Angie Valentina Córdoba Pinzón-- A00347829(GRUPO 5)

Alejandra Gonzalez Velez--A00346154(GRUPO 5)

Jair Aguirre García --A00346739 (GRUPO 1)

IDENTIFICACIÓN DEL PROBLEMA

Identificación de necesidades y síntomas.

-La empresa requiere de algoritmos de ordenamiento para números enteros de gran tamaño arbitrario y en forma de coma flotante.

-Los algoritmos de ordenamiento utilizados deben ser eficientes.

-La solución al problema debe garantizar que el usuario use algunas configuraciones antes de generar los números.

-La empresa no cuenta con ideas de qué algoritmos son los más eficientes en ordenamiento.

-La solución al problema debe presentar el tiempo que toma el ordenamiento a cada algoritmo utilizado.

Definición del problema: La empresa de fabricación requiere algoritmos de ordenamiento altamente eficientes para su próximo coprocesador matemático, que permitan generar números enteros o en coma flotante de forma arbitraria.

RECOPILACIÓN DE INFORMACIÓN

Definiciones

Coma flotante:

La representación de 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. (Fuente: https://es.wikipedia.org/wiki/Coma_flotante#Ejemplo)

Algoritmos de ordenamiento: Es la operación de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento. (fuente: http://lwh.free.fr/pages/algo/tri/tri_es.htm)

Métodos de Ordenamiento Elementales:

a) Inserción

b) Selección

c) Burbujeo

Clase random:

Una instancia de esta clase se usa para generar una secuencia de números pseudoaleatorios. La clase usa una semilla de 48 bits, que se modifica usando una fórmula congruente lineal. (Fuente: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html)

SOLUCIONES CREATIVAS

Métodos de Ordenamiento más Eficientes:

- ***Alternativa 1: QuickSort (Ordenamiento Rápido):***

  Es el algoritmo de ordenamiento más eficiente de todos, se basa en la técnica de "Divide y Vencerás", que permite en promedio, ordenar n elementos en un tiempo proporcional a n*log(n).
  Algoritmo Fundamental:

  a)Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
  b) Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
  c) La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
  d) Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

- **Alternativa 2: Heap Sort (Ordenamiento por Montículos):**

  Es un algoritmo de ordenamiento no recursivo, no estable, 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.

- **Alternativa 3: MergeSort (Ordenamiento por Combinación):**

  El método MergeSort es un algoritmo de ordenación recursivo con un número de comparaciones entre elementos del array mínimo.Su funcionamiento es similar al Quicksort, y está basado en la técnica divide y vencerás. De forma resumida el funcionamiento del método MergeSort es el siguiente:

  a) Si la longitud del array es menor o igual a 1 entonces ya está ordenado.
  b) El array a ordenar se divide en dos mitades de tamaño similar.
  c) Cada mitad se ordena de forma recursiva aplicando el método MergeSort.

d) A continuación las dos mitades ya ordenadas se mezclan formando una secuencia ordenada.

(Fuente: https://gl-epn-programacion-ii.blogspot.com/2010/06/metodos-de-ordenamiento.html

http://puntocomnoesunlenguaje.blogspot.com/2014/10/java-mergesort.html)

Otros Métodos de ordenamiento:

- **Alternativa 4: Método de inserción directa (Insertion sort):**

  Este algoritmo es muy elemental, antiguo y lento, aunque su utilización está justificada en problemas pequeños debido a su bajo consumo de recursos. Este método toma cada elemento del arreglo para ser ordenado y lo compara con los que se encuentran en posiciones anteriores a la de él dentro del arreglo. Si resulta que el elemento con el que se está comparando es mayor que el elemento que ordenar, se recorre hacia la siguiente posición superior. Si por el contrario, resulta que el elemento con el que se está comparando es menor que el elemento que ordenar, se detiene el proceso de comparación, pues se encontró que el elemento ya está ordenado y se coloca en su posición (que es la siguiente a la del último número con el que se comparó).

- **Alternativa 5: Método de selección (Selection sort):**

  Este método también es muy lento, sin embargo, por su estructura, reduce al mínimo el número de intercambios. Consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo.

