Identificación del problema

Se requiere Implementar varios algoritmos de ordenamiento, como instrucciones básicas para un coprocesador matemático, donde los algoritmos diferentes de ordenamiento permitan ordenar, muy rápidamente, números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.

Se debe dar una implementación del prototipo de pruebas en software de los algoritmos que finalmente serán implementados como operaciones nativas en hardware.

(1) una interfaz gráfica de usuario que le permita ingresar los valores a ordenar

(2) una interfaz gráfica que le permita generar aleatoriamente los valores (tanto enteros como de coma flotante) permitiendo configurar la cantidad total de números a generar.

(3) ordenar los valores ingresados o generados utilizando el algoritmo apropiado y presentar el tiempo que toma el ordenamiento. De acuerdo con el tipo de número a ordenar y el intervalo de números generados, el programa debe restringir/permitir los algoritmos para ordenar.

Recopilación de información

Definiciones:

Algoritmo de Ordenamiento:
Es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación —o reordenamiento— de la entrada que satisfaga la relación de orden dada. Las relaciones de orden más usadas son el orden numérico y el orden lexicográfico.

Complejidad temporal
Se denomina complejidad temporal a la función T(n) que mide el número de instrucciones realizadas por el algoritmo para procesar los n elementos de entrada. Cada instrucción tiene asociado un costo temporal.

Complejidad espacial
La misma idea que se utiliza para medir la complejidad en tiempo de un algoritmo se utiliza para medir su complejidad en espacio. Decir que un programa es O(n) en espacio significa que sus requerimientos de memoria aumentan proporcionalmente con el tamaño del problema. Esto es, si el problema se duplica, se necesita el doble de memoria.

Necesidades que solucionar en el problema

Requerimientos Funcionales

R1	Ordenar valores ingresados.
Resumen:	Ordena los valores de tipo entero o de coma flotante ingresados por el usuario.
Entradas:	Valores ingresados.
Retornos:	Valores ordenados.

R2	Ordenar valores generados aleatoriamente.
Resumen:	Ordena los valores de tipo entero o de coma flotante generados por el sistema.
Entradas:	Valores generados.
Retornos:	Valores ordenados.

R3	Permitir el ingreso de valores a ordenar.
Resumen:	Permite ingresar valores de tipo entero o de coma flotante.
Entradas:	Valores a ordenar.
Retornos:	

R4	Generar valores aleatoriamente.
Resumen:	Permitir genera aleatoriamente valores a ordenar.
Entradas:	
Retornos:	Valores generados aleatoriamente.

R5	Permitir escoger la cantidad de valores a ordenar.
Resumen:	Permite que el usuario elija la cantidad de valores que se generen.
Entradas:	Cantidad de valores.
Retornos:	

R6	Permitir elegir si se repite el valor.
Resumen:	Permite elegir si en el reglo de valores, puede haber valores repetidos o no.
Entradas:	
Retornos:	

R7	Generar aleatoriamente con valores ordenados.
Resumen:	Permite generar aleatoriamente valores ordenados.
Entradas:	
Retornos:	Valores ordenados generados aleatoriamente.

R8	Generar aleatoriamente con valores ordenados inversamente.
Resumen:	Permite generar aleatoriamente valores ordenados de manera inversa.
Entradas:	
Retornos:	Valores ordenados inversamente generados aleatoriamente.

R9	Generar aleatoriamente valores con un % de ordenación.
Resumen:	Permite generar aleatoriamente valores con un % de orden.
Entradas:	
Retornos:	Valores con un % de orden generado aleatoriamente.

