Integrantes:

Douglas Lopez A00347533

Laura Rubio A00346353

David Obando A00348505

Problema:
Una empresa de fabricación de microprocesadores requiere implementar métodos de ordenamiento, como operaciones nativas o propias en los coprocesadores que están creando, debido a que necesitan que estos puedan realizar trabajos de cómputo intensivo y no lo tenga que realizar el procesador principal, para que a su vez se pueda aumentar el rendimiento del sistema.

REQUERIMIENTOS

NOMBRE	R1. Ingresar datos a ordenar
RESUMEN	Se va a ingresar los datos o números que el usuario quiere ordenar
ENTRADA	Números a ordenar
SALIDA	Se ingresaron y guardaron los números

NOMBRE	R2. Generar números aleatorios
RESUMEN	Se va a generar números aleatorios que van a hacer ordenados
ENTRADA	Ninguna
SALIDA	Se generaron los datos

NOMBRE	R3. Elegir la cantidad de números aleatorios
RESUMEN	Se va a generar la cantidad de números aleatorios especificada por el usuario
ENTRADA	Cantidad
SALIDA	Se generó la cantidad de números

NOMBRE	R4. Escoger el intervalo de los números aleatorios

RESUMEN	Se va a generar los números aleatorios dentro del intervalo especificado
ENTRADA	Intervalo
SALIDA	Se generaron los números dentro del intervalo

NOMBRE	R5. Indicar la repetición o no de los números aleatorios
RESUMEN	Se va a permitir indicar si los números aleatorios pueden ser generados con o sin repetición
ENTRADA	Ninguna
SALIDA	Se generaron los números de acuerdo a la especificación de repetición del usuario

NOMBRE	R6. Generar números aleatorios ordenados
RESUMEN	Se va a generar un arreglo de números aleatorios ordenados
ENTRADA	Arreglo de números aleatorios
SALIDA	Se generó el arreglo

NOMBRE	R7. Generar números aleatorios ordenados inversamente
RESUMEN	Se va a generar un arreglo de números aleatorios ordenados de manera inversa
ENTRADA	Arreglo de números aleatorios
SALIDA	Se generó el arreglo

NOMBRE	R8. Generar números aleatorios en orden completamente aleatorio
RESUMEN	Se va a generar un arreglo de números aleatorios ordenados de manera totalmente aleatoria.
ENTRADA	Arreglo de números aleatorios
SALIDA	Se generó el arreglo

NOMBRE	R9. Generar números aleatorios con un ordenamiento parcial
RESUMEN	Se va a generar un arreglo de números aleatorios con un porcentaje de números desordenados y el resto ordenado dado por el usuario
ENTRADA	Arreglo de números aleatorios % de números desordenados
SALIDA	Se generó el arreglo

NOMBRE	R10. Ordenar arreglo
RESUMEN	Se va a ordenar el arreglo ingresado por el usuario y/o generado aleatoriamente según las especificaciones del mismo.
ENTRADA	Arreglo de números
SALIDA	Se ordenó el arreglo

NOMBRE	R11. Medir tiempo de ejecución
RESUMEN	Se va a medir el tiempo que le toma al algoritmo, ordenar el arreglo
ENTRADA	Ninguna
SALIDA	Se midió el tiempo

NOMBRE	R12. Mostrar los algoritmos disponibles para ordenar
RESUMEN	Se va a restringir o permitir el uso de un algoritmo de acuerdo al tipo de números a ordenar y al intervalo de números generados
ENTRADA	Tipo de números a ordenar Intervalos de números generados
SALIDA	Se mostraron los algoritmos disponibles

2. Búsqueda de información:

Para iniciar la búsqueda sobre métodos de ordenamiento recurrimos a un documento[1] donde mencionan los algoritmos más populares.

sort	# hits 2000	# hits 2002
Quick	26,780	80,200
Merge	13,330	33,500
Heap	9,830	22,960
Bubble	12,400	33,800
Insertion	8,450	21,870
Selection	6,720	20,600
Shell	4,540	8,620

Table 1: Web-based popularity of sorts

Teniendo en cuenta la imagen anterior, encontramos que los algoritmos de ordenamiento se dividen en dos grandes categorías: Directos e indirectos, en la primera de ellas se encuentran Burbuja, Inserción y Selección, los cuales son efectivos para entradas relativamente pequeñas, para nuestro caso es necesario basarse principalmente en su complejidad temporal cuando sus entradas son bastante grandes, mientras que los indirectos se refieren a algunos más avanzados, como resultado logramos dar inicio a una lista de posibles soluciones al problema planteado.

