que solicita, es el número de cubetas que vas a utilizar. Posteriormente, iniciaría y ejecutará un arreglo de Servidores igual al número de cubetas para después crear un arreglo de Cliente de igual longitud. Después creará las cubetas necesarias.

La función **makeList(ArrayList<Integer> rand)** crea 4000 números aleatorios en un rango de 0 a 1000. Estos números, son los que se van a utilizar en las cubetas para empezar con el ordenamiento.

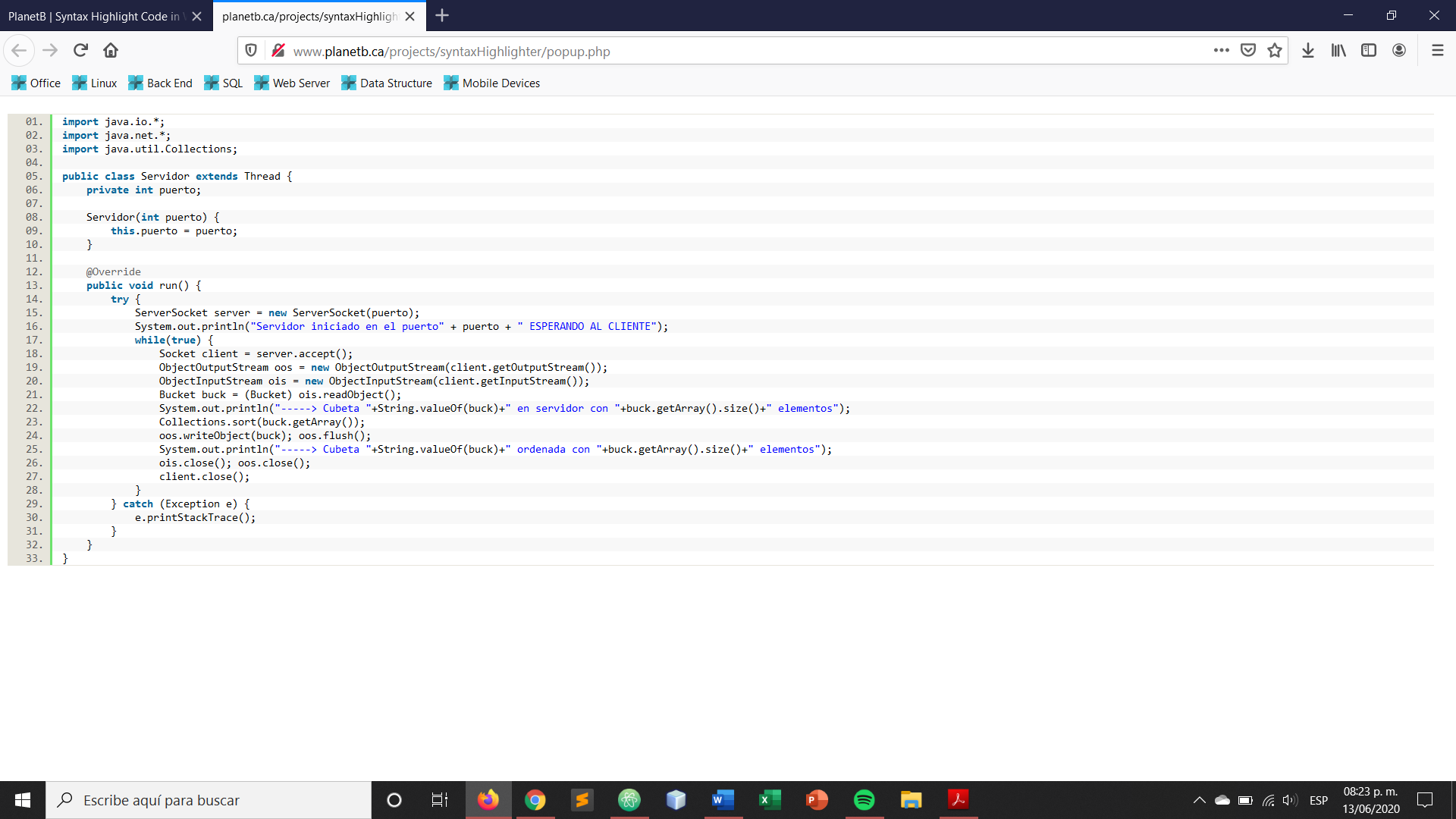
A continuación, se muestra el código implementado de la clase Main.



Ahora, vamos a explicar la parte del Servidor.

