Método de la Ingeniería
Aplicación para la Solución de un Problema
Miguel Angel Romero Rosas y Jonathan Arenas

Contexto Problemático

Una empresa de fabricación de microprocesadores está evaluando la posibilidad de implementar varios algoritmos de ordenamiento como instrucciones básicas de su próximo coprocesador matemático. la empresa ha decidido implementar tres (3) algoritmos diferentes de ordenamiento que permitan ordenar, muy rápidamente, números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.

Desarrollo de la Solución

Paso 1. Identificación del Problema

Identificación de necesidades y síntomas

- Una empresa de fabricación de microprocesadores desea implementar tres algoritmos de ordenamientos como una operación nativa de su próximo coprocesador.
- La solución al problema debe garantizar un ordenamiento muy rápidamente de números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.
- La solución del problema debe de ser eficiente ya que esto evitaría que el procesador principal tenga que realizar estas tareas de cómputo intensivo.
- El coprocesador matemático que se quiere implementar debe acelerar el rendimiento del Sistema por el hecho de esta descarga de trabajo en el procesador principal.
- Una empresa de fabricación de microprocesadores desea implementar tres algoritmos de ordenamientos como una operación nativa de su próximo coprocesador.
- Sus anteriores líneas de coprocesadores no implementaban o no soportaban varios algoritmos de ordenamiento.
- los algoritmos deben permitir ordenar muy rápidamente.
- La empresa desea un software para probar los algoritmos qué serán implementados en el hardware.
- No disponen de un software que permita evaluar la eficiencia y eficacia de los algoritmos en el hardware
- La empresa desea poder ingresar los valores que desean ordenar.
- La empresa desea que el programa genere aleatoriamente los valores.
- La empresa desea tener control sobre la cantidad total de valores generados y el intervalo en el cual se generarán los mismos.
- La empresa, además, requiere poder elegir entre números repetidos o ausencia de estos en los números generados.
- La generación aleatoria debe permitir elegir entre unas configuraciones de los valores en la secuencia.
- La empresa requiere que el software ordene los valores ingresados utilizando el algoritmo apropiado de acuerdo con el tipo de número a ordenar y el intervalo de números generados.
- Finalmente, La empresa requiere que la interfaz gráfica muestre el tiempo que se demoró el algoritmo en ordenar las entradas.

Definición del Problema

Una empresa que fabrica microprocesadores desea incluir en su próximo coprocesador la funcionalidad de ordenar entradas numéricas grandes, ya sean enteros o de coma flotante. Para esto, requieren incluir tres algoritmos de ordenamiento en el hardware que tengan un buen rendimiento dependiendo el tipo de número y la magnitud de números de la entrada. Además, Desean un software que permita evaluar estos algoritmos en el hardware. Así, la empresa requiere que el software lea y genere entradas numéricas enteras y de coma flotante. Para el caso de la generación, el software, precisa brindar control sobre el número total de datos, el intervalo de generación, existencia de números repetidos y elección entre posibles configuraciones de los valores en la secuencia. Finalmente, ordenar la serie de datos y mostrarlos junto al tiempo empleado.

Paso 2. Requerimientos Funcionales:

- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda ingresar los valores que se desean ordenar.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda configurar el tipo de dato a generar.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda configurar la cantidad total de números a generar.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda configurar el intervalo en el cual se generarán los números.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda decidir la ausencia o existencia de números repetidos.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda elegir entre las siguientes configuraciones de los valores en la secuencia: (a)valores ya ordenados (b)valores ordenados inversamente (c)valores en orden completamente aleatorio (d)valores desordenados en un % dado por el usuario.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda generar aleatoriamente valores.
- la interfaz gráfica debe dar al modelo del mundo los valores que se desean generar junto con la configuración deseada para la serie de datos de salida.
- El modelo del mundo debe generar la serie de datos con las especificaciones brindadas por la interfaz del usuario.
- el modelo del mundo debe brindar a la interfaz del usuario la serie de datos generados.
- La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda ordenar los valores.
- La interfaz gráfica debe dar al modelo del mundo los valores que se desean ordenar.
- El modelo del mundo debe brindar al programa 3 algoritmos de ordenamiento para entradas grandes enteras y de coma flotante.
- El modelo del mundo debe elegir un algoritmo mediante el cual, dependiendo de la entrada, se llevará a cabo la operación de ordenar.
- El modelo del mundo debe brindar a la interfaz gráfica los valores de la secuencia.
- El modelo del mundo debe brindar a la interfaz gráfica el tiempo empleado en el proceso de ordenamiento.
- La interfaz gráfica debe mostrar al usuario la serie de datos ordenados.
- La interfaz gráfica debe mostrar al usuario el tiempo empleado en el proceso de ordenar.

Paso 3. Recopilación de Información

Definiciones

Números en formato coma flotante:
es una forma de notación científica usada en los microprocesadores con la cual se pueden representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta, y con la que se pueden realizar operaciones aritméticas.

Sin embargo, hay que tener en cuenta que no se puede evaluar si un coma flotante es mayor, o menor, que otro de la misma forma que se haría con un número entero. Pues, podemos tener A= 1,0 y B=1,0 pero puede que B= 1,00000001.

Cada dato float cuesta 4 bytes.

También, tienen un estándar que convierte los datos de coma flotante en bytes, se llama IEEE 754.

Para comprobar números flotantes no se evalúa si son iguales. Por el contrario, se busca evaluar si su diferencia es muy pequeña. El margen de error que compara esta diferencia normalmente se llama épsilon. En su forma más simple:

```
if(Math.abs(a-b) < 0.00001)
```

Pero como épsilon puede ser, o muy grande para floats pequeños o muy pequeño para floats muy grandes, es necesario ver si el error relativo es menor que épsilon:

```
if(Math.abs((a-b)/b) < 0.00001)
```

Hay casos especiales donde esto falla:

- Cuando tanto a como b son cero. 0.0/0.0 es NaN, lo que provoca una excepción en algunas plataformas o devuelve falso para todas las comparaciones.
- Cuando solo b es cero, la división devuelve infinito, lo que también puede causar una excepción, o es mayor que épsilon incluso cuando a es más pequeño.
- Devuelve «falso» cuando tanto a como b son muy pequeños, pero a ambos lados del cero, incluso cuando son los números no nulos más pequeños.
- Además, el resultado no es conmutativo (`nearlyEquals (a, b)` no es siempre lo mismo que `nearlyEquals (b, a)`). Para solucionar estos problemas, el código tiene que ser mucho más complejo, así que necesitamos meterlo en una función:

```
public static Boolean nearlyEqual(float a, float b, float epsilon)
{
    final float absA = Math.abs(a);
    final float absB = Math.abs(b);
    final float diff = Math.abs(a - b);

    if (a == b) { // Atajo, maneja los infinitos
        return true;
    } else if (a * b == 0) { // a o b o ambos son cero
        // El error relativo no es importante aquí
        return diff < (epsilon * epsilon);
```

```
        } else { // Usar el error relativo
            return diff / (absA + absB) < epsilon;
        }
    }
```

Este método pasa las pruebas para muchos casos especiales importantes, pero como puedes ver, utiliza cierta lógica no trivial. En particular, tiene que utilizar una definición totalmente distinta del margen de error cuando a o b son cero, porque la definición clásica del error relativo es inútil en esos casos.

Hay algunos casos en los que el método de arriba todavía produce resultados inesperados (concretamente, es mucho más estricto cuando un valor es casi cero que cuando es exactamente cero), y algunas de esas pruebas para las que fue desarrollado probablemente especifica un comportamiento que no es apropiado para algunas aplicaciones.

Comparando valores de punto flotante como enteros

Hay una alternativa a aplicar toda esta complejidad conceptual a una tarea aparentemente tan sencilla: en lugar de comparar a y b como números reales, podemos concebirlos como pasos discretos y definir el margen de error como el número máximo de valores de punto flotante posibles entre esos dos números.

Esto es conceptualmente muy evidente y fácil y tiene la ventaja de que escala implícitamente el margen de error relativo con la magnitud de los valores. Técnicamente es un poco más complejo, pero no mucho más de lo que puedas pensar, porque los números de punto flotante del IEEE 754 están diseñados para mantener su orden cuando sus secuencias de bits se interpretan como enteros.

Sin embargo, este método requiere que el lenguaje de programación soporte conversión entre valores de punto flotante y secuencias de bits enteras.

Coprocesador: es un microprocesador de un ordenador utilizado como suplemento de las funciones del procesador principal (la CPU). Las operaciones ejecutadas por uno de estos coprocesadores pueden ser operaciones de aritmética en coma flotante, procesamiento gráfico, procesamiento de señales, procesado de texto, criptografía, etc.

Operaciones con aritmética:

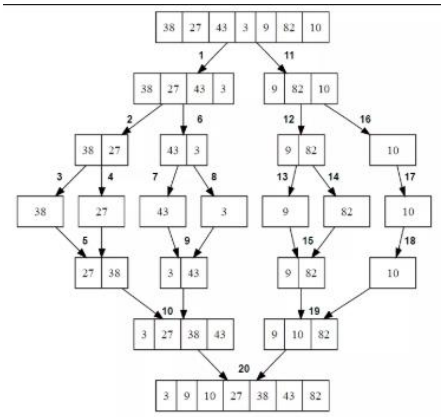
son: suma, resta, multiplicación, división, potenciación, división entera. Los operadores lógicos son: "y", "o", "no" y "o exclusivo".