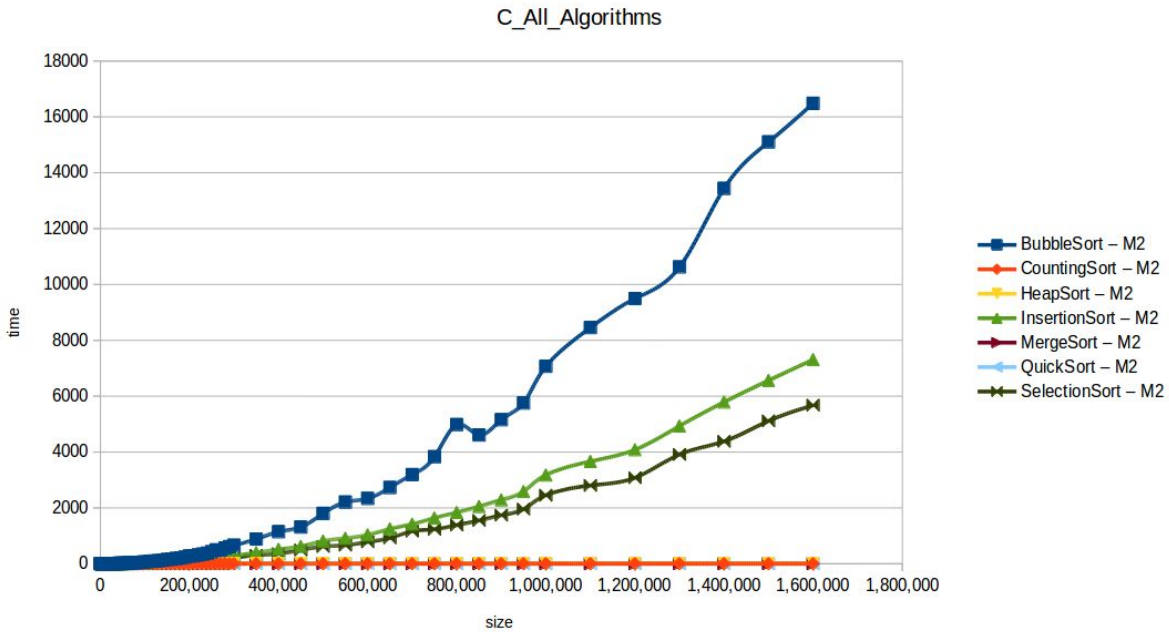
Las alternativas para los métodos de ordenamiento son las siguientes:

1. Ordenamiento de inserción *
2. Ordenamiento de burbuja
3. Ordenamiento por cuentas
4. Ordenamiento Shell
5. Ordenamiento por montículos
6. Ordenamiento por mezcla. *
7. Ordenamiento rápido *

Investigamos un poco más sobre cada uno de ellos para establecer qué tan próximos están a nuestras necesidades, en el libro Introduction to algorithms explican que" Inserción, mezcla, montículos y ordenamiento rápido son métodos de comparación, ellos determinan el orden de un vector de entrada mediante la comparación de elementos" .La diferencia entre ellos, se basa en

[1] Bubble Sort: An archaeological Algorithmic Analysis

cómo dividen o hacen estas comparaciones, esto hace que se puedan caracterizar según su rapidez, a continuación hallamos una gráfica que permite relacionar cada uno de ellos.