Como todos los servidores que hemos implementado en este curso, iniciamos creando un ServerSocket y lo conectamos con un Socket para poder escuchar las peticiones del cliente, La parte importante es la implementación del método *Collections.sort* que está presente en la clase java.util.Collections. Se utiliza para ordenar los elementos presentes en la lista especificada de Colección en orden ascendente. Funciona de manera similar al método java.util.Arrays.sort () pero es mejor, ya que puede ordenar los elementos de Array, así como la lista vinculada, la cola y muchos más presentes en él.

A continuación, se muestra el código de la clase Servidor.java.

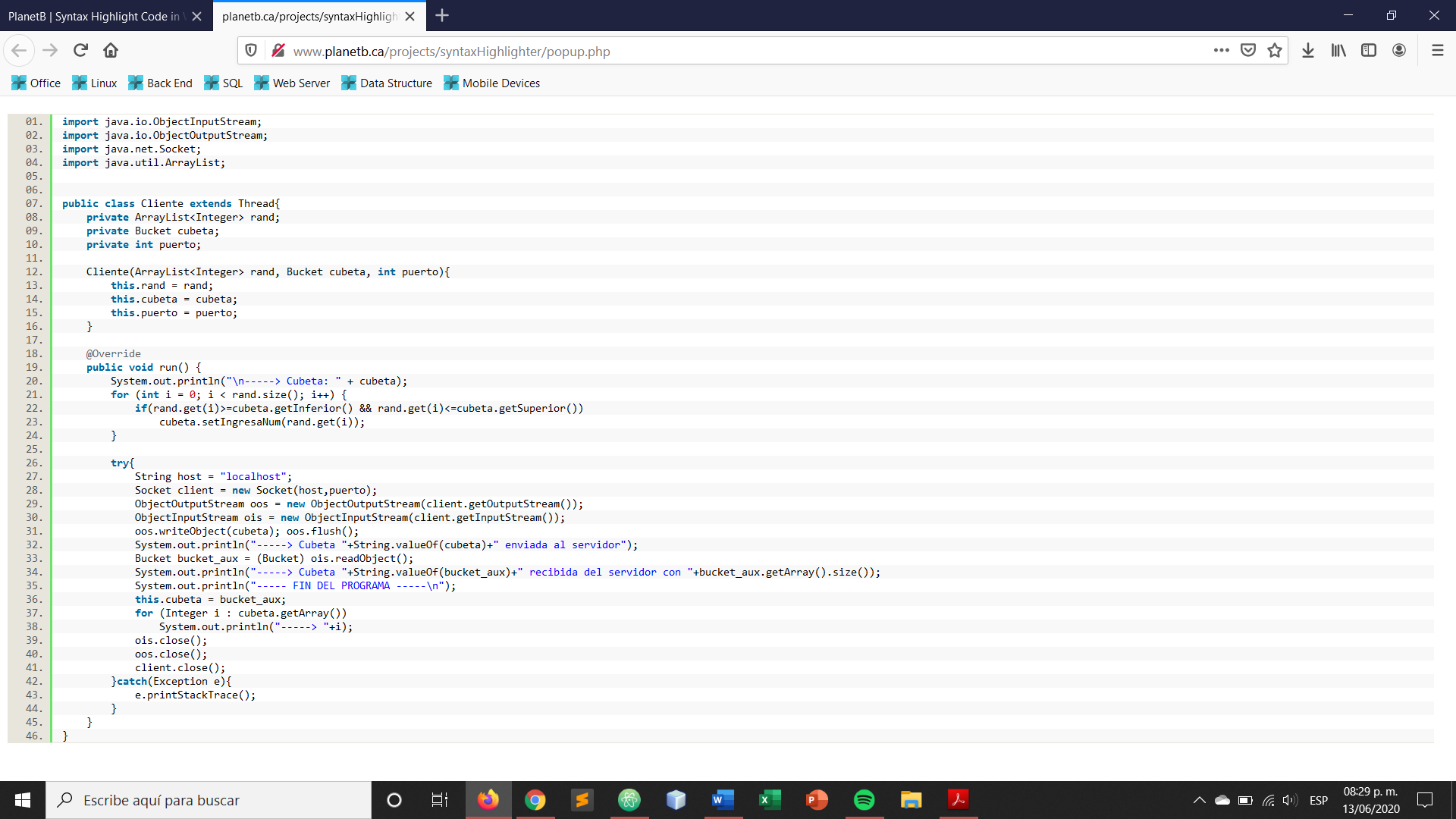


A continuación, se explicará la parte del Cliente.

La clase Cliente.java requiere de tres parámetros que son expresados en su constructor. El ArrayList de enteros son los números por ordenar, la respectiva cubeta que se va a utilizar y el puerto del Servidor correspondiente.

Ingresamos los números a la cubeta y la enviamos al Servidor para que se encargue del ordenamiento de estos. Después, el Servidor nos va a regresar la misma cubeta, pero con los números ordenados, por lo que únicamente los imprimimos para verificar que efectivamente se realizó el proceso de manera exitosa.