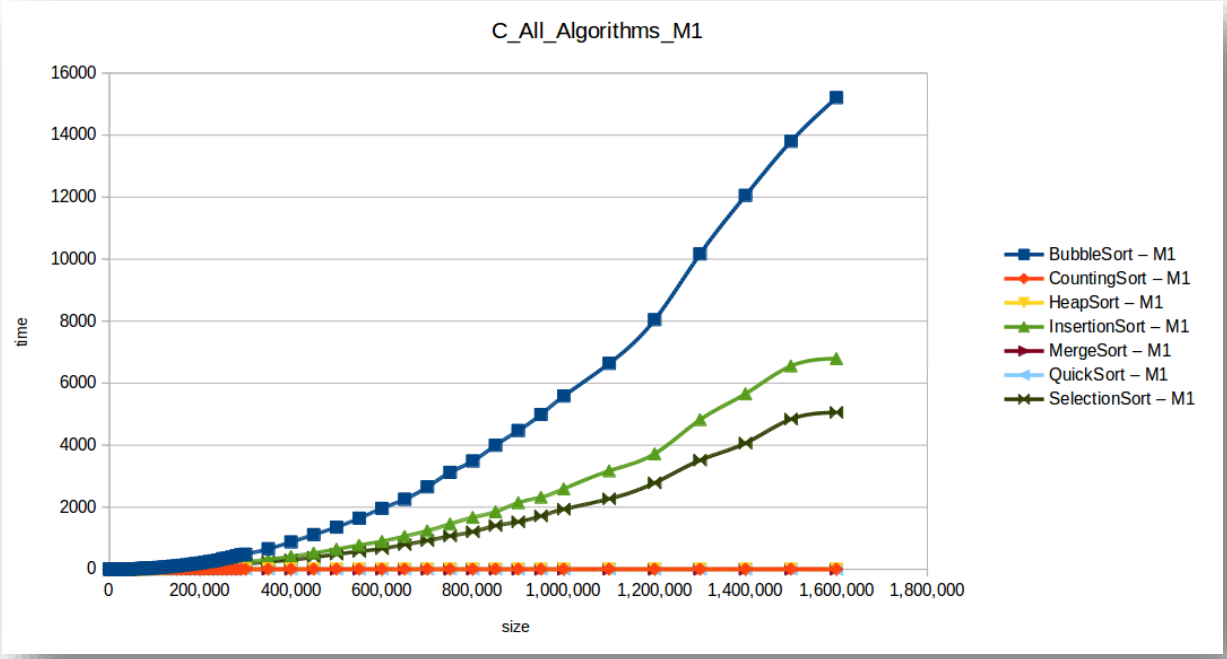
R10	Tiempo que toma el ordenamiento.
Resumen:	Permite conocer el tiempo que toma el ordenamiento de los valores.
Entradas:	
Retornos:	Tiempo que toma el ordenamiento.

R11	Restringir/permitir algoritmos para ordenar
Resumen:	Permite o restringe los algoritmos a ordenar.
Entradas:	
Retornos:	

Búsqueda de soluciones

Para la solución se trató de plantear que se debe usar los algoritmos los cuales sean más eficiente, cuantos menos recursos computacionales consuma: tiempo y espacio de memoria requerida para su ejecución. Las razones, teniendo en cuanta a tomar alternativas acertadas para la solución de problema se opta por tener en cuenta los factores de:

- La complejidad en tiempo del algoritmo. Dado que establece su comportamiento cuanto al número de datos a procesar sea grande.
- La complejidad en espacio del algoritmo. Sólo cuando esta complejidad resulta razonable es posible utilizar este algoritmo con seguridad. Si las necesidades de memoria del algoritmo crecen considerablemente con el tamaño del problema el rango de utilidad del algoritmo es baja y se debe descartar.
- La dificultad de implementar el algoritmo. En algunos casos el algoritmo óptimo puede resultar tan difícil de implementar que no se justifique desarrollarlo para la aplicación que se le va a dar.
- El tamaño del problema que se va a resolver. Un algoritmo para un proceso tamaño pequeño no justifica la realización de un análisis, ya que da lo mismo una implementación que otra.



Los algoritmos de ordenamiento:

Heapsort

Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en un montículo (heap), y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado. Basa su funcionamiento en una propiedad de los montículos, por la cual, la cima contiene siempre el menor elemento (o el mayor, según se haya definido el montículo) de todos los almacenados en él.

MergeSort

En ciencias de la computación, merge sort (también comúnmente llamado mergesort) es un algoritmo de clasificación basado en comparación, eficaz y de propósito general. La mayoría de las implementaciones producen un tipo estable, lo que significa que la implementación conserva el orden de entrada de elementos iguales en la salida ordenada. Merge sort es un algoritmo de divide y vencerás.

