La distribución es un algoritmo externo de ordenamiento en el cual se van analizando cada uno de los dígitos de los números que deseamos ordenar, para que, dependiendo del valor del dígito evaluado, el número sea almacenado en algún archivo o estructura auxiliar, y posteriormente, estos números sean recabados de cada una de las estructuras para volver a ser analizados, pero esta vez, en la siguiente posición del dígito del número a evaluar.

Dentro de la clase de ***Distribución***, se realizó un procedimiento para poder realizar este ordenamiento externo a partir de un archivo original que contiene el conjunto de números a ordenar y de un conjunto de 10 archivos auxiliares, que nos permitieron ir almacenando los números dependiendo del valor del dígito que era evaluado en cada caso.

Lo primero que se realiza al elegir el ordenamiento externo por distribución es pedirle al usuario el nombre del archivo de donde se obtendrán los números a ordenar y el tipo de ordenamiento, ya sea ascendente o descendente. Dentro del ***main***, se van a almacenar estos datos y se va a crear una instancia de la clase ***Distribucion***, en la cual, se cuenta con un método constructor el cual al recibir los datos del nombre del archivo y el tipo de ordenamiento, los va a almacenar en atributos de la instancia, siendo estos los atributos con nombre ***“nombreArchivoOriginal”*** y ***“tipoOrdenamiento”*.** Luego, dentro del ***main*** se puede observar que a partir de la instancia que se creó, se manda llamar al método **“*ordenamiento”*.**

Dentro del método **“*ordenamiento”*,** se hace uso de diversos métodos, los cuales se van a explicar a medida que son utilizados, para tener una mejor idea de cómo funciona este algoritmo creado en ***NetBeans*** en el lenguaje de programación ***Java*.** Lo primero que se hace dentro de este método es mandar llamar al método ***“crearArchivo”*,** el cual es un método que no devuelve ningún valor, pero que recibe como parámetro el nombre de un archivo, junto con la extensión del mismo. Para este proyecto, los nombres de los archivos a utilizar fueron almacenados en atributos de la clase, donde cada una de las colas es representada por un atributo de tipo *String*, con nombre **“Q#”** donde el símbolo **“#”** es reemplazado por el dígito que representa a dicha cola, donde el atributo es inicializado y declarado con el nombre del archivo que íbamos a utilizar para almacenar las colas, siendo estos **“Q#.txt”,** donde nuevamente, el **“#”** es reemplazado por el dígito que corresponde a cada cola. Dentro del método **“*crearArchivo”*,** se hace uso de un ***try-catch*,** el cual nos permite a la vez hacer uso de un ***FileWriter*** con el cual se indica cada uno de los archivos que deseamos utilizar, el ***FileWriter*** cuenta con la característica de que en caso de que no existe el archivo lo crea, por lo tanto, en esta función solamente se indicó cada uno de los archivos que deseamos utilizar o “crear”, y por medio del ***FileWriter***, se crean y en caso de existir, gracias al uso del booleano *false*, se va a eliminar todos los datos que contenían dichos archivos.

Después de haber creado todos los archivos auxiliares referidos al manejo de colas o a la impresión del procedimiento o de las iteraciones dentro del ordenamiento, se hace uso de una nueva función llamada **“*lecturaOriginal”*,** el cual es un método que no devuelve ningún elemento y que no recibe ningún tipo de parámetro, sino que trabaja con los atributos que tenemos dentro de la clase. Este método sirve para leer el archivo original, el cual fue indicado por el usuario dentro del ***main*** del proyecto. Se guardan todos los números contenidos dentro de dicho archivo original dentro de un arreglo ***números*,** donde cada número es almacenado dentro de una posición del arreglo gracias a que se utiliza a la **“,”** como un indicador de separación entre números. Luego, se hace uso de un ***ArrayList*** de tipo ***Double*,** con el cual se almacenan los elementos contenidos dentro del arreglo a un ***ArrayList*** que los contiene como números y, finalmente, estos números contenidos dentro del ***ArrayList*** son almacenados en un ***LinkedList*** de tipo ***String*** y se borran todos los elementos contenidos dentro del archivo original.