El siguiente código corresponde a la parte del Cliente.



FUNCIONAMIENTO DE LA APLICACIÓN

El funcionamiento de esta aplicación es bastante simple. Únicamente debemos ejecutar el Main.java e introducir el número de cubetas que deseamos utilizar para la implementación del algoritmo.

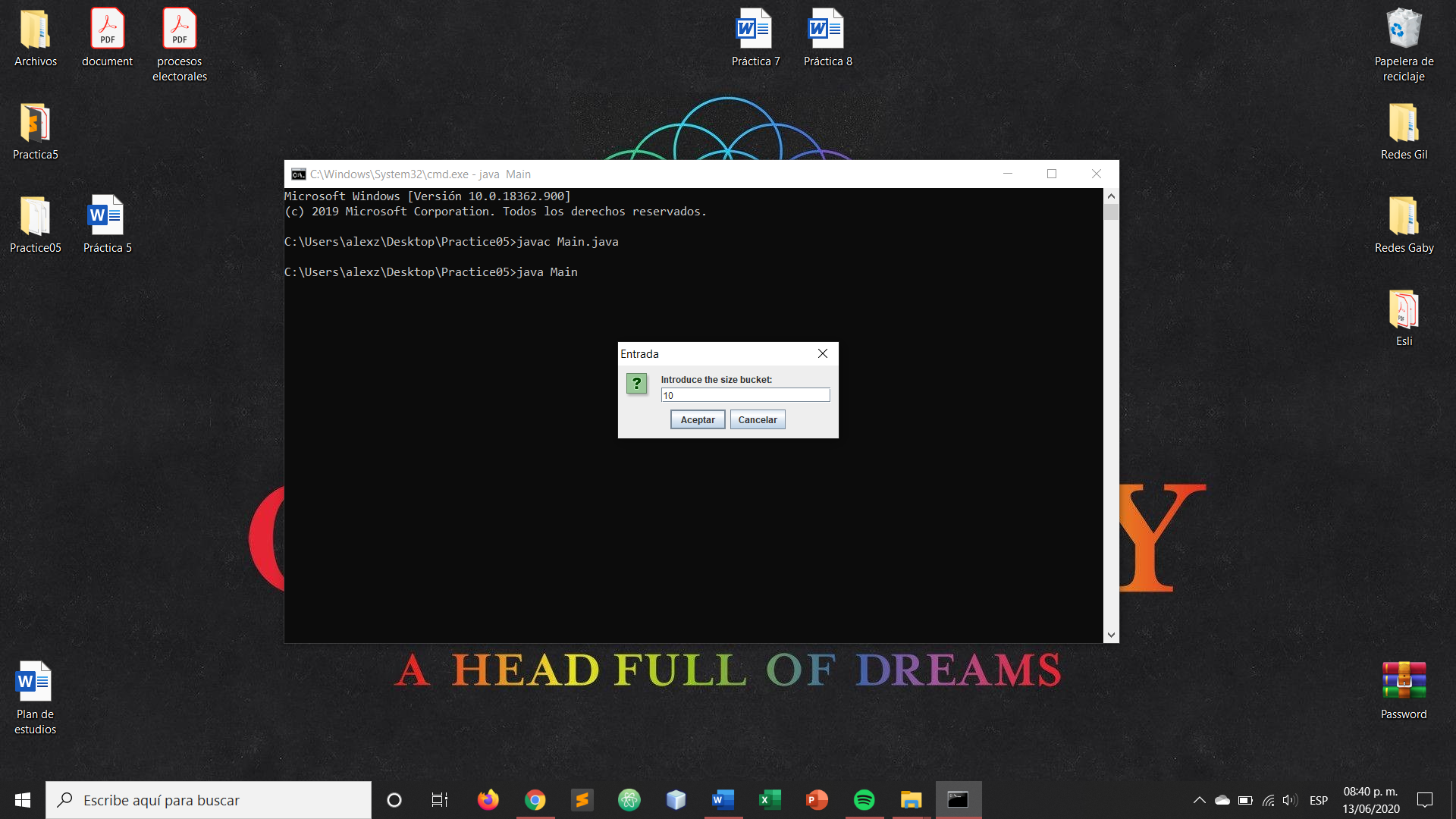


Imagen 1. Número de cubetas.

Una vez que las introducimos, el algoritmo empezara a ejecutarse, ordenando los números que hemos creado aleatoriamente en un rango de 0 a 999. Para poder visualizar el ordenamiento y el uso de las cubetas, imprimimos en pantalla los números utilizados por sus respectivas cubetas.