  (Fuente: https://techlandia.com/ventajas-desventajas-algoritmos-ordenamiento-info_181515/)

- **Alternativa 6: Método de Shell (Shell sort):**

  El método de ordenación Shell, así llamado en honor a su inventor Donald Shell, es considerado un algoritmo avanzado cuya velocidad es notablemente mayor con respecto a los anteriores. Este método utiliza una segmentación entre los datos; funciona comparando elementos que están distantes. La distancia entre comparaciones decrece conforme el algoritmo se ejecuta hasta la última fase, en la cual se comparan los elementos adyacentes, por esta razón se le llama también ordenación por disminución de incrementos.

  (Fuente: http://www.teknoplof.com/2010/03/25/algoritmos-de-ordenamiento/)

Alternativa 7: Comb Sort

Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all inversions are removed one by one. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a

factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one inversion counts with one swap and performs better than Bublle Sort.

The shrink factor has been empirically found to be 1.3 (by testing Combsort on over 200,000 random lists)

(fuente: https://www.geeksforgeeks.org/comb-sort/)

TRANSICIÓN DE LAS IDEAS A LOS DISEÑOS PRELIMINARES

Se Descartó la alternativa 1 (QuickSort), la alternativa 3 (Mergesort) y la alternativa 2 (Heap Sort) debido a que el planteamiento del problema limita a hacer uso de algoritmos recursivos.

La revisión cuidadosa de otras alternativas nos conduce a:

Alternativa 4: Método de inserción directa.

- Es un algoritmo de ordenamiento relativamente simple.
- Exhibe un buen rendimiento cuando se trabaja con una lista pequeña.
- Requiere espacio mínimo.
- No es muy eficiente con una lista grande de elementos.

Alternativa 5: Método de selección.

-Es un algoritmo afectivo en ordenamiento de datos con un tamaño pequeño.

-Tiene un bajo consumo de recursos mientras la lista de datos sea pequeña.

- Es un algoritmo simple.

Alternativa 6: Método shell Sort.

- Es uno de los algoritmos de ordenamiento más veloces.
- Realiza numerosas comparaciones e intercambios.
- Es efectivo en el manejo de lista de datos de gran tamaño.

Alternativa 7: Método Comb Sort

-Es un algoritmo tan veloz como el QuickSort

-Es efectivo en el manejo de listas de datos de gran tamaño.

- Es tan eficiente como el método Shell Sort.

-Es un algoritmo relativamente simple.

EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN.

Para la evaluación y selección se tomarán las tres mejores alternativas, es decir los algoritmos de ordenamiento que más se ajusten a la solución del problema en cuestión.

Criterios:

Criterio A: La alternativa entrega una solución:

- [1] Parcial.
- [2] Exacta.

Criterio B: El número de soluciones que la alternativa entrega son: .

- [3] Todas.
- [2] Más de una, pero no todas.
- [1] Una.

Criterio C: La alternativa es fácil de implementar.

- [2] Compatible con todas las operaciones de cómputo moderno.
- [1] Requiere operaciones que ya están obsoletas.

Criterio D: La alternativa es eficiente a un nivel:

- [4] Constante.
- [3] Mayor a constante.
- [2] Logarítmica
- [1] Lineal.

Evaluación de alternativas

	Criterio A	Criterio B	Criterio C	Criterio D	Total
Alternativa 7	Exacta 2	Todas 3	Compatible 2	Mayor a constante. 3	10
Alternativa 4	parcial 1	Más de una, pero no todas. 2	Compatible 2	Mayor a constante. 3	8
Alternativa 5	Parcial 1	una 1	Compatible 2	Mayor a constante. 3	7
Alternativa 6	Exacta 2	Todas 3	Compatible 2	Logarítmica 2	9

Selección

-De acuerdo con la evaluación anterior las alternativas que más se ajustan a la solución del problema en cuestión son las alternativas 7 y 6, que son las de mayor puntuación, 10 y 9 respectivamente, y la alternativa 4 con la tercera mejor puntuación. Se escogieron tres alternativas porque así lo amerita el problema.

Preparación de informes y especificaciones

Especificación del problema

Problema: Ordenar aleatoriamente un arreglo de enteros o en coma flotante.

Entradas: Un arreglo de tamaño arbitrario.