Como siguiente método a utilizar sería el método **“*ValorRecursivoMáximo”*,** el cual es un método que no devuelve ningún valor y no recibe ningún parámetro, sin embargo, se encarga de recorrer la lista ligada que contiene a todos los elementos de tipo ***String*** y busca cual es la longitud máxima de estos elementos, para que, de esta forma, se tenga un control sobre cuál es el número que cuenta con una mayor cantidad de dígitos para que al momento de realizar el ordenamiento y la revisión de los dígitos de cada número, se verifique verdaderamente la posición de cada uno de los dígitos de todos los números de forma uniforme. Este valor lo almacenará en el atributo **“*ValorRecursivoMaximo”*.**

Ahora se hace uso del método **“*ArregloDeArreglos”*,** que sigue siendo un método que no devuelve ningún valor y no recibe ningún parámetro, pero que trabaja con los atributos de la clase. Dentro de este método, se va a hacer uso de un ***ArrayList*** que contiene ***ArrayList*** de tipo ***Double*.** Esta estructura es inicializada y lo primero que se hace es almacenar todos los números contenidos dentro de nuestro ***ArrayList*** de ***Double*** a la primera posición de cada uno de los ***ArrayList*** contenidos dentro de nuestro arreglo de arreglos, para que, de este modo, se tengan todos los números almacenados dentro de este ***ArrayList.***Posteriormente, se va a descomponer a cada uno de los números en dígitos, donde cada dígito va a ocupar una posición dentro de nuestro arreglo de arreglos de ***Doubles*,** para el caso de trabajar con los puntos decimales, se hace uso de su clave en código **ASCII,** para que, en caso de que se encuentre un punto, éste lo trabaje como un cero. Así mismo, se busca uniformizar el número total de decimales para que, en caso de que el número no contara con punto o no contara con decimales, el método le completara los decimales que le faltaban con dígitos “cero”; del mismo modo, en caso de que el número evaluado fuera muy corto o pequeño, se iba a completar los dígitos que le faltaban mediante el uso de dígitos “cero”. Después de estos procedimientos, dentro del arreglo de arreglos de valores ***Double*,** se tendría un conjunto de arreglos de misma longitud que tendrían sus números en la primera posición de los arreglos contenidos y que se encontrarían uniformizados con la misma cantidad de números decimales y la misma cantidad de dígitos para su evaluación dentro de funciones posteriores.

Finalmente, dentro del método ***“ordenamiento”*,** se va a empezar a realizar la evaluación de cada uno de los dígitos contenidos dentro de los números para realizar el ordenamiento, para esto, se hace uso de un ciclo ***for*,** el cual va a evaluar el último dígito de cada número hasta el primer dígito de cada número, haciendo uso del atributo **“*ValorRecursivoMaximo”*,** que nos indicaba la longitud máxima de los números a ordenar, lo cual lo logramos a partir de la uniformización de números en el método ***“ArregloDeArreglos”*.** Mediante el método de ***“radixSort”*,** no se devuelve ningún valor, pero se recibe como parámetro la posición de los números que se está evaluando, por lo que, se va a trabajar con la estructura arreglo de arreglos que creamos con anterioridad donde descompusimos todos los números en sus dígitos y dependiendo del valor de los dígitos, se iba a escribir el número original en cada una de las colas o archivos auxiliares que utilizamos hasta que ya no se contara con más números por distribuir o para clasificar por medio de los dígitos contenidos dentro de la posición verificada.

Ahora, dependiendo del tipo de ordenamiento ingresado por el usuario, el cual es evaluado por un *if*, se va a hacer uso de los métodos **“*pasarAOriginalAscendente”* o “*pasarAOriginalDescendente”*,** donde estos métodos hacen uso de dos funciones que se van a explicar a continuación.