Numero entero:

es un elemento del conjunto numérico que contiene los números naturales N ={1,2,3,4….}, sus opuestos y el cero.1 Los enteros negativos, como −1 o −3 (se leen «menos uno», «menos tres», etc.), son menores que cero y todos los enteros positivos.

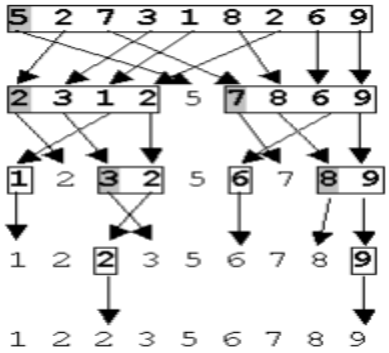
Alternativas:

Alternativa 1: Método Merge Sort:



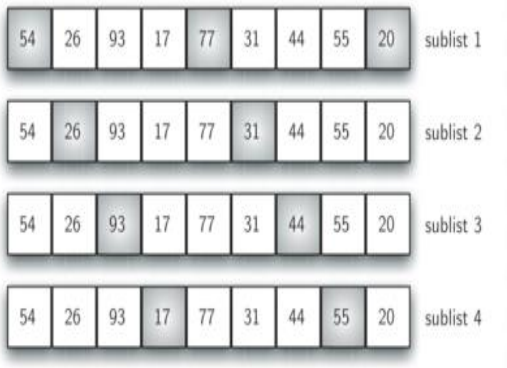
El algoritmo de ordenación por combinación, Merge Sort, se basa en la técnica Divide y Vencerás, ordena recursivamente un conjunto de elementos dividiéndolo en dos, ordenando cada una de estas partes en forma independiente y combinando los dos resultados. Este a su vez **recibe como entrada un arreglo de números enteros** denominado v, lo parte utilizando el **método copyOfRange** de la clase en java, se llama recursivamente con cada una de las dos partes como argumento y, una vez terminada la ordenación de dichas partes, invoca al proceso de combinación de las dos respuestas implementado en el método combinar, el cual recibe como entrada el arreglo original y las dos mitades de este previamente ordenadas que serán combinadas en el arreglo original. Este método es muy eficiente, comparado con otros métodos de ordenación, **pero solo es para números enteros.**

Alternativa 2: Método Quick Sort:



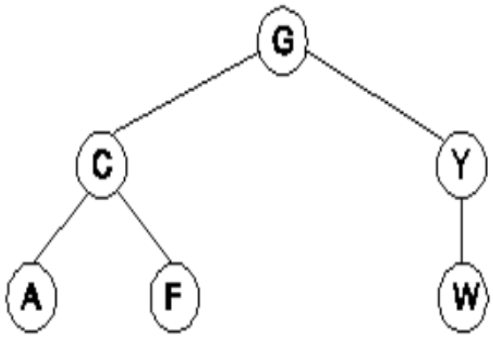
El objetivo de este algoritmo es dividir recursivamente el vector en partes iguales, indicando un elemento de inicio, fin y un pivote, que nos permitirá segmentar nuestra lista. Una vez dividida, lo que hace, es dejar todos los mayores que el pivote a su derecha y todos los menores a su izq. Al finalizar el algoritmo, nuestros elementos están ordenados. Esto lo que nos va a permitir en el programa es ordenar de una manera más eficiente y eficaz los números que lleguen como entrada al algoritmo.

Alternativa 3: Método Shell Sort



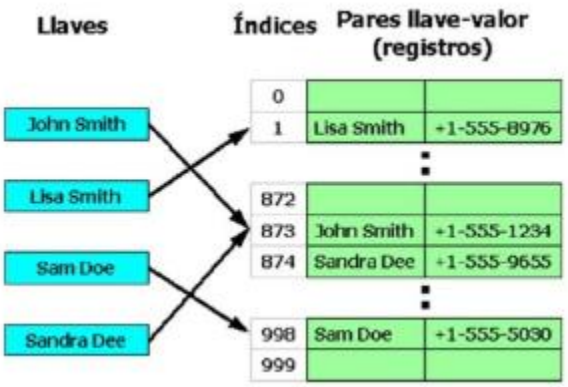
Es una mejora del método de inserción directa, utilizado cuando el array tiene un gran número de elementos.
Cualquier algoritmo de ordenación que intercambia elementos adyacentes (como los algoritmos burbuja, selección o inserción) tiene un tiempo promedio de ejecución de orden cuadrático (n^2). El método Shell mejora este tiempo comparando cada elemento con el que está a un cierto número de posiciones llamado salto, en lugar de compararlo con el que está justo a su lado. Este salto es constante, y su valor inicial es N/2 (siendo N el número de elementos, y siendo división entera).

Alternativa 4: Arboles (Estructuras de Datos)



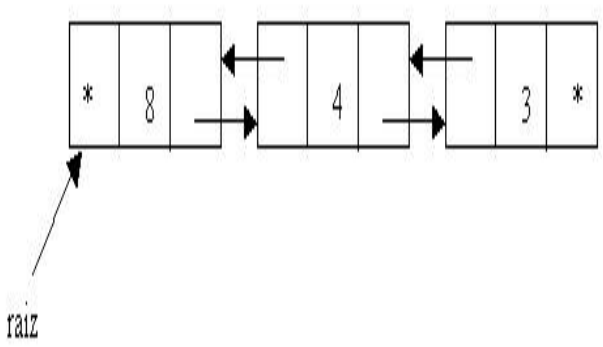
A los arboles ordenados de grado dos se les conoce como arboles binarios ya que cada nodo del árbol no tendrá más de dos descendientes directos, los cuales se pueden llamar izquierda o derecha. Las aplicaciones de los arboles binarios son muy variadas ya que se les puede utilizar para representar una estructura en la cual es posible tomar decisiones con dos opciones en distintos puntos. Esta estructura de datos nos va a facilitar un poco a la hora de introducir los datos al Sistema, ya que nos estará organizando, tanto los enteros como los números de coma flotante en una jerarquía de un árbol.

Alternativa 5: Tabla Hash



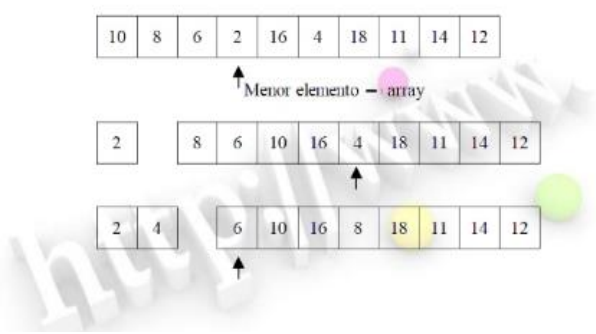
Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información. Las tablas hash almacenan la información en posiciones pseudo-aleatorias, así que el acceso ordenado a su contenido es bastante lento. Otras estructuras como árboles binarios auto-balanceables tienen un tiempo promedio de búsqueda mayor (tiempo de búsqueda O(log n)), pero la información está ordenada en todo momento.

Alternativa 6: Listas Doblemente Enlazadas



Observemos que una lista doblemente encadenada tiene dos punteros por cada nodo, uno apunta al nodo siguiente y otro al nodo anterior. Seguimos teniendo un puntero (raíz) que tiene la dirección del primer nodo. El puntero sig del último nodo igual que las listas simplemente encadenadas apunta a null, y el puntero ant del primer nodo apunta a null. Se pueden plantear Listas tipo pila, cola y genéricas con enlace doble. Hay que tener en cuenta que el requerimiento de memoria es mayor en las listas doblemente encadenadas ya que tenemos dos punteros por nodo.

Alternativa 7: Método de ordenamiento por Selección:



La ordenación por selección funciona seleccionando el menor elemento de la matriz y llevándolo al principio; a continuación, selecciona el siguiente menor y lo pone en la segunda posición de la matriz y así sucesivamente. Usando el método por selección podremos ordenar los elementos enteros y de coma flotante de una manera más eficaz.

Paso 4. Elementos relacionados entre las Alternativas creativas escogidas

Una de las relaciones que Podemos encontrar entre las soluciones creativas es que todas tratan de organizar números. Si bien, cada uno tienen una manera más eficiente para resolver el problema en el peor de los casos, todas ellas están pensadas para satisfacer la necesidad de ordenar una entrada de números. Las practicas que se realizan en cada una varían demasiado entre cada alternativa. Pero algo en común es que se pueden aplicar a arreglos de forma lineal. La otra característica es la facilidad en implementación algorítmica: Las soluciones encontradas son compatibles con las operaciones aritméticas básicas de un equipo de cómputo moderno, hacienda el problema no complejo de tratar, ya que las soluciones escogidas son posibles en todos sus sentidos.