[2]

Lo anterior, nos ayuda a tomar decisiones, sin embargo es necesario para nuestro caso no solo analizar el más rápido sino también las entradas que puede recibir el algoritmo, como es el caso de ordenamiento por cuentas (Counting sort), el cual no cumple su función para entradas decimales. A continuación, teniendo en cuenta lo recolectado haremos la búsqueda de soluciones para posteriormente seleccionar las ideales.

3. Búsqueda de soluciones creativas:

Se necesita una solución al problema; para esto se basó en la búsqueda de algoritmos de ordenamientos rápidos (fundamentado en su complejidad) y algoritmos de selección de números aleatorios eficientes, generando una lluvia de ideas con respecto a las características de cada uno. Para revisar nos apoyamos en unas ciertas preguntas para poder alcanzar la mejor alternativa de solución a nuestro problema:

¿Es realmente fiable el proceso de ejecución del algoritmo?
¿Sirve para suplir las necesidades del problema?
¿Produce la salida del algoritmo correctamente?

[2] Tomado de: https://pereiratechtalks.com/analisis-de-algoritmos-de-ordenamiento/

Después de una rigorosa y ardua recopilación de información, procedemos a tomar alternativas que nos ayudarán a enfocar nuestras necesidades hacía determinadas posibles resoluciones, cabe aclarar que, no son definitivas, pero sí admisibles.
Cuando estamos en este proceso de búsqueda de soluciones creativas, debemos saber cosas importantes acerca de las alternativas que nos vamos topando durante todo este proceso del método de diseño de la ingeniería, por tanto, a continuación nos percatamos en informaciones encontradas en fuentes altamente visitadas por los usuarios.

Alternativas:

Ordenamiento por Mezcla

Ahora nos concentramos en un algoritmo que como bien vimos en la fase anterior, es uno de los más populares y usados por los programadores para llevar a cabo la solución a sus problemas; además este algoritmo es un poco peculiar, ya que consta de recursividad para su implementación, esto quiere decir que se puede repetir o aplicar indefinidamente.
Por otra parte, su funcionalidad es bastante precisa y clara, porque se encarga de dividir continuamente una lista por la mitad. Si la lista está vacía o tiene un solo ítem, se ordena por definición(el caso base). Si la lista tiene más de un ítem, dividimos la lista e invocamos recursivamente un ordenamiento por mezcla para ambas mitades. Una vez que las dos mitades están ordenadas, se realiza la operación fundamental, denominada mezcla. La mezcla es el proceso de tomar dos listas ordenadas más pequeñas y combinarlas en una sola lista nueva y ordenada.

Ordenamiento por Inserción

Ahora bien, el ordenamiento por inserción es una manera muy natural de ordenar para un ser humano. La idea de este algoritmo de ordenación consiste en ir insertando un elemento de la lista ó un arreglo en la parte ordenada de la misma, asumiendo que el primer elemento es la parte ordenada, insertando el elemento en la posición correcta dentro de la parte ordenada, y así sucesivamente hasta obtener la lista ordenada.

Ordenamiento Rápido

En este sentido, nos encontramos con el algoritmo más popular entre los usuarios según fuentes ya mencionadas.Este algoritmo (también llamado QuickSort) se basa en la técnica divide y vencerás; quizás es la técnica más eficiente y en la mayoría de los casos da mejores resultados.

Lo más importante de este algoritmo es escoger un elemento el cual llamaremos como pivote, después resituaremos los demás elementos de la lista a cada lado del pivote, de manera que de un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
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.

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.

Ordenamiento Burbuja
Quizás este algoritmo no sea el más eficiente, pero tiene algo en que nos basamos, y es que cumple con su objetivo durante todo su proceso; su proceso es bastante sencillo, funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiandolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Incluyendo a todo esto, otra parte importante para la solución total de nuestro problema, es lo que sería un algoritmo para la obtención de números aleatorios, éste se fundamenta en la secuencia de una tupla n-números escogida por el usuario, como lo planteamos en los requerimientos del problema. Por lo que nos llevó a buscar una solución bastante creativa y en base a las funcionalidades de Java. Se investigó un poco dentro de las funcionalidades de Java y se encontró con el siguiente algoritmo:

Números aleatorios al estilo Java 8:
Es un algoritmo bastante útil, ya que se basa en los métodos internos que contienen las clases Random e IntStreams de Java, por esto ofrece más fiabilidad durante el proceso de ejecución. Esto quiere decir que, no vamos a tener un proceso de codificación bastante amplio en nuestro programa para solucionar este problema.