Quicksort

Es un algoritmo de clasificación eficiente, que sirve como un método sistemático para colocar los elementos de una matriz en orden. siendo un algoritmo comúnmente utilizado para la clasificación

Quicksort es un tipo de comparación, lo que significa que puede ordenar elementos de cualquier tipo para los cuales se define una relación "menor que" (formalmente, una orden total)- Al igual que Merge Sort, Quicksort es un algoritmo Divide y Conquista. Selecciona un elemento como pivote y divide el conjunto dado alrededor del pivote elegido. Hay muchas versiones diferentes de Quicksort que eligen pivotar de diferentes maneras.

SelectionSort
Conocido por su simplicidad, el algoritmo de ordenación de selección comienza por encontrar el valor mínimo en la matriz y moverlo a la primera posición. Este paso se repite para el segundo valor más bajo, luego el tercero, y así sucesivamente hasta que se ordena el conjunto.

BubbleSort
a veces denominado clasificación de hundimiento, es un algoritmo de clasificación simple que recorre repetidamente la lista para ser ordenada, compara cada par de elementos adyacentes y los intercambia si están en el orden incorrecto. El pase a través de la lista se repite hasta que no se necesiten intercambios, lo que indica que la lista está ordenada. El algoritmo, que es una clasificación de comparación, se llama así por la forma en que los elementos más pequeños o más grandes "burbujean" en la parte superior de la lista.

Counting Sort
El ordenamiento de conteo es un algoritmo para clasificar una colección de objetos de acuerdo con claves que son enteros pequeños; es decir, es un algoritmo de clasificación de enteros. Funciona contando el número de objetos que tienen cada valor de clave distinto y usando la aritmética en esos recuentos para determinar las posiciones de cada valor clave en la secuencia de salida. Su tiempo de ejecución es lineal en el número de elementos y la diferencia entre los valores de clave máximo y mínimo, por lo que solo es adecuado para uso directo en situaciones donde la variación en las claves no es significativamente mayor que la cantidad de elementos.

InsertionSort
El género de inserción es un algoritmo de clasificación en el que los elementos se transfieren uno a la vez a la posición correcta. En otras palabras, una ordenación de inserción ayuda a construir la lista ordenada final, un elemento a la vez, con el movimiento de elementos de mayor clasificación. Un tipo de inserción tiene los beneficios de la simplicidad y la baja sobrecarga.

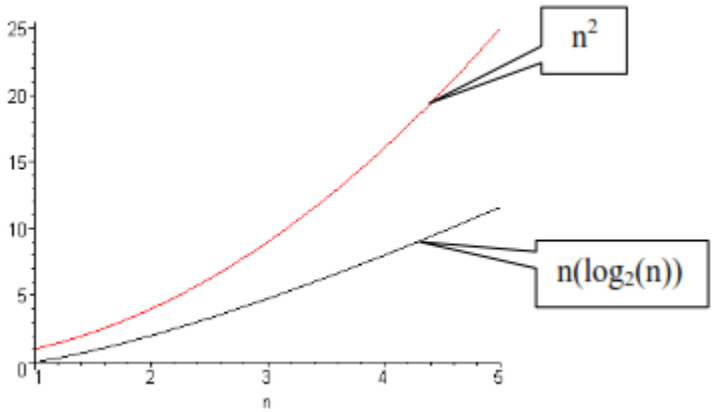
Transición de la formulación de ideas a los diseños preliminares

De todos los distintos métodos de ordenamiento se decide por optar los mejores en tiempo de ejecución que como muestra la gráfica Como se puede observar en la comparativa, no podemos conocer a ciencia cierta quién fue el ganador, dado que hay 4 algoritmos que se solapan entre si debido a la escala de la gráfica, y corresponden al Quicksort, MergeSort, HeapSort y CountingSort.

También se tiene en cuenta la complejidad de cada uno de los algoritmos que nos confirma la mejor opción de solución para el problema.