A = { A_i , $A_{(i+1)}$, $A_{(i+2)}$, ..., $A_{(i+n)}$ } donde $A_i \in$ **Q(Racionales)**

Salida: El arreglo de números debidamente ordenado

B = { A_i , $A_{(i+1)}$, $A_{(i+2)}$, ..., $A_{(i+n)}$ } $\forall \; A_i \leq A_{(i+1)}$

Pseudocódigo del algoritmo shell Sort (alternativa 6)

```
for s  t to 1 by – 1 {s es el índice del incremento o distancia}
        h  dist[s] {h es el incremento o distancia entre elementos a comparar}
 for j  h + 1 to n {j empieza en el segundo elemento del arreglo original}
        KEY  L[j]; i  j – h
         while i>0 and L[i]>KEY do
                L[i + h]  L[i]; i  j – h
        end
        L[i + h]  KEY
    end
end
```

Pseudocódigo del algoritmo CombSort (alternativa 7)

```
1. PROCEDIMIENTO comb_sort ( VECTOR a[1:n])
2. gap ← n
3. REPETIR
4.     permut ← FALSO
5.     gap ← gap / 1.3
6.     SI gap < 1 ENTONCES gap ← 1
7.     PARA i VARIANDO DE 1 HASTA n CON INCRECREMENT DE gap HACER
8.         SI a[i] > a[i+gap] ENTONCES
9.             intercambiar a[i] Y a[i+gap]
10.                permut ← VERDADERO
11.            FIN SI
12.        FIN POUR
13.    MIENTRAS QUE permut = VERDADERO O gap > 1
```

Pseudocódigo del algoritmo insertionSort (alternativa 4)

```
1. PROCEDIMIENTO Insertion_sort ( Vector a[1:n])
2.     PARA i VARIANDO DE 2 HASTA n HACER
3.         INSERTAR a[i] EN SU LUGAR EN a[1:i-1];
4. FIN PROCEDIMIENTO;
```

Implementación del diseño

implementación en lenguaje de programación.

Lista de tareas a implementar:

A. Generar aleatoriamente y Ordenar inversamente los valores .
B. Generar y Ordenar los valores de forma aleatoria.
C. Ordenar los valores sólo en un porcentaje.
D. Validar si un valor está repetido.
E. Calcular el tiempo en ordenar los valores.
F. Método controlador .(generateRandom)
G. Generar valores aleatoriamente.
H. Ordenar valores.

Ordenar inversamente los valores

Nombre	investArray
Descripción	Ordena de forma decreciente el arreglo ingresado por parámetro.
Entrada	arrayToSort: double, arreglo desordenado.
Retorno	Sin retorno

Código:

```
public void investArray(double[] arrayToSort) {
        double aux;
         for (int i=0; i<arrayToSort.length/2; i++) {
                aux = arrayToSort[i];
                arrayToSort[i] = arrayToSort[arrayToSort.length-1-i];
                arrayToSort[arrayToSort.length-1-i] = aux;
              }
}
```

Generar y Ordenar valores aleatoriamente

Nombre	generateRandomOption2
Descripción	Se encarga de generar números aleatoriamente y ordenarlos
Entrada	quantity: int, tamaño de datos rank1: double, inicio del intervalo. rank2: fin del intervalo. option: int, opcion elgida por el usuario.
Retorno	Sin retorno

Código:

```
public void generateRandomOption2(int quantity, double rank1, double rank2, int option) {
                if(option == 0) {
```

```
                for(int i=0; i<quantity; i++) {
                 double random = Math.random()*(rank2-rank1)+rank1;
                 array[i] = random;
                }
                if(quantity <= 500) {
                        insertionSort(array);
                        } else if(quantity > 500 && quantity <= 100000) {
                        combSort(array);
                } else {
                        shellSort(array);
                }
        }
```