Relacionando lo anterior, junto a la investigación de información y las preguntas planteadas, es claro ver que son algoritmos altamente utilizados para la resolución de problemas de ordenamiento, cumple con su función, por ende son candidatos para ser una posible solución a nuestras exigencias. Más adelante se planteará un descarte y unos criterios más a fondo para la toma de decisión de la mejor alternativa.

4. Transición de las ideas a los Diseños Preliminares

En resumen de lo que llevamos en las fases anteriores, se investigó y se tomaron como alternativas los algoritmos de ordenamiento por burbuja, rápido, mezcla e inserción. Teniendo claro esto, pasamos a la cuarta fase de nuestro método de Diseño en la Ingeniería.

En un principio todos los algoritmos fueron basados en su funcionalidad y salida, pero qué tan factible puede ser esto, puede que el algoritmo dé una solución correcta a nuestro problema, pero qué tan eficaz puede ser durante ciertos casos X en nuestro programa. Más adelante haremos una breve comparación para dejar más claro hacia qué sentido queremos llegar. Por ahora, fijémonos en un momento en qué nos enuncia el problema, claramente nos pide una solución bastante rápida y eficaz, por lo que ya vamos conociendo un poco hacia dónde vamos, para este caso, solamente se hará un breve descarte para uno de los algoritmos ya planteados anteriormente, ya que para darle resolución a nuestro problema se requieren 3 métodos de ordenamiento.

Características por cada uno de los algoritmos que elegimos:

Ordenamiento por burbuja

- Funciona comparando elementos de dos en dos en un ciclo, intercambiando según sea el caso ascendente o descendente.
- Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios.
- Es bastante sencillo.
- Es un código reducido.
- Se realiza el ordenamiento.
- Eficaz.
- Consume bastante tiempo en la computadora
- Requiere muchas lecturas/escrituras en memoria.

Ordenamiento Rápido (QuickSort)

- Es el algoritmo de ordenación más rápido conocido (O(nlogn)).
- Realiza menos operaciones.
- Se basa en la técnica divide y vencerás.
- Muy rápido.
- No requiere memoria adicional.
- Implementación un poco más complicada.
- Utiliza muchos recursos.

Ordenamiento por Inserción

- Descarga bastante la utilización de memoria pero no reduce su complejidad.
- Código sencillo.
- Al igual que el ordenamiento burbuja, tiene buen rendimiento con una pequeña lista.
- Se requiere de espacio mínimo

Ordenamiento por Mezcla

- Es estable.
- Está definido recursivamente.
- Su implementación no recursiva implementa una pila, por lo que requiere un espacio de memoria para almacenarla.
- Al igual que QuickSort es un algoritmo de ordenación rápido (O(nlogn))

De lo anterior se puede deducir que, el método burbuja e inserción en comparación con QuickSort o Mezcla son bastante distantes, por lo que dos últimos son más eficientes. Como lo dijimos al comienzo de esta fase, se requiere que sea lo más factible para criterios de rapidez, ya que si nos preguntamos si de verdad cumple con ordenar, la respuesta sería sí (todos los escogidos lo hacen); pero para este caso, los más eficientes serían QuickSort y Mezcla ya que cumplen con su propósito a menor tiempo.
Ahora bien, para el caso entre Inserción y Burbuja estaría el dilema, pero en verdad no sería tan difícil decidir, ya que Burbuja por las características mencionadas anteriormente presenta mayores desventajas, tarda un poco más a la hora de ejecución cada que el array presenta mayores datos y consume mucho más memoria.