El primer método del que hacen uso es del método ***“lecturaArchivo”*,** el cual devuelve un ***ArrayList*** de tipo ***String*** y recibe como parámetros el nombre de un archivo y un identificador de este archivo, para que, dentro de este método se lea el archivo indicado por el usuario y se almacenen los números dentro de un arreglo, donde cada uno de los números serán divididos tomando como indicador la separación de las comas y almacenando los números en posiciones individuales. Los números contenidos son mostrados en pantalla y son escritos en un archivo auxiliar que luego nos permitirá realizar un análisis de todas las iteraciones y el proceso de ordenamiento del archivo original. Se devuelve el ***ArrayList*** que contiene a todos los números almacenados dentro del archivo.

Como segundo método, se hace uso del método **“*vaciarColas”*,** el cual no devuelve ningún valor, sin embargo, recibe como parámetro un ***ArrayList***de tipo***String*,** el mismo que fue recuperado de la función explicada con anterioridad, para que, de este ***ArrayList*,** se vayan sacando de la cola cada uno de los elementos almacenados y estos sean escritos en el archivo original mediante un método ***write()*** de la clase ***FileWriter*.** Dentro de este método se debe verificar que el archivo original se sobreescriba para que no se pierda ningún número escrito previamente por otras colas.

Después de haber explicado estos dos métodos nos regresamos al análisis de las funciones **“*pasarAOriginalAscendente”* o “*pasarAOriginalDescendente”*,** donde lo primero que se realiza es leer todas las colas (sin importar el orden), para almacenarlas dentro de ***ArrayList*** de tipo ***String*** y tener todos los números almacenados en estas estructuras y finalmente, dependiendo del ordenamiento ascendente o descendente, se van a vaciar las colas al archivo original mediante el método **“*vaciarColas”*.** Si se realiza un ordenamiento ascendente se vaciarán las colas empezando por la cola cero, siguiente en la cola uno y así sucesivamente hasta llegar a la cola nueve; en caso de ser un ordenamiento descendente, se empezará a vaciar por la cola nueve, se seguirá con la cola ocho hasta llegar a la cola cero.

Finalmente, mediante un ***if*** se va a evaluar si no nos encontramos evaluando la primera posición de cada uno de los dígitos, de no ser así, se va a leer el archivo original mediante el método **“*lecturaOriginal”*,** dentro del cual, nuevamente se guardan todos los números contenidos dentro de dicho archivo original dentro de un arreglo *números*, donde cada número es almacenado dentro de una posición del arreglo gracias a que se utiliza a la “,” como un indicador de separación entre números y se hará uso de un ***ArrayList*** de tipo ***Double*,** con el cual se almacenan los elementos contenidos dentro del arreglo a un ***ArrayList*** que los contiene como números que se encuentran a un paso menos de encontrarse completamente ordenados. Finalmente, se borran todos los elementos contenidos dentro del archivo original.

Nuevamente, mediante el uso del método **“*ArregloDeArreglos”*,** se va a descomponer a cada uno de los números en dígitos, y se volverá a realizar las verificaciones explicadas con anterioridad para uniformizar el largo de los números y el número de decimales que contienen.

Tras todo este proceso, y gracias al ciclo ***for*** que realiza los métodos de ***“radixSort”*,** **“*pasarAOriginalAscendente”* o “*pasarAOriginalDescendente”*, “*lecturaOriginal”* y** **“*ArregloDeArreglos”*,** se van a ir evaluando cada una de las posiciones de cada uno de los dígitos de los números contenidos dentro del archivo original y serán ordenados según las indicaciones del usuario, al sacarlos y meterlos dentro de las diferentes colas para que, tras cada iteración, se encuentren un poco más ordenados para finalmente, tener un orden final y poder visualizar todo el procedimiento realizando en el archivo **“*IteracionesDistribucion.txt”*.**