Generar aleatoriamente y Ordenar inversamente los valores

Nombre	generateRandomOption3
Descripción	Se encarga de generar números aleatoriamente y ordenarlos inversamente.
Entrada	quantity: int, tamaño de datos rank1: double, inicio del intervalo. rank2: fin del intervalo. option: int, opcion elgida por el usuario.
Retorno	Sin retorno

código:

```
        public void generateRandomOption3(int quantity, double rank1, double rank2, int option)
{
                if(option == 0) {
                for(int i=0; i<quantity; i++) {
                        double random = Math.random()*(rank2-rank1)+rank1;
```

```
                array[i] = random;
                } if(quantity <= 500) {
                insertionSort(array);
                } else if(quantity > 500 && quantity <= 100000) {
                combSort(array);
                        } else {
                shellSort(array);
                }
                investArray(array);
        } else {
                for(int i=0; i<quantity; i++) {
                 double random = Math.random()*(rank2-rank1)+rank1;
                 array[i] = random;
                        }checkRepetition(quantity, array, rank1, rank2);
                if(quantity <= 500) {
                insertionSort(array);
                } else if(quantity > 500 && quantity <= 100000) {
                combSort(array);
                } else {
                shellSort(array);
        }       investArray(array);
        }
}
```

Ordenar valores por un porcentaje dado

Nombre	generateRandomOption4
Descripción	Se encarga de generar números aleatoriamente y ordenarlos inversamente.

Entrada	quantity: int, tamaño de datos rank1: double, inicio del intervalo. rank2: fin del intervalo. option: int, opcion elgida por el usuario. porcetage: double, porcentaje indicado por el usuario.
Retorno	Sin retorno

```
public void generateRandomOption4(int quantity, double rank1, double rank2, int option, double
percentage) {
                if(option == 0) {
                        for(int i=0; i<quantity; i++) {
                         double random = Math.random()*(rank2-rank1)+rank1;
                                 array[i] = random;
                        }if(quantity <= 500) {
                        insertionSort(array);
                        } else if(quantity > 500 && quantity <= 100000) {
                        combSort(array);
                                } else {
                                shellSort(array);
                        }
                        int k = (int)(quantity*percentage)/100;
                        for(int i=(quantity-k); i<((quantity-k)+(k/2)); i++) {
                                for(int j=quantity-1; j>((quantity-k)+(k/2)); j--) {
                                        double aux;
                                        aux = array[i];
                                        array[i] = array[j];
                        array[j] = aux;
                }
        }
}else {
```

```
        for(int i=0; i<quantity; i++) {
                double random = Math.random()*(rank2-rank1)+rank1;
                array[i] = random;
               }
        checkRepetition(quantity, array, rank1, rank2);
                        if(quantity <= 500) {
                                insertionSort(array);
                        } else if(quantity > 500 && quantity <= 100000) {
                                combSort(array);
                        } else {
                        shellSort(array);
                        }
                int k = (int)(quantity*percentage)/100;
                for(int i=(quantity-k); i<((quantity-k)+(k/2)); i++) {
                                for(int j=quantity-1; j>((quantity-k)+(k/2)); j--) {
                                double aux;
                                aux = array[i];
                                array[i] = array[j];
                                array[j] = aux;
                                }
                }
        }
}
```

Validar un valor repetido

Nombre	checkRepetition
Descripción	valida si un valor está repetido en un arreglo y lo cambia.

Entrada	quantity: int, tamaño de datos rank1: double, inicio del intervalo. rank2: fin del intervalo. arrayToSort: double[], arreglo desordenado
Retorno	sin retorno.

código:

```
public void checkRepetition(int quantity, double[] arrayToSort, double rank1, double rank2) {
                for(int i = 0; i < quantity; i++){
                        for(int j = 0; j < quantity; j++){
                                if(arrayToSort[i]==arrayToSort[j] && i!=j){
                                        double rnd = Math.random()*(rank2-rank1)+rank1;
                                        arrayToSort[i]=rnd;
                                        i=0;
                                }

                        }
                }

        }
```

Método controlador

Nombre	generateRamdom
Descripción	Se encarga de direccionar las opciones que elige el usuario a su respectivo método que se encarga de la solución a sus necesidades.
Entrada	quantity: int, tamaño de datos rank1: double, inicio del intervalo. rank2: fin del intervalo. option: int, opcion elgida por el usuario.

	optionRandom: opción de ordenamiennto de datos. percentage: double, porcentaje de ordenamiento de los datos
Retorno	Un arreglo de números ordenados.