5. Evaluación y Selección de la Mejor Solución

En la evaluación vamos a tener en cuenta varios criterios:

Criterio A: Eficiencia.
[3] Logarítmica
[2] Lineal
[1]Cuadratica

Criterio B: Requiere memoria computacional.
[3] Poca
[2] Media
[1] Bastante

Criterio C: Entre más datos
[3] No varía
[2] Varía poco
[1] Varía mucho

Criterio D: Realiza operaciones
[2] Pocas
[1] Muchas

Evaluación:

Métodos / Criterios	**Criterio A**	**Criterio B**	**Criterio C**	**Criterio D**	**Total**
Burbuja	1	1	1	1	4
Inserción	1	2	1	1	5
QuickSort	3	3	3	2	11
Mezcla	3	2	2	2	9

De acuerdo con la evaluación anterior se debe seleccionar los métodos con más puntajes, en este caso, Inserción, QuickSort y Mezcla, ya que presentan criterios bien evaluados que suplen con las necesidades del problema.

Para concluir esta fase, sólo queda decir que, llevando un análisis más cercano a nuestros métodos se puede determinar cuáles serían los más acertados para la solución a nuestro problema, en este caso descartamos uno de los cuatro, por lo que quedamos sólo con 3 métodos definitivos, Inserción, Rápido y Mezcla.

6. Preparación de Informes y Especificaciones

Especificación del problema:
Una empresa de fabricación de microprocesadores requiere implementar métodos de ordenamiento, como operaciones nativas o propias en los coprocesadores que están creando, debido a que necesitan que estos puedan realizar trabajos de cómputo intensivo y no lo tenga que realizar el procesador principal, para que a su vez se pueda aumentar el rendimiento del sistema.

En resumidas cuentas en las entradas del problema, se requiere una tupla de n-números el cual se le aplicará la ordenación mediante los métodos seleccionados en las fase pasada.

Pseudocódigo y análisis de complejidad.

Ordenamiento inserción (pseudocódigo)	Costo	#Veces que se ejecuta
desde j = 2 hasta longitud A haz inicio	c1	n
llave:=A[j]	c2	n-1
i:=j - 1	c3	n-1
`mientras (i > 0) y (A[i] > llave) haz inicio`	c4	(n^2 + n)/2-1
`A[i + 1] := A[i]`	c5	(n^2 +n)/2
`i:=i - 1`	c6	(n^2+n)/2
fin-mientras	0	0
`A[i + 1]:=llave`	c7	n-1
`fin-desde`	0	0
	T=	(3/2)cn^2 + 3/2cn2 - 4c ... O(n^2)

Tabla 1.1

Ordenamiento por Mezcla.

Para entender el análisis de complejidad del ordenamiento por mezcla, nos basaremos en la técnica de divide y vencerás:

Dividir: Dividir el problema en un cierto números de subproblemas.

Vence: Soluciona los problemas de manera recursiva.

Combinar: Combina el resultado de los subproblemas para obtener la solución al problema original.

Lo anterior se trato porque el ordenamiento por Mezcla se fundamenta en esta técnica. Cabe aclarar que esta técnica tiene una estimación en cuanto a su análisis, como lo veremos a continuación.

Divide: Como esta parte se refiere a dividir el problema en subproblemas, entonces se deduce que divide la secuencia **n** de elementos en dos subsecuencias, por lo que saldría **n/2.**

Vencer: Ordena ambas secuencias de manera recursiva. Después como dividimos el problema a la mitad y se resuelve recursivamente, entonces estamos resolviendo dos problema de tamaño n/2 lo que conlleva a darnos un tiempo de **T(n) = 2T(n/2) + cn** lo cual la "c" representa una constante igual al tiempo que se requiera para resolver un problema de tamaño 1.