Paso 5. Fuentes:

http://puntocomnoesunlenguaje.blogspot.com/2014/09/metodo-shell-de-ordenacion.html
https://jarroba.com/tablas-hash-o-tabla-de-dispersion/
https://www.monografias.com/trabajos92/arboles-binario/arboles-binario.shtml
https://estructuradedatosunivia.wordpress.com/2014/11/18/metodos-de-ordenamiento-mergesort/
http://puntoflotante.org/errors/comparison/

Fase 3. Búsqueda de Soluciones Creativas

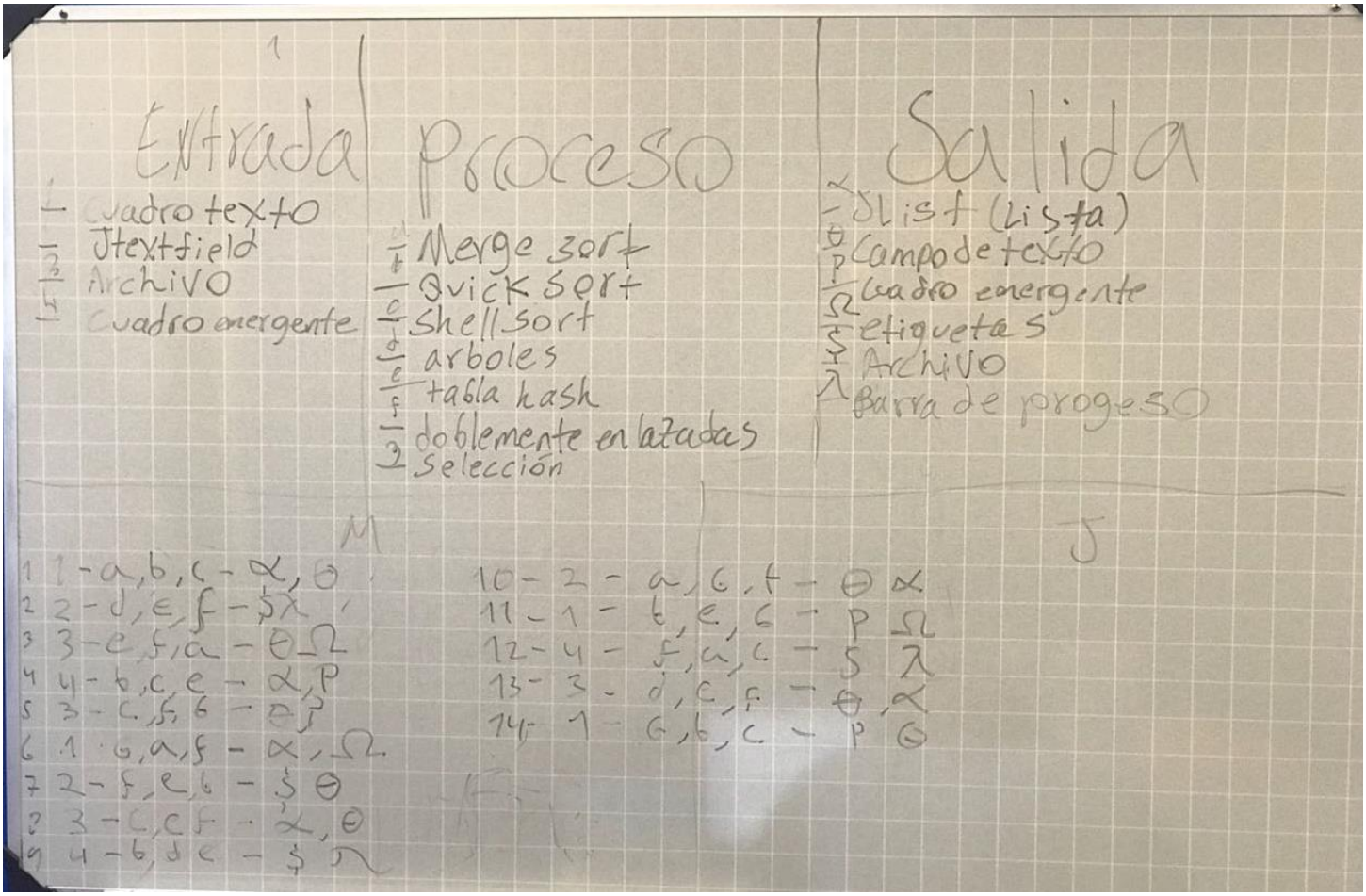
Paso 1: método generador de ideas:

Para desarrollar las 7 alternativas creativas usamos el método de lluvia de ideas. Primero, dividimos el problema en tres categorías: entradas, procesos, salidas. Luego, en cada categoría, enlistamos ideas abstractas que nos sirven como sustantivos para la siguiente etapa. Categorías como “procesos” tiene ya predefinidas las alternativas investigadas en el paso anterior de recolección de información. Con las categorías de entrada y salida lo que buscamos es buscar componentes que nos permitan darnos una idea de como recibir y mostrar la información. A continuación, cada componente enlistado en cada categoría es indexado. Por ejemplo, las ideas que salieron en la categoría de entrada están enumeradas; las categorías en proceso están ordenados alfabéticamente y en la categoría de salidas están enlistadas con letras griegas.

Una vez teniendo llena esta tabla, cada uno procede a escribir combinaciones tomando componentes de cada categoría. Decidimos poner dos condiciones, la primera es solo tomar un componente de la categoría de entrada, y tres, diferentes, de la categoría de procesos. De esta forma, nos aseguramos tener un solo componente de entrada y los tres métodos de ordenamiento requeridos. Con respecto a la salida decidimos seleccionar mas de uno pues nos daría una general, o alternativas, para mostrar el resultado.

Un ejemplo del procedimiento es el siguiente:

1,a,b,c,delta,teta esto quiere decir que la idea se arma con el sustantivo 1 de la categoría de entrada, primero, segundo y tercero de la categoría de proceso y el componente nombrado delta y el componente nombrado teta de la categoría de salida.



Con este formato, procedemos a armar oraciones que representen una idea para solucionar nuestro problema. Continuamos seleccionado las 7 ideas postuladas por cada uno, las unimos para mejorarlas y sacamos las 7 ideas que serán postuladas para el siguiente proceso el cual es analizarlos.

Propuestas de Soluciones Creativas:

Propuestas Miguel:

1) La entrada de todos los elementos va a ser recibida por medio de un campo de texto, el cual después de haber ingresado los elementos hará uso de los métodos de ordenamiento Margesort, Quicksort y Shellsort , y una vez ya organizados los elementos me va a arrojar el resultado en una lista que estará dentro de un campo de texto.
2) La entrada de los elementos será ingresada por medio de un JTextField, y una vez los elementos estén ingresados haremos uso al llamado de los métodos Shellsort, y selección, los cuales organizaran la estructura de datos doblemente enlazada.

3) La entrada de los elementos será por medio de un cuadro emergente, el cual una vez los elementos estén ingresados y separados por medio de un "- ", agregará los elementos en un árbol binario, también se podrá hacer uso de los métodos de ordenamiento QuickSort, o se podrán agregar los elementos en una estructura doblemente enlazada.

4) La entrada de los elementos será por medio de un JTextField, el cual una vez ingresados los elementos a ordenar, hará llamado a los métodos MergeSort, Selección. Una vez los elementos estén organizados, estos serán mostrados por medio de un campo de Texto o por medio de una lista.

5) Los elementos a Ordenar serán introducidos al programa por medio de un cuadro emergente, y una vez introducidos estos serán guardados como una lista doblemente enlazada, los cuales podrán ser ordenador por medio de los métodos MergeSort y Selección y serán sacados por medio de un campo de texto.

6) Los elementos a Ordenar serán introducidos al programa por medio de un Archivo de texto, estos serán leídos e ingresados como una lista doblemente enlazada, los cuales serán ordenados por medio de los métodos MergeSort y QuickSort. Una vez ya ordenados serán mostrados en el sistema por medio de un cuadro emergente.

Propuestas Jhonatan:

1)Leer las entradas por un archivo y procesarlas por medio del mergesort, Quicksort o árboles. Para luego, mostrarlas en un JList(lista) y en un campo de texto.

2)Leer las entradas por medio de un cuadro de texto (JtextArea) y procesarlas por medio del mergesort, Quicksort y heapsort. Para luego, mostrarlas en un campo de texto, en etiquetas para los tiempos de ejecución y una Lista (JList);

3)Leer las entradas por medio de un cuadro de texto (JtextArea) y procesarlas por medio de shellsort, tablas hash y selectionSort. Para luego, mostrarlas en un campo de texto, en etiquetas para los tiempos de ejecución y una barra de progreso.

4)Leer las entradas por un archivo y procesarlas en un mergesort, selectionsort, y listas doblemente enlazadas. para luego, mostrarlas en una lista (JList) y en un campo de texto.

5)Leer las entradas por un cuadro emergente (JMessageDialog) y procesarlas por medio del mergesort, quicksort y shellsort. Para luego guardar un archivo con los resultados, mostrar el proceso con una barra de progreso y los datos en un campo de texto.