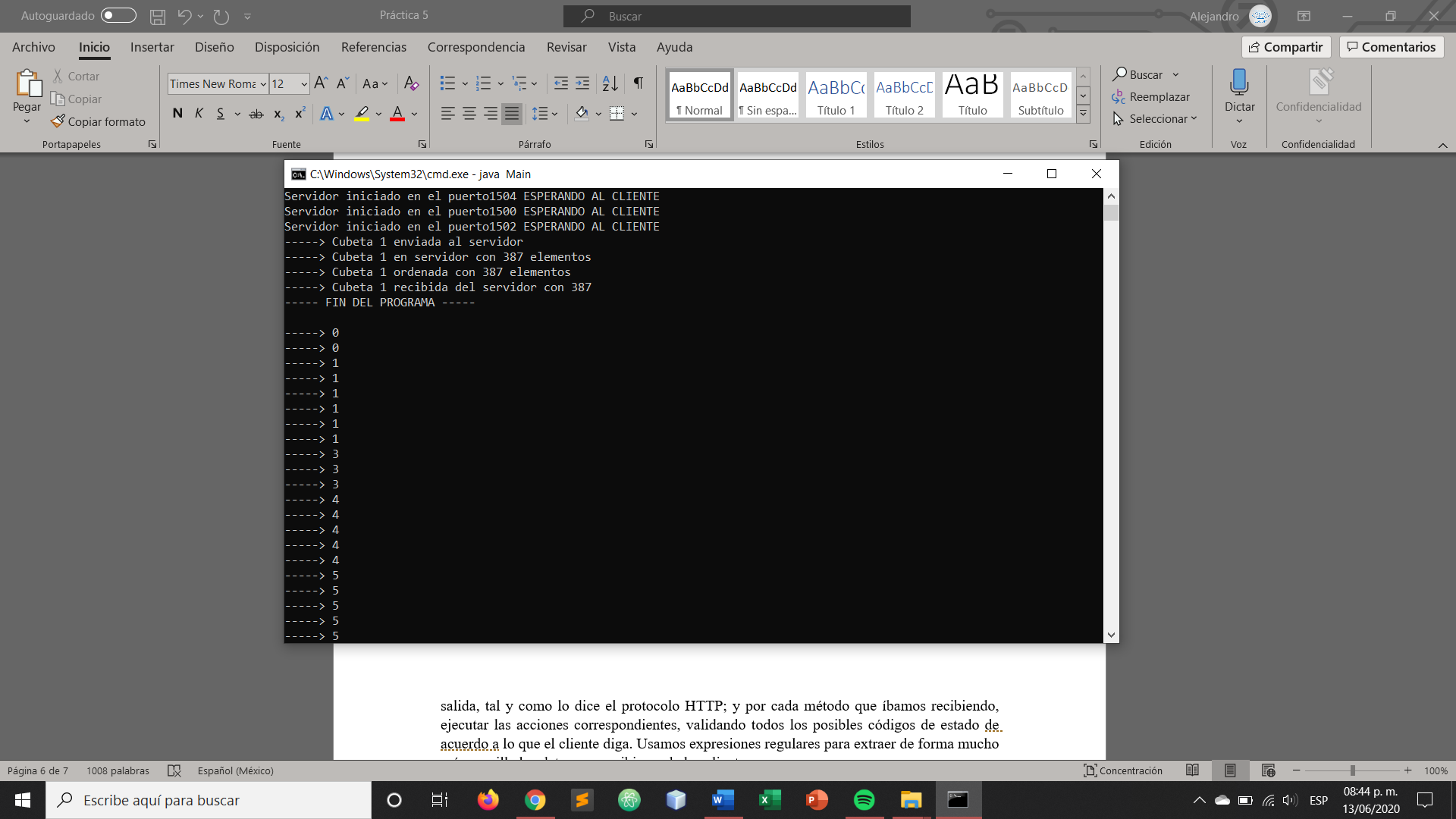


Imagen 2. Cubeta 1 con 387 elementos.

CONCLUSIÓN

El ordenamiento implementado con cubetas tiene la bondad de segmentar nuestros datos y que el procesamiento posterior de este para fines específicos se nos va a facilitar mucho más pues al estar trabajando con rangos de datos al necesitar un numero en específico solo ubicamos el rango y ya solo implementamos un algoritmo de búsqueda, pero este ya no trabará con el total de los datos y con esto conseguimos reducir la complejidad del programa.

El llevar a cabo esta práctica resulto más fácil de lo que parecía al principio, además, se realizó en un lenguaje de programación que facilita mucho las cosas como lo es Java, en algún momento tuvimos inconvenientes, estos se presentaron por no contemplar algunos casos especiales dentro de los ciclos for, sin embargo, logramos resolverlos de manera rápida y logramos el objetivo de la práctica.