Pseudocódigo Mezcla

```
Mezcla(A,p,q,r)
1     n1:=q - p + 1
2     n2:=r - q
3     Crea arreglo L[1..n1 + 1]
4     Crea arreglo R[1..n2 + 1]
5     desde i:=1 hasta n1 haz
6     L[i]:=A[p + i - 1]
7     desde j:=1 hasta n2 haz
8     R[j]:=A[q + j]
9     L[n1 + 1]:=infinito
10   R[n2 + 1]:=infinito
11   i:=1
12   j:=1
13   desde k:=p hasta r haz inicio
14    Si L[i] <= R[j] entonces inicio
15          A[k]:=L[i]
16          i:=i + 1
17    Si-No entonces inicio
18          A[k]:=R[j]
19          j:=j + 1
20    fin-Si-entonces
21   fin-desde
```

Valor n	función de recurrencia	tiempo total
1	c	c
2	2T(2) + 2c	2c + 2c = 4c = c(2 lg(2) +2)
4	2T(4) + 4c	*8c + 4c =12c = c(4 lg(4) + 4)*
8	2T(8) + 8c	*24c+8c=32c=c(8 lg(8) + 8)*
.	.	
n	2T(n) + nc	*c(n lg(n) + n)*

Tabla 1.2

De la tabla 1.2 vemos que el tiempo real de corrida del ordenamiento por mezcla es proporcional a ***n lg n + n***, de nuevo, como en el análisis de complejidad únicamente nos importa el término que crezca más rápido y en ésta función ese término es ***n lg n***, por lo tanto la complejidad del sistema es ***O(n lg n)***.

Ordenamiento rápido (QuickSort)

Para este método sucede lo mismo que en el algoritmo de ordenamiento por mezcla, ya que los dos se basan en la misma técnica de divide y vencerás, son algoritmos con fuentes recursivas, por lo tanto como lo mencionamos anteriormente tiene una función definida en este caso **T(n) = 2T(n/2) + cn**

Para este caso, veremos una idea de como funciona el algoritmo de QuickSort mediante su pseudocódigo.

```
inicio
      variables A: arreglo[1..100] entero
      variables i,j,central:entero
      variables primero, ultimo: entero
      para i = 1 hasta 100
                  leer(A[i])
            Fin para
      primero = 1
      ultimo = 100
      qsort(A[],100)
Fin

Funcion qsort(primero, ultimo:entero)
            i = primero
            j = ultimo
            central = A[(primero,ultimo) div 2]
            repetir
                        mientras A[i]central
                                    j = j - 1
                              fin mientras
                        si i < = j
                                    aux = A[i]
                                    A[j] = A[i]
```

```
                                    A[i] = aux
                                    i = i + 1
                                    j = j - 1
                                fin si
                        hasta que i > j
                si primero < j
                                partir(primero,j)
                        fin si
                si i < ultimo
                                partir(i, ultimo)
                        fin si
        fin funcion qsort
```

SI analizamos el pseudocódigo, la funcionalidad durante el proceso es diferente a la del ordenamiento de Mezcla, pero su técnica es la misma. Por lo que, el análisis de complejidad para este algoritmo sería:

Valor n	función de recurrencia	tiempo total
1	c	c
2	2T(2) + 2c	2c + 2c = 4c = c(2 lg(2) +2)
4	2T(4) + 4c	*8c + 4c =12c = c(4 lg(4) + 4)*
8	2T(8) + 8c	*24c+8c=32c=c(8 lg(8) + 8)*
.	.	
n	2T(n) + nc	*c(n lg(n) + n)*

Esto sigue siendo proporcional **n log(n) + n**, por lo que su crecimiento sigue persistente en **n log(n)**, entonces su complejidad es **O(n log(n)).**

Complejidad Espacial

En este proceso de complejidad, se requiere saber los bits que ocupa cada dato primitivo en memoria, por ende, de ahí es donde se genera el espacio que ocupa cada algoritmo en memoria con un n números de elementos.

Ordenamiento Insercion

Array (integer) = 32n -------> esto es porque los bits para datos integer ocupa 32 bits y no conocemos el tamaño del array por eso se declara la variable n.

int i = 32

int j = 32

int temp = 32

return array = 32n

por tanto, el total de espacio que ocupara este algoritmo durante su ejecución está dado por la siguiente función lineal F(n) = 64n + 96.