6)leer las entradas por un archivo y procesar las por el shellsort, arboles binarios y el heapsort. Para luego mostraros en un campo de texto, en etiquetas para los tiempos, mostrar el progreso en una barra y finalmente guardar un archivo

Fase 4. ALTERNATIVAS DE SOLUCIÓN CREATIVAS:

1**) Alternative 1:**

El usuario tiene la opción de ingresar las entradas por medios manuales. Existen dos posibilidades: por medio de cuadros emergentes y por archivos de texto plano. el usuario verá un cuadro donde podrá ingresar los números separados por comas. También, puede elegir la ruta donde tiene su archivo de texto plano con los números que desea ordenar. Estos números, se agregarán a la lista doblemente enlazada y cuando el usuario decida ordenarlos, los mismos serán procesados por el método mergesort, selectionsort o shellsort según los criterios establecidos. Finalmente, se mostrarán en una JList la lista de forma ordenada.

2) Alternative 2:

El usuario ingresará los datos por medio de un campo de texto. En él, el usuario deberá ingresar los números que desea ordenar con un guion que separa uno de otro. Una vez ingresados, se guardarán en un arreglo lineal. Cuando el usuario de la orden de guardar los números, el campo de texto se vaciará. El programa deberá controlar todo con respecto a si el campo de texto es vacío o se ingresan datos diferentes a enteros, flotantes y guiones. Luego, cuando el usuario decida ordenar los datos, la lista será ordenada por los métodos mergesort, quicksort y heapsort, dependiendo los criterios de la complejidad asintótica. Finalmente, se mostrará el arreglo ordenado en un campo de texto y en una lista. Además, mostrará el tiempo de ejecución, el algoritmo utilizado, el número de entradas ingresadas, y la cantidad de memoria utilizada cada una en una etiqueta.

3) Alternative 3:

El usuario ingresará los datos por medio de un campo emergente. Dicho cuadro, aparecerá cuando el usuario de la orden de añadir números a la lista que desea ordenar. Una condición que debe cumplir el usuario y, el programa hacer que se cumpla, es que los números deben ir separados por guiones. Los datos serán almacenados en un arreglo lineal. Una vez que el usuario decida ordenar los datos, el programa deberá determinar si, debe ordenarse por mergesort o quicksort, o bien crear un árbol binario de búsqueda y usar el método de post order o in order para devolver el arreglo ordenado, dependiendo de los criterios de la complejidad asintótica. Posteriormente, se mostrará el arreglo ordenado en un campo de texto donde también se verá que estrategia se implementó, cuanto tiempo tardó en realizar la operación, los recursos de memoria que se utilizaron y la cantidad de datos ingresados. Finalmente, se exporta un archivo de texto plano con los números ordenados, el método que lo ordeno, el tiempo que se demoró, la cantidad de entradas que se ingresaron y el total de memoria utilizada.

4) Alternative 4:

El usuario deberá ingresar los datos que desea ordenar por medio de un archivo de texto plano. El programa tendrá un campo donde el usuario podrá ingresar la ruta de su archivo y por medio de un botón se leerá. Los números en este archivo deben separados por un guion y los decimales deben tener punto. Si no se encuentra ningún error una vez terminado el proceso de leer el archivo, se almacenarán los datos en un arreglo lineal en el programa y se mostrarán en un campo de texto. En el caso de que se presente algún problema leyendo el archivo, el programa proveerá al usuario una indicación sobre el causante del problema. Una vez el usuario decida ordenar los datos, deberá presionar el botón de ordenar. Hecho esto, el arreglo será procesado por los

métodos heapsort, shellsort o mergesort, dependiendo los criterios del análisis asintótico. Finalmente, se mostrará una etiqueta indicando el método que soluciono el problema, el tiempo empleado, la cantidad de datos ingresados y el total de memoria utilizada y una lista con los datos ordenados.

5) Alternative 5:

El usuario deberá ingresar los datos que desea ordenar definiendo la ruta de su archivo de texto plano con los datos. El programa tendrá un botón que permite mostrar un cuadro emergente. Permitiendo así, que el usuario ingrese la ruta de su archivo. Una vez terminado el proceso de leer el archivo, si no se encuentra ningún error, se procede a almacenar los datos en un arreglo doblemente enlazado. En caso de que se encuentre un error, el programa proveerá al usuario un mensaje con la causa del error. Los números en este archivo deben separados por un guion y los decimales deben tener punto. Cuando el usuario decida ordenar el arreglo, solo deberá oprimir el botón de ordenar datos. Una vez realizada esta acción, el programa decidirá usar el selectionSort, Shellsort o implementar una tabla hash para ordenar los números dependiendo los criterios del análisis asintótico. Finalmente, se mostrará la lista ordenada en una lista y un campo de texto con la información de qué algoritmo o estructura se usó, el tiempo de ejecución, el espacio en memoria utilizado y el número de entradas que se ingresaron.

6) Alternative 6:

El usuario deberá ingresar los datos que desea ordenar escribiéndolos en un campo de texto. Él deberá separar los números por guiones y escribir los decimales con un punto (.). Una vez haya escrito los números que desea ordenar, se procederá a ingresarlos en un arreglo lineal. Una vez realizado este ingreso y comprobado que el usuario no haya ingresado un valor diferente a decimales, enteros y guiones, el programa limpiará el campo de texto, permitiendo así que el usuario pueda agregar más números a su arreglo. Cuando el usuario decida ordenar los datos, deberá presionar el botón de "ordenar datos". Después de dar la orden de empezar a ordenar, el programa elegirá si procesar el arreglo con el método heapsort, crear un árbol binario o una tabla hash dependiendo los criterios del análisis asintótico. Luego, los números ordenados se verán en un campo de texto donde se mostrará el arreglo ordenado, la estrategia que proceso el problema, el tiempo que se tomó, los recursos de memoria que utilizó y el número de entradas registradas. Finalmente, se exportará un archivo de texto plano con los datos antes mencionados.

7) Alternative 7:

El usuario deberá ingresar los datos que desea ordenar ya sea especificando la ruta de un archivo donde tenga el arreglo de números que desea ordenar (deben estar separados por guiones y los decimales deben usar punto), o escribiéndolos de forma manual en un campo de texto. Si el usuario desea ingresar los datos por medio de un archivo de texto, el programa mostrará un campo de texto donde podrá definir la ruta del archivo. Si el usuario ingreso el arreglo por un archivo de texto, el programa evaluará si no se ha cometido ningún error, tanto en la ruta como en los datos introducidos. Por otro lado, si el usuario lo ingreso manualmente, el programa debe verificar si no hay ningún error en la entrada. A continuación, se lleva a almacenar todos los números en un arreglo lineal. Cuando el usuario decida ordenar los datos, el programa deberá elegir si usar el mergesort, heapsort o quicksort para ordenar el arreglo en base a los criterios del análisis asintótico. Una vez ordenado el arreglo, el programa mostrará en una lista dicho arreglo. Además, mostrará la estrategia que soluciono el problema, el tiempo empleado, los recursos de memoria usados y la cantidad de datos que se evaluaron. Finalmente, el programa le permitirá al usuario exportar un reporte con los datos antes mencionados.

Fase 5. Análisis de cada idea propuesta, resaltando los Pros y Contra de cada solución.

Advantages and Disadvantages:

1) Alternative 1:

Esta alternativa tiene una ventaja principal que tanto muestra los datos principales en el que la aplicación funciona y expone la funcionalidad necesaria para procesar los datos a través de interfaz de usuario, y dependiendo de la entrada que el usuario ingrese, se habilitaran mecanismos como barras de menús, barras de herramientas, etc. Esta alternativa a la hora de ingresar los datos es muy buena ya que por medio de los cuadros emergentes mostrara información específica a los usuarios, recopilar a información de los usuarios, y también puede mostrar y recopilar información. Una desventaja de usar un cuadro emergente es que si la entrada que va a ingresar el usuario no será mostrada correctamente dentro del cuadro emergente. Por otra parte, una ventaja de usar el texto plano como mecanismo de ingreso de datos es que el usuario no tendrá que invertir tiempo a la hora de poner los elementos a ordenar, ya que estos estarán previamente listos en el archivo te texto plano. Una desventaja del archivo de texto plano es que tomara un tiempo estipulado a la hora de leer el archivo. Puede haber errores de lectura que el usuario por no tener conocimiento de cómo deberían de ser ingresados los datos no pueda avanzar.

Una ventaja de usar los métodos de ordenamiento, Merge Sort, Slecction Sort, y Shell Sort es que estos tienen una complejidad de tiempo mucho menor en comparación con otros algoritmos de ordenamiento, lo que lleva a ordenar los elementos en el menor tiempo posible. Una de las desventajas que podemos encontrar en algunos de esos algoritmos es la implementación en la fase del mundo, ya que pueden llegar a ser un poco complejos de procesar en el sistema. A la hora de la salida los elementos serán mostrador por medio de un JList. La ventaja de este mecanismo es que nos permitirá mostrar el resultado de manera eficiente y con entradas grandes.