Estables				
Nombre traducido	Nombre original	Complejidad	Memoria	Método
Ordenamiento de burbuja	Bubblesort	O(n^2)	O(1)	Intercambio
Ordenamiento de burbuja bidireccional	Cocktail sort	O(n^2)	O(1)	Intercambio
Ordenamiento por inserción	Insertion sort	O(n^2)("(en el peor de los casos)")	O(1)	Inserción
Ordenamiento por casilleros	Bucket sort	O(n)	O(n)	No comparativo
Ordenamiento por cuentas	Counting sort	O(n+k)	O(n+k)	No comparativo
Ordenamiento por mezcla	Merge sort	O(n log n)	O(n)	Mezcla
Ordenamiento con árbol binario	Binary tree sort	O(n log n)	O(n)	Inserción
	Pigeonhole sort	O(n+k)	O(k)	
Ordenamiento Radix	Radix sort	O(nk)	O(n)	No comparativo
	Distribution sort	O(n^k) versión recursiva	O(n^2)	
	Gnome sort	O(n^2)	O(1)	
Inestables				
Nombre traducido	Nombre original	Complejidad	Memoria	Método
Ordenamiento Shell	Shell sort	O($n^{1.25}$)	O(1)	Inserción
	Comb sort	O(n log n)	O(1)	Intercambio
Ordenamiento por selección	Selection sort	O(n^2)	O(1)	Selección
Ordenamiento por montículos	Heapsort	O(n log n)	O(1)	Selección
	Smoothsort	O(n log n)	O(1)	Selección
Ordenamiento rápido	Quicksort	Promedio: O(n log n), peor caso: O(n^2)	O(log n)	Partición
	Several Unique Sort	Promedio: O(n u), peor caso: O(n^2);		

La comparación de los algoritmos de ordenamiento se hace a base de su complejidad entonces



Como observamos en la gráfica la complejidad n(log(n)) es más eficiente que la complejidad n^2 por que requiere menos tiempo para valores más grandes, haciendo más eficiente el algoritmo.

Entonces conociendo esto se descarta los algoritmos de *InsertionSort, BubbleSort, SelectionSort*

Evaluación y selección de la mejor opción

Dado a lo observado en las tablas ya podemos tener una clara idea de los algoritmos a utilizar para la solución, Los algoritmos veloces, que son el Quicksort, MergeSort y CountingSort los algoritmos mas eficientes a utilizar en la solución del problema dado que definimos la solución en el criterio en la eficiencia de cada algoritmo y como denotamos el problema de eficiencia de un programa se puede plantear como un compromiso entre el tiempo y el espacio utilizado. Otro factor es que la solución planteada sea factible para la solución del problema dado que en algunos casos los algoritmos son mas eficientes para diferentes tipos de datos y en nuestro caso se manejaran dos tipos de datos los cuales son:

- Valores de tipo entero
- Valores de tipo coma flotante

Y como se observa en las gráficas el algoritmo más factible es el CountingSort

El algoritmo CountingSort con una complejidad algorítmica de O(n+k), siendo n la cantidad de datos a ordenar y k el tamaño del vector auxiliar (máximo - mínimo). Pero el algoritmo CountingSort, tiene una gran limitación, la cual es que solo funciona con números enteros, dado que requiere de un vector auxiliar donde almacenar la cuenta de cada valor;

El algoritmo Quicksort con una complejidad de O (n log n), siendo uno de los más usados con el cual puedes ordenar también números reales; pero que igual tiene sus falencias, dado que es un algoritmo probabilístico y hay casos extremos en los cuales la complejidad se puede elevar a O(n^2), dependiendo de la distribución de los datos y una mala elección de pivote.

Se debe realizar la implementación tomando en cuenta estas restricciones en el problema para una solución óptima acorde a las necesidades del usuario

Preparación de informes y especificaciones

La solución esta basada en que dado una cadena de valores ya sea de tipo entero o de coma flotante el programa sea capaz de ordenar la cadena de valores de la manera más optima con los algoritmos de ordenamiento implementados.

Problema: Ordenamiento de valores enteros o de coma flotante. Indicando las restricciones especificadas por parte del usuario.

Entrada: Cadena de valores ingresados o generados por parte del usuario. Cadena {a1, a2, a3, a4, a5, …, an}.

Salida: Cadena de valores ordenada. Cadena {a1<a2<a3<a4<a5<…<an}.

Consideraciones

- Cuando la cadena es de tipo combinado de valores enteros y de coma flotante. Se utiliza el algoritmo que mejor se adapte a la solución óptima.

Implementación

Implementación en el IDE de NetBeans 8.2 con javaFx