Ordenamiento Rápido (QuickSort)

Parametros:

ArrayList = 64n -------> ya que es un arraylist de double.

int i = 32

int j = 32

-

int k = 32

int l = 32

Double pivot = 64

por tanto, F(n) = 64n + 192.

Ordenamiento por Mezcla

Parametros:

ArrayList = 64n ----------> ya que es un arraylist de double

int i = 32

int size = 32

.

.

int left = 32

int right = 32

int max = 32

por tanto, F(n) = 64n + 160

Por otra parte, veremos unos test que se encargaran de darle una previsión a lo que se espera que lance el programa, fijándonos en los métodos que se emplearon para la solución del problema

Clase	Método	Escenario	Valores	Resultado
Principal Test	testVerificateIntegerDouble()	Se instanció un objeto de tipo Principal	Numeros0=0,50,3,8,72 Porcentaje0=100% Numeros1=8.3,5.55,9.0 Porcentaje1=50%	Numero0 = false Numero1 = true
Principal Test	testCalculatePercent()	Se instanció un objeto de tipo Principal	Tamaño arreglo = 6 Porcentaje = 50%	Tamaño arreglo = 3
Principal Test	testChangeNumbersDouble()	Se instanció un objeto de tipo Principal	Numeros0 = 1,2,3,4 Porcentaje0 = 100%	Numeros0 = 1.0,2.0,3.0,4.0
Principal Test	testChangeNumbersInt()	Se instanció un objeto de tipo Principal	Numeros1 = 1.0,2.0,3.0,4.0 Porcentaje1 = 100%	Numeros1 = 1,2,3,4
Principal Test	testRandomInt()	Se instanció un objeto de tipo Principal	Cantidad = 2 Repetidos = false Intervalo = [0,7]	Numeros0 = 5,2

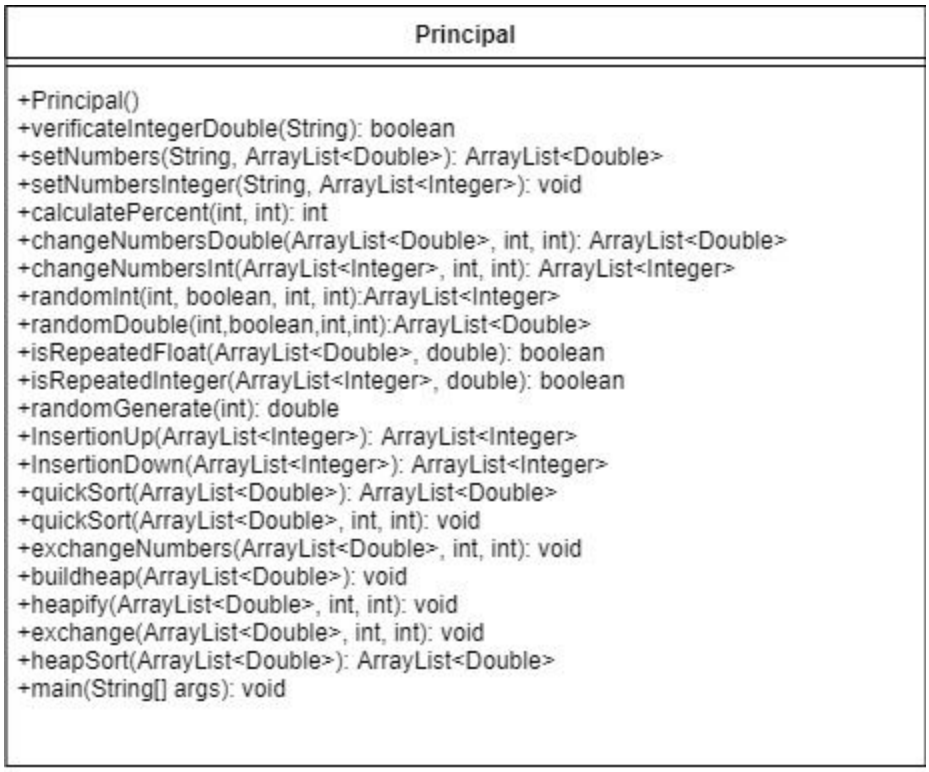
Principal Test	testRandomDouble()	Se instanció un objeto de tipo Principal	Cantidad = 4 Repetidos = true Intervalo = [0,10]	Numeros0 = 1.0,2.0,2.0,3.0
Principal Test	testIsRepeatedFloat()	Se instanció un objeto de tipo Principal	Numeros0= 1.0,1.0,3.0,4.0 NumeroRep = 1.0	true
Principal Test	testIsRepeatedInteger()	Se instanció un objeto de tipo Principal	Numeros0= 1,1,3,4,5 NumeroRep = 1	true
Principal Test	testQuickSort()	Se instanció un objeto de tipo Principal	Numeros0=0,5200,3,8,74 Porcentaje0=100% Numeros1=8,5.55,9 Porcentaje1=50%	Numeros0=0,3,8,5200,74 Numeros1=5.55,9,8 La lista de números dados debe estar ordenada de menor a mayor
Principal Test	testHeapSort()	Se instanció un objeto de tipo Principal	Numeros0=5,15,0,9 Porcentaje=90% Numeros1=5,3,2,100000 Porcentaje1=33%	Numeros0=0,5,9,15 Numeros1=10000,3,2,5 La lista de números dados debe estar ordenada de menor a mayor
Principal Test	testInserciónUp()	Se instanció un objeto de tipo Principal	Numeros0=5,1,0,98.5 Porcentaje0=81%	Numeros0=0,1,5,98.5 La lista de números dados debe estar ordenada de menor a mayor