2) Alternative 2:

En esta alternativa podemos encontrar que el usuario ingresara los datos o elementos por medio de un cuadro de texto, esto le facilitara al usuario en lo relacionado con el tiempo de ingreso, pero tendrá una desventaja la cual es que cuando el usuario deseara ingresar entradas grandes el cuadro de texto no será una buena funcionalidad ya que no le permitirá ver muy bien los elementos ingresados. Por otra parte, cuando hablamos de los métodos que vamos a utilizar, tanto el mergesort, quizksort y heapsort tienen una complejidad temporal muy baja en comparación con otros métodos de ordenamiento, pero una o varias desventajas que podríamos encontrar con este, es a la hora de la complejidad espacial, la cual se basa en la cantidad de memoria que utilizara el programa implementando estos algoritmos. Otra desventaja que podríamos encontrar en los algoritmos de ordenamiento es que a la hora de la implementación puede volverse algo tedioso.

Una de las ventajas de esta alternativa a la hora de mostrar el resultado obtenido es que esta nos mostrará por medio de un cuadro de texto se mostrará el arreglo ordenado y en una lista. Además, mostrará el tiempo de ejecución, el algoritmo utilizado, el número de entradas ingresadas, y la cantidad de memoria utilizada cada una en una etiqueta, esto significa que el usuario tendrá toda la información de lo que está haciendo el programa. Una desventaja seria que a la hora de mostrar entradas grandes no

3) Alternative 3:

En esta alternativa podemos encontrar que el usuario ingresara los datos por medio de un campo emergente. Una de las ventajas que podríamos encontrar a la hora de implementar esta solución es que el usuario podrá ingresar los datos de una manera concisa y sin gasto de tiempo. También, por medio del campo de texto se le hará saber al usuario como ingresar los datos de una manera correcta reduciendo así el nivel de error. Una vez los elementos sean ingresados estos harán el llamado a los métodos MergeSort, o Quick Sort. Estos métodos tienen una ventaja ya que estos son muy eficientes en cuestión de tiempo y retornan una respuesta eficaz, pero a la hora de la implementación se vuelve un poco complicado de tratar. Por otra parte, también tenemos la opción de crear un árbol binario con los elementos ingresados por el usuario, la ventaja de esta opción es que ya nos estaría organizando los elementos en la lista, y esto nos facilitaría a la hora de buscar los elementos en ella. Una ventaja de utilizar un árbol es la facilidad de implementación por el medio recursivo.

Finalmente, los datos ordenados serán exportados como un archivo plano. Una de las ventajas de exportar los elementos como un archivo plano es que el usuario tendrá facilidad de consultar los datos ingresados, el método que lo ordeno, el tiempo que se demoró, la cantidad de entradas que se ingresaron y el total de memoria utilizada, cuantas veces el quiera, ya que estos estarán guardados en el equipo. Una desventaja de esto es que podría tomar un poco de tiempo dependiendo si la entrada fue grande o no.

Alternativa 4:

Pros:

- **Evita que el usuario ingrese datos de forma manual:**
  Ya que el usuario ingresará los datos por medio de un archivo plano, se beneficiará de la función de generación de datos aleatorios que ofrecerá el programa. Al mismo tiempo, el software no tendrá que esperar entradas inesperadas que el usuario quiera ingresar.

- **Permite usar métodos de ordenamiento muy eficientes:**
  Tanto el MergeSort, el Shellort y el HeapSort brindarán tres estrategias que ofrecen un nivel de respuesta muy alto en función del tiempo.

- **Facilita mostrar datos pequeños:**
  Al implementar una JList, los datos pequeños serán muy fáciles de visualizar. El usuario podrá arrastrar el scroll y ver la lista de datos ordenados.

Contras:

- **Implementa algoritmos complicados:**
  ShellSort, HeapSort son algoritmos complicados de entender y, posiblemente, de implementar.

- **Vista para entradas grandes difíciles de mostrar:**
  La JList puede que no soporte entradas grandes de datos y colapse. En caso de que soporte la entrada, el usuario no tendrá claridad en la lista que está viendo pues será una lista vertical inmensa.

- **Complejidad en la sincronización de la función “generar números”:**
  Como la única entrada será dada por un archivo de texto, la función que genera los números debe proveer un archivo que cumpla con los requisitos del proceso de entrada. Esto quiere decir que, el archivo creado por la función “generar números” siempre debe cumplir las especificaciones de aceptación del proceso de ingreso de datos.

Alternativa 5:

Pros:

- **Evita que el usuario ingrese datos de forma manual:**
  Ya que se leerán por medio de un archivo plano los números que el usuario desea ordenar. El mismo puede usar un generador de números aleatorios o bien el ya incluido en el programa para crear dicho archivo y evitar escribirlos manualmente.

- **Múltiples estrategias para procesar el ordenamiento:**
  SelectionSort, ShellSort y una tabla hash brindan un respaldo para los casos en que cierta estrategia resulte ineficiente. Por ejemplo, SelectionSort, brinda una alternativa para entradas pequeñas.

- **Visualización mejor estructurada:**
  Al usar el campo de texto para mostrar la información relevante como el tiempo de ejecución, entre otras, y la lista mostrando el arreglo ordenado, el usuario podrá tener un panorama mejor estructurado del resultado.

Contras:

- **Estructura de datos nueva:**
  La tabla hash da muchas incertidumbres, tanto en su implementación como en su ejecución. Ya que es un tema nuevo, es posible que nos tome mucho tiempo implementarla. Al mismo tiempo, es posible que no de los resultados que necesitemos o haya sido programada mal.

- **Sobre carga de la lista que mostrará los datos:**
  Cómo se mostrará en una JList, en caso de que el usuario escoja un millón de datos, es posible que el componente no lo soporte. O bien el usuario deba usar su rueda del ratón demasiado para poder ver su lista ordenada.

- **Complejidad en la sincronización de la función "generar números":**
  Ya que el programa solo recibirá un archivo, el único método para generarlo rápidamente es el generador de datos. Este debe asegurarse de que el archivo exportado no genere ningún error al momento de ser ingresado de regreso en la entrada.

Alternativa 6:

Pros:

- **Exportación de resultados:**
  El usuario tendrá la posibilidad de guardar un registro con los datos de cada prueba pues, el programa, una vez terminado el ordenamiento, exportará un reporte con los datos que se registraron durante el proceso.

- **Métodos eficientes de procesar la información:**
  Al implementar un árbol binario de búsqueda, una tabla hash y el ordenamiento HeapSort, tendremos tres estrategias interesantes que podrían procesar el ordenamiento de manera eficiente y tal vez complementaria. Pues una podría ayudar a la otra.

- **Visualizaciones estructuradas de los datos:**
  Mostrando los datos relevantes como el tiempo de ejecución, el algoritmo que lo soluciono y los recursos utilizados por medio de etiquetas, serán datos que son fácilmente visibles.

Contras:

- **Estructura de datos nueva:**
  La tabla hash da muchas incertidumbres, tanto en su implementación como en su ejecución. Ya que es un tema nuevo, es posible que nos tome mucho tiempo implementarla. Al mismo tiempo, es posible que no de los resultados que necesitemos o haya sido programada mal.

- **Ordenamiento complejo:**
  Ordenar por el método de HeapSort representa un desafío pues es algo complicado de entender e implementar.

- **Incertidumbre en la visualización de los datos:**
  Es posible que el campo de texto no muestre los datos o que colapse

7) Alternative 7:

En esta alternativa encontramos que el usuario tendrá la opción de ingresar los datos por un archivo plano. Una de las ventajas de ingresar los elementos por un archivo plano es que este ya viene con unos criterios de lectura y el usuario no tendrá que escribirlos en un cuadro de texto, esto reduciría el margen de error a la hora de la lectura de archivos. Por otra parte, en el mundo encontraremos que los métodos, QuickSort, MergeSort y PidgeonHol Sort brindan alternativas distintas que satisfacen las deficiencias de cada uno , complementándose entre sí. Una d elas ventajas que podríamos encontrar a la hora de usar el método QuickSort, es que Requiere de pocos recursos en comparación a otros métodos de ordenamiento, en la mayoría de los casos, se requiere aproximadamente N log N operaciones y este tiene un ciclo interno extremadamente corto.Una de las desventajas que podríamos encontrar al implementar este método es que se complica la implementación si la recursión no es posible, en el peor caso, se requiere N2, un simple error en la implementación puede pasar sin detección, lo que provocaría un rendimiento pésimo y no es útil para aplicaciones de entrada dinámica, donde se requiere reordenar una lista de elementos con nuevos valores. Por otra parte, tenemos el método MergeSort, una de sus varias ventajas es que es estable mientras la función de mezcla sea implementada correctamente, también es muy estable cuando la cantidad de registros a acomodar es de índice bajo, en caso contrario gasta el doble del espacio que ocupan inicialmente los datos. Hablando por otro lado de sus desventajas podríamos encontrar que este está definido recursivamente. Si se deseara implementarla no recursivamente se tendría que emplear una pila y se requeriría un espacio adicional de memoria para almacenarla. Por último, tenemos el método

Fase 6. En este paso procedemos a definir los criterios que nos permitirán evaluar las alternativas de las soluciones anteriormente propuestas y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más

Criterio 1: En el criterio 1 evaluaremos la precisión de la solución que se va a escoger, definiéndola en una escala numérica.