Código:

```
public void generateRandom(int quantity, double rank1, double rank2, int option, int optionRandom, double percentage) {
	if(option == 0) {
	if(optionRandom == 1 ) {
		generateRandomOption1(quantity, rank1, rank2, option);
		}else if(optionRandom == 2) {
		generateRandomOption2(quantity, rank1, rank2, option);
			}else if(optionRandom == 3 ) {
			generateRandomOption3(quantity, rank1, rank2, option);
			}else if(optionRandom == 4) {
				generateRandomOption4(quantity, rank1, rank2, option, percentage);
			}
		}else if(option == 1) { //no repeat
			if(optionRandom == 1 ) {
				//random
				generateRandomOption1(quantity, rank1, rank2, option);
			}else if(optionRandom == 2) {//sort
				generateRandomOption2(quantity, rank1, rank2, option);
			}else if(optionRandom == 3 ) {
				//sort inverse
			generateRandomOption3(quantity, rank1, rank2, option);
			}else if(optionRandom == 4) {
	generateRandomOption4(quantity, rank1, rank2, option, percentage);
```

```
            }
        }
}
```

Generar valores aleatoriamente

Nombre	generateRandomOption1
Descripción	Se encarga de generar números aleatoriamente.
Entrada	quantity: int, tamaño de datos rank1: double, inicio del intervalo. rank2: fin del intervalo. option: int, opcion elgida por el usuario.
Retorno	Un arreglo de números ordenados.

```
public void generateRandomOption1(int quantity, double rank1, double rank2, int option) {
            if(option == 0) {
                        for(int i=0; i<quantity; i++) {
                         double random = Math.random()*(rank2-rank1)+rank1;
                                     array[i] = random;
                                    }
                        }else {
                                    for(int i=0; i<quantity; i++) {
                                    double random = Math.random()*(rank2-rank1)+rank1;
                                     array[i] = random;
                                                }
                                    checkRepetition(quantity, array, rank1, rank2);
                                    }
```

```
        }
```

Ordenar Valores

Los siguientes métodos cumplen la misma función pero lo hacen a velocidades diferentes.

Nombre	CombSort
Descripción	Ordena de forma creciente el arreglo ingresado por parámetro.
Entrada	arrayToSort: double, arreglo desordenado.
Retorno	sin retorno

```
public void combSort(double[] arrayToSort) {
                                int n = arrayToSort.length;
                            int gap = n;
    boolean swapped = true;
    while (gap != 1 || swapped == true) {
        gap = getNextGap(gap);
        swapped = false;
       for (int i=0; i<n-gap; i++)
       {
           if (arrayToSort[i] > arrayToSort[i+gap])
           {
            double temp = arrayToSort[i];
              arrayToSort[i] = arrayToSort[i+gap];
              arrayToSort[i+gap] = temp;

              swapped = true;
           }
```

```
        }
    }
        }
```

Nombre	SellSort
Descripción	Ordena de forma creciente el arreglo ingresado por parámetro.
Entrada	arrayToSort: double, arreglo desordenado.
Retorno	sin retorno

```
public void shellSort(double[] arrayToSort) {
                int jump;
        double aux;
        boolean changes;
                for(jump= arrayToSort.length/2; jump!=0; jump/=2){
                changes=true;
                   while(changes){
                changes=false;
            for(int i=jump; i< arrayToSort.length; i++) {
                if(arrayToSort[i-jump]>arrayToSort[i]){
                                aux=arrayToSort[i];
                        arrayToSort[i]=arrayToSort[i-jump];
                        arrayToSort[i-jump]=aux;
                        changes=true;
                        }
                }
            }
                }
}
```

Nombre	insertionSort
Descripción	Ordena de forma creciente el arreglo ingresado por parámetro.
Entrada	arrayToSort: double, arreglo desordenado.
Retorno	sin retorno

```
public void insertionSort(double[] arrayToSort) {
        for (int i = 1; i < arrayToSort.length; i++) {
                for (int j = i; j >0;j--) {
                        double current = arrayToSort[j];
                        double toEvaluate = arrayToSort[j-1];
                        if(toEvaluate > current) {
                                double aux = arrayToSort[j-1];
                                arrayToSort[j-1] = arrayToSort[j];
                                arrayToSort[j] = aux;
                        }
                }
        }
}
```