Principal Test	testInserción Down()	Se instanció un objeto de tipo Principal	Numeros0=0,85,98,7	Numeros0=98,85,7,0 Numeros0=114,9,8,0 La lista de números dados debe estar ordenada de mayor a menor

tabla 1.3

7. Implementación del Diseño

En esta última etapa se empieza a llevar a cabo el proceso de codificación, pero antes de ponernos a solucionar el problema mediante el lenguaje de programación, hay que tener una idea de lo que vamos a hacer durante o siquiera por dónde empezar, para eso usamos un diagrama basado en UML, conocido comúnmente como diagrama de clases. Este fue nuestra guía para llevar a cabo los métodos que hicieron posible alcanzar el objetivo de la solución.



Bibliografía:

http://chuwiki.chuidiang.org/index.php?title=Generar_n%C3%BAmeros_aleatorios_en_Java
Algoritmos y estructuras de datos, capítulo 6: Algoritmos de Ordenación y Búsqueda. Tomado de:
http://novella.mhhe.com/sites/dl/free/844814077x/619434/A06.pdf
Introduction to the Algoritms,Third edicion, T.Cormen et all (2009).
http://interactivepython.org/runestone/static/pythoned/SortSearch/ElOrdenamientoPorMezcla.html
Solución de problemas con algoritmos y estructuras de datos. 5.11 El ordenamiento por mezcla.
http://lwh.free.fr/pages/algo/tri/tri_insertion_es.html
Ordenamiento por Inserción.
https://prezi.com/bwfhhwbpkk2t/metodo-burbuja/
Ordenamiento por burbuja.
https://prezi.com/t0ivqgci_gwk/metodo-de-ordenamiento-por-mezcla/
Ordenamiento por mezcla.
http://nereida.deioc.ull.es/~cleon/doctorado/doc04/src/mallba/quicksort.html
Ordenación Rápida (QuickSort).
https://insertionsort.wordpress.com/2013/05/21/insertion-sort-insercion-directa/
Insertion Sort.
http://www.olimpiadadeinformatica.org.mx/omi/omi/archivos/apuntes/AnalisisDeComplejidad.htm
Complejidad.