A) La solución es Exacta—Valor: 2
B) La solución es Aproximada –Valor: 1

Criterio 2: En el criterio 2 evaluaremos la facilidad que tendrá el usuario al momento de ingresar los datos.

A) Fácil ----) Valor: 3
B) Media ---) Valor: 2
C) Algo Complicada ---) Valor:1

Criterio 4: En el criterio 4 evaluaremos el tiempo utilizado por cada algoritmo.

A) Constante --) Valor: 1
B) Logarítmica –) Valor: 6
C) Lineal --) Valor: 5
D) Cuadrática --) Valor: 4
E) Cubica --) Valor: 3
F) Exponencial --) Valor: 2

Criterio 5: En el criterio 5 evaluaremos el espacio en memoria utilizado por cada algoritmo.

A) Constante --) Valor: 1
B) Logarítmica –) Valor: 6
C) Lineal --) Valor: 5
D) Cuadrática --) Valor: 4
E) Cubica --) Valor: 3
F) Exponencial --) Valor: 2

Criterio 6: En el criterio 6 evaluaremos la facilidad con la que se muestran los datos en el programa.

A) Fácil ----) Valor: 3
B) Media ---) Valor: 2
C) Algo Complicada ---) Valor:1

Alternatives	Criterio 1	Criterio 2	Criterio 3	Criterio 4	Criterio 5	Criterio 6
	Precisión de la solución	Facilidad de ingreso de datos	Facilidad de implement ación del mundo	Tiempo Utilizado por cada Algoritmo	Espacio de Memoria Utilizado por cada Algoritmo	Facilidad de implement ación del algoritmo
Alternative 1		Cuadro Emergente : 3 Archivos de texto Plano: 2	Medio: 2	Merge Sort: O(n log n): 6 Selecction Sort: O(n²): 4 Shell Sort: O(n1.25): 2	Merge Sort: O(n) : 5 Selecction Sort: O(1): 1 Shell Sort: O(1): 1	Merge Sort: 2 Selecction Sort: 3 Shell Sort: 2
Alternative 2		Campo de texto: 3	Medio 2	Merge Sort: O(n log n): 6 Quizksort: O(n²): 4 Heapsort: O(n log n): 6	Merge Sort: O(n) : 5 Quizksort: O(log n): 6 Heapsort: O(1): 1	Merge Sort: 2 Quizksort: 3 Heapsort: 2
Alternative 3		Campo Emergente : 3	Medio : 2	Merge Sort: O(n log n): 6 Quizksort: O(n²): 4 Arbol Binario: O(log n): 6	Merge Sort: O(n) : 5 Quizksort: O(log n): 6	Merge Sort: 2 Quizksort: 3 ArbolBinar io: 3
Alternative 4		Archivo de texto plano: 2	Medio: 2	Merge Sort: O(n log n): 6 Shell Sort: O(n1.25): 2 Heapsort: O(n log n): 6	Merge Sort: O(n) : 5 Shell Sort: O(1): 1 Heapsort: O(1): 1	Merge Sort: 2 Shell Sort: 2 Heapsort: 2

Alternative 5		**Archivo de texto Plano: 2**	**Medio: 2**	**Selecction Sort: O(n²): 4** **Shell Sort: O(n1.25): 2** **Tablas Hash: O(1): 1**	**Selecction Sort:** O(1): 1 **Shell Sort: O(1): 1** **Tablas Hash: O(c): 1**	**Selecction Sort: 3** **Shell Sort: 2** **Tablas Hash: 1**
Alternative 6		**Campo de texto: 3**	**Medio: 2**	**Arbol Binario: O(log n): 6** **Tablas Hash: O(1): 1** **Heapsort: O(n log n): 6**	**Tablas Hash: O(c): 1** **Heapsort: O(1): 1**	**ArbolBinario: 3** **Tablas Hash: 1** **Heapsort: 2**
Alternative 7		**Archivo de texto plano: 2** **Campo de texto: 3**	**Medio: 2**	**Quizksort: O(n²): 4** **Merge Sort: O(n log n): 6**	**Quizksort: O(log n): 6** **Merge Sort: O(n) : 5**	**Quizksort: 3** **Merge Sort: 2**

Fase 7. Evaluación y selección de la mejor solución

Alternative3:
Categoria2(3+2)+Categoria3(2)+Categroria4(6+4+2)+Categoria5(5+1+1)+Categoria6(2+3+2)+=37

Alternative 4:

Categoría 2(2)+Categoria3(2)+Categoría 4(6+2+6)+ Categoria5(5+1+1)+Categoría 6(2+2+2)+=30

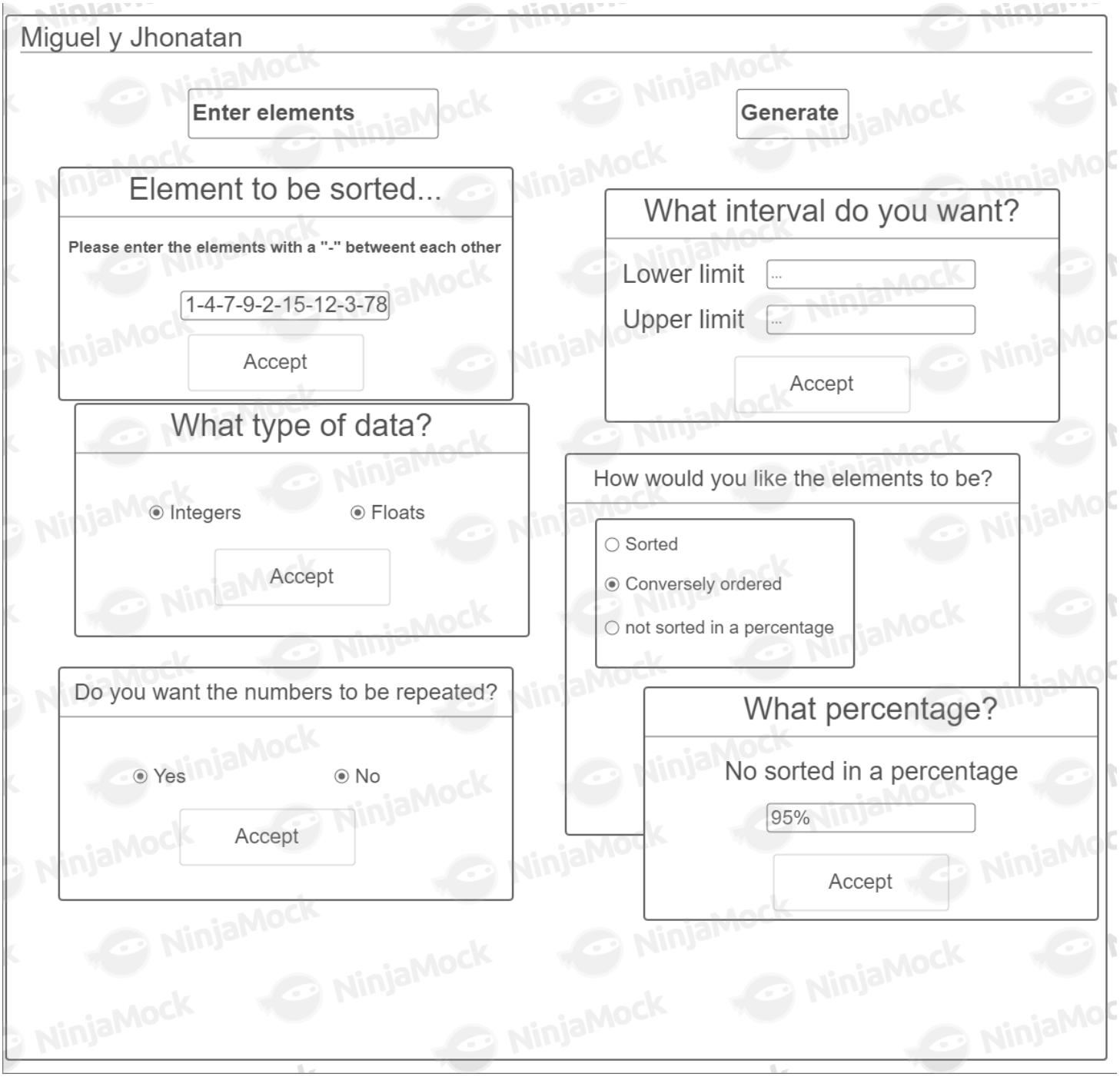
Alternative 7:

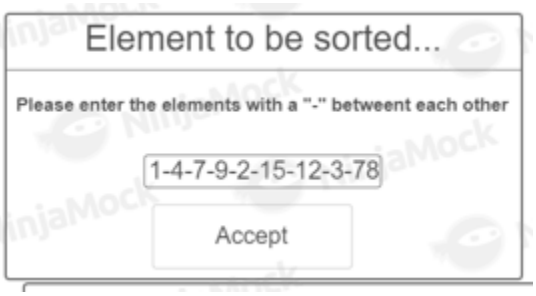
Categoría 2(2+3)+Categoria3(2)+Categoría 4(4+6+6)+ Categoria5(6+5+1)+Categoría 6(3+2+2)+=42

En esta fase lo que procedimos a hacer fue evaluar la mejor solución, teniendo en cuenda la tabla de criterios anteriormente definida y descartar las ideas que no alcanzaran el puntaje promedio para pasar. Al final de la evaluación encontramos que las mejores soluciones son la Alternativa 3, 4 y 7, descartando así las Alternativas 1, 2, 5 y 6. Con base en el puntaje obtenido lograremos escoger la mejor solución entre las tres restantes y así poder implementar con todos sus requerimientos correspondientes.

Fase 8. Diseño preliminar de cada idea no descartada: (Modelos de Simulación)

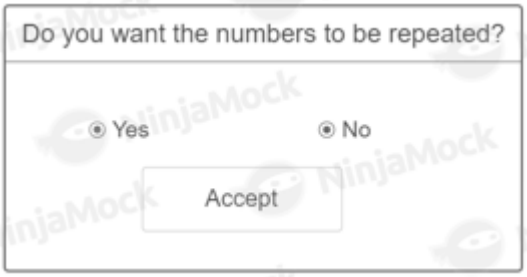
Alternativa 3:





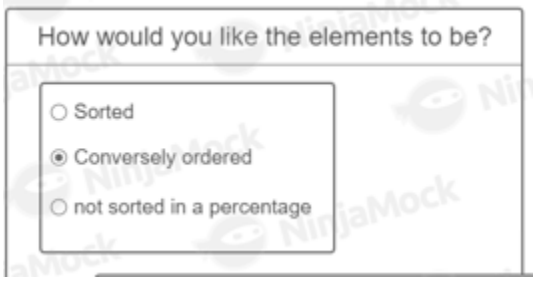
En esta alternativa el usuario tendrá dos opciones, la primera será donde el ingresará los elementos manualmente, en este caso (Enter Elements).Una vez el usuario de click allí, el programa le mostrara una opción en donde el podrá organizarlos, también le explicara como deberán de ser organizados, para que no haya margen de error a la hora de ingreso de elementos.

Por otra parte, cuando el usuario haga click en el botón generar, este le mostrara otra opción en donde el tendrá que escoger que tipo de datos le gustaría ordenar, ya sea Integers o Floats.

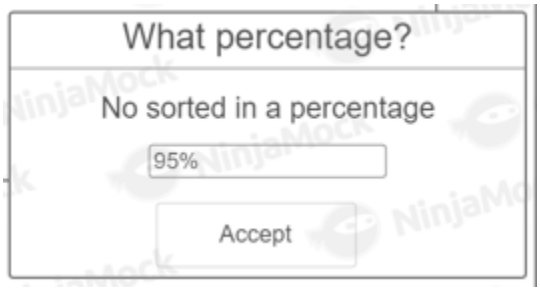


Una vez el usuario decida que tipo de elementos le gustaría generar, aparecerá otra opción por parte del sistema en donde le preguntara al usuario que si le gustaría que los elementos se repitieran.

Cuando el usuario de la opción de Yes, el programa le preguntara cual es el intervalo en que los números a ordenar debería de estar, desde donde debe de empezar (Lower Limit) hasta a donde debería llegar (Upper Limit).



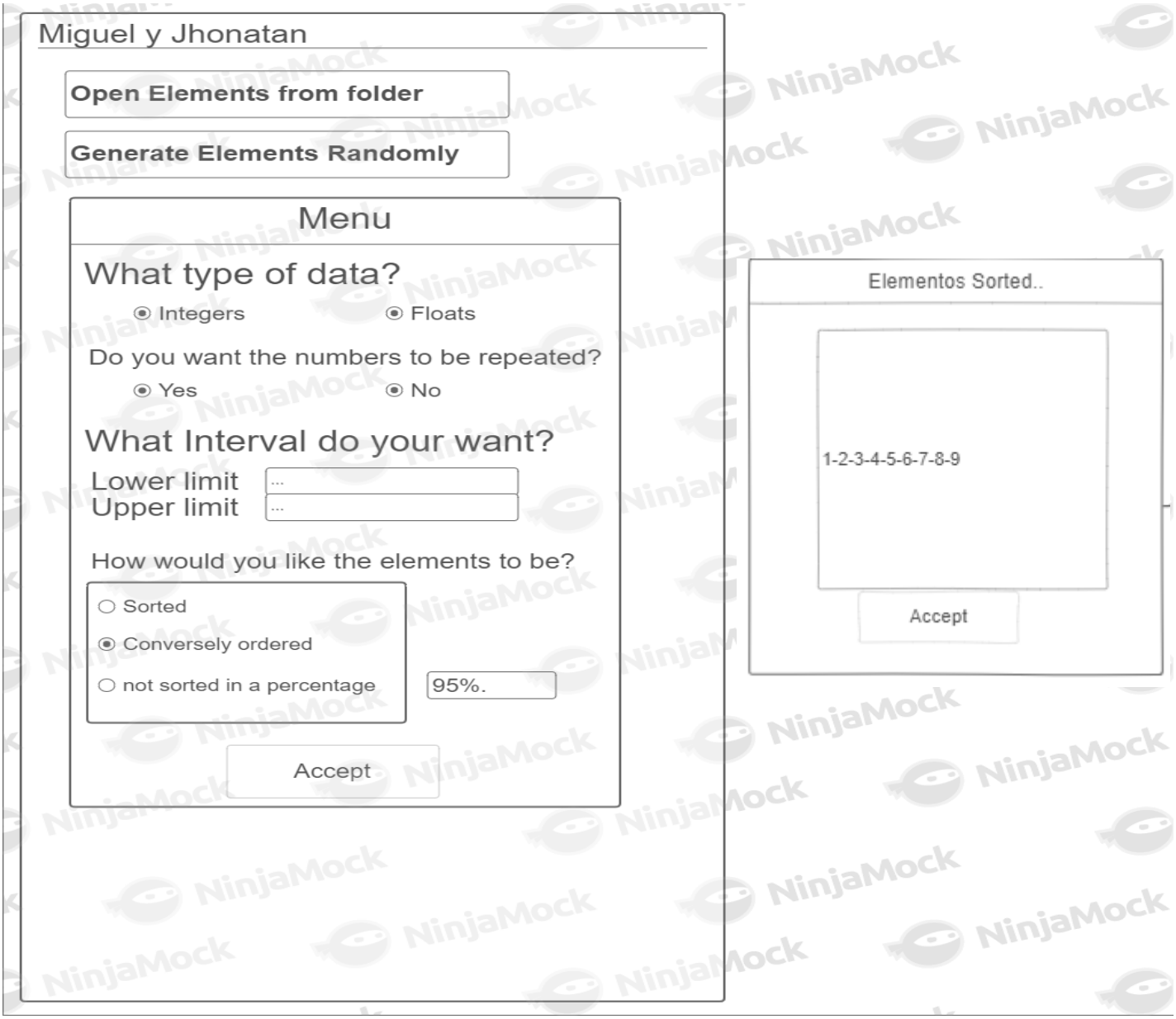
Una vez el usuario haya ingresado tanto el limite inferior como el limite superior, el programa le mostrara una opción en la cual le preguntara al usuario como le gustaría los elementos, ya se (Ordenados, Inversamente Ordenados o un Porcentaje Ordenado)



Si el usuario llegara a escoger la opción de (Valores desordenados en un porcentaje) el programa le mostrara otra opción en donde el usuario tendrá que elegir qué porcentaje de los elementos le gustaría que estuvieran desordenados.

Una vez el usuario haya respondido cada pregunta del programa, este le arrojara por medio de un archivo plano y se mostrara al usuario, los elementos ordenados, separados por un “-“.

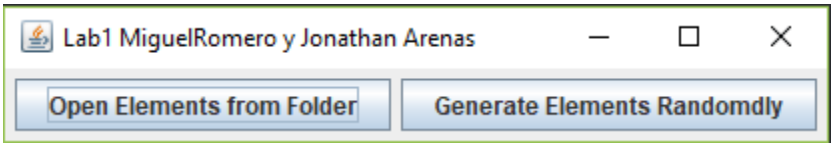
Alternative 4:

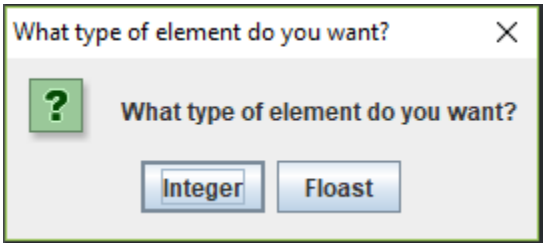


En esta alternativa el usuario tendrá dos opciones, una en la cual el tendrá que ingresar los elementos por medio de un archivo plano. Una vez ingresados los elementos, este los ordenara y serán mostrados por medio de una lista. Por otra parte, el Usuario también contará con una opción en la cual el podrá generar los elementos aleatoriamente, si el usuario escoge esta opción el programa le mostrara un menú, en donde el tendrá que escoger el tipo de dato, si le gustaría que los elementos se repitieran, de donde hasta donde le gustaría que los elementos estuvieran y una última opción en donde el programa le preguntara al usuario como le gustaría que le generara los elementos, ya sea

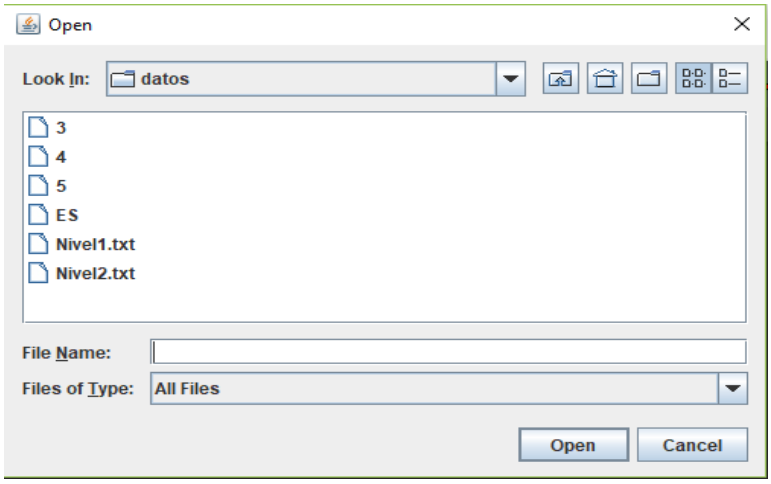
ordenados, inversamente ordenados, y un porcentaje de desordenamiento. Para concluir, los elementos serán mostrados por medio de una lista.

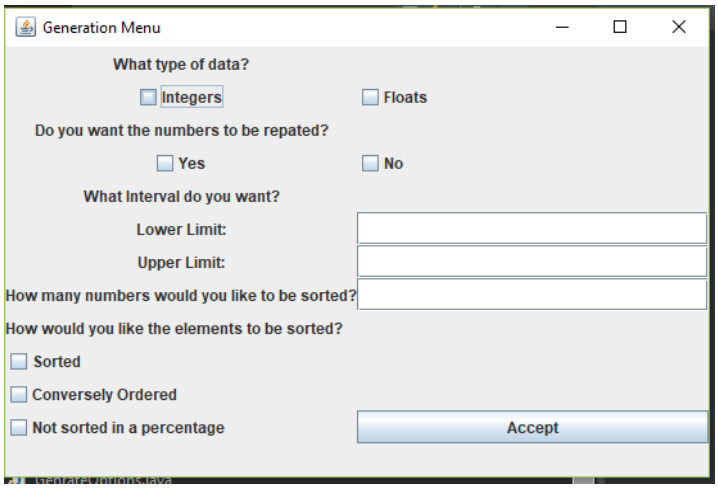
Alternative 7:





En esta alternativa el usuario tendrá dos opciones, la primera en donde el ingresará los elementos por medio de un archivo plano. Esta opción le permitirá al usuario ingresar los elementos y reducir el margen de error. En esta opción por otra parte, el sistema le preguntara al usuario si le gustaría ingresar los elementos, ya sean enteros o Floats. También el usuario tendrá la opción de generar los elementos de una manera aleatoria, y una vez el usuario haga clic en esa opción, el sistema le mostrará un panel con las opciones correspondientes.





En este menú, el sistema le mostrara las opciones en las cuales el usuario podrá escoger el tipo de dato, si le gustaría que el dato se repitiera, que intervalo le gustaría los elementos a ordenar.] (De donde a donde). También, el usuario podrá escoger cuantos números le gustaría ordenar. Por otro lado, el usuario tendrá la oportunidad si generar los elementos ya ordenados, inversamente ordenados y un porcentaje no ordenado. Una vez el usuario haya especificado todos los criterios, el sistema le generara los elementos deseados y se los mostrara por el sistemas.

Fase 8. Preparación de Informes y Especificaciones

Selección

De acuerdo con la evaluación anterior se debe seleccionar la Alternativa 7, ya que obtuvo la mayor puntuación de acuerdo con los criterios definidos. Se debe tener en cuenta que hay que hacer un manejo adecuado del criterio en el cual la alternativa fue peor evaluada que la otra alternativa.

Especificación del Problema (en términos de entrada/salida)

Problema: Ordenamiento de elementos ya sean enteros o de coma flotante.

Entradas:

8.1 Método de Ordenamiento Radix Sort:

En esta fase procedemos a evaluar cada uno de los métodos y a sacarle su respectiva complejidad temporal. Empezaremos por el método RADIXSORT.

Pseudocodigo del Algoritmo de Ordenamiento.	**Complejidad Temporal del Algoritmo**
Funcion arr <- radixSort (arr, n)	
Definir max como Entero	**1**
max <- getMax(arr,n);	**1**
Definir expo como entero	
expo<-1	**1**
Repetir	
countSort(arr,n,expo)	$\log_{10} n$ **x O(n)**
expo<- expo*10	$\log_{10} n$ **x O(n)**
Hasta Que max/expo<0	$\log_{10} n$
Fin Funcion	
Funcion float <- intBitsToFloat (int)	
Fin Funcion	
//Este metodo convierte un float	
//en un entero que representa su estado en bits	
Funcion int <- floatToIntBits(float)	
FinFuncion	
//este metodo busca el número mayor del arreglo	
//sea arr un arreglo de floats y n el numero de entradas	
Funcion max <- getMax (arr, n)	
Definir bitMax Como Entero	**1**

```
        bitMax <- floatToIntBits(arr[0])                                    1
        Para i<-1 Hasta n-1  Con Paso 1  Hacer                              n
                Definir bitTemp como entero                                 n-1
                bitTemp <- floatToIntBits(arr[i])                           n-1
                Si bitTemp> bitMax Entonces                                 n-1
                        bitMax <- bitTemp                                   n-1
                Fin Si
                Fin Para
        max <- bitMax                                                       1
Fin Funcion
//sea arr una lista de floats
//sea n el tamaño del arreglo
//sea exp el exponente que indica que digito se está evaluando
Funcion countSort ( arr, n, expo, )
        Dimension output[n] //arreglo de salida                             1
        Dimension count[10]                                                 1
        Para i<-0 Hasta 9 Con Paso 1 Hacer                                  10
        count[i] = 0                                                        9
        Fin Para
        //almacena la cuenta de ocurrencia en count[]
        Para i<-0 Hasta n-1 Con Paso 1 Hacer                                n
                Definir bitRepre como Entero                                n-1
                bitRepre <- floatToIntBits(arr[i])                          n-1
                count[(bitRepre/expo)%10] <- count[(bitRepre/expo)%10]+1    n-1
                Fin Para
        //ahora se cambian lo que hay en count[i] para tener
```

```
//la posicion de los digitos en el output[]
Para i<-1 Hasta 9 Con Paso 1 Hacer                                  10
        count[i] <- count[i] + count[i-1]                           9
Fin Para
//construir el arreglo de salida output[]
//con los enteros que representan al float en forma bit
Definir i como entero
i <- n-1                                                            1
Mientras i>=0 Hacer                                                 n
        Definir bitRepre como Entero                                n-1
        bitRepre <- floatToIntBits(arr[i])                          n-1
        output[count[(bitRepre/expo)%10]-1] <- bitRepre             n-1
        count[(bitRepre/expo)%10]<- count[(bitRepre/expo)%10]-1     n-1
        i<- i-1v                                                    n-1
Fin Mientras
//copiar el arreglo output[] al arr[],
//hasta este punto, el arreglo tiene ordenado los
//floats depediendo su digito en representacion bit
//lo que se hace es convertir lo que esta en el output[]
//al float
Para i<-0 Hasta n-1 Con Paso 1 Hacer                                n
        arr[i] <- intBitsToFloat(output[i])                         n-1
Fin Para

    Fin Funcion
```

8.2 Complejidad del Algoritmo de Ordenamiento Radix

La complejidad del algortimo Radix es O(n) y lo comprobamos al hacer la suma de cada una de las filas corresponientes.

47+$\log_{10} n$ x O(n)+ $\log_{10} n$ x O(n)+ $\log_{10} n$+ 13(n-1)+4n= O(N)

8.3 Complejidad Espacial del Algoritmo de Ordenamiento Radix.

Adaptado de :

https://www.geeksforgeeks.org/radix-sort/

https://stackoverflow.com/questions/42303108/how-can-i-use-radix-sort-for-an-array-of-float-numbers

https://stackoverflow.com/questions/14619653/how-to-convert-a-float-into-a-byte-array-and-vice-versa/14619742

Consideraciones

Se deben tener en cuenta los siguientes casos para calcular el valor de las raíces:

1. La función tiene dos soluciones reales, que ocurre cuando la parábola descrita por la función cuadrática corta el eje X en dos puntos diferentes. En este caso el discriminante es positivo.

2. La función tiene una solución real, que ocurre cuando la parábola es tangente al eje X. Aquí el discriminante es cero.

3. No hay solución real. La parábola no corta el eje X. El discriminante es menor que cero, y por tanto la

función puede tener una o dos soluciones complejas.