En el presente informe, referente a la práctica de “Algoritmos de Ordenamiento Elementales”, trataremos 4 (cuatro) algoritmos de ordenamiento (Selección, Burbujeo, Inserción y Selección doble), en los cuales analizaremos y compararemos los diferentes tiempos de ejecución que tienen estos algoritmos sobre distintas secuencias (Ordenado, Aleatorio e Inverso) y tamaños de entrada (1K, 100K, 500k, 1M).

Para esto utilizamos el lenguaje de programación java en el cual codificamos los 4 algoritmos de ordenamiento y el generador de los distintos conjuntos de datos, y medimos los tiempos que le tomaba a cada algoritmo ejecutarse para cada conjunto de datos.

Esto nos sirve para entender cómo se comporta cada algoritmo sobre cada conjunto de datos y compararlos entre sí para determinar en qué situaciones un algoritmo es más eficiente que otro, sus ventajas y desventajas, entre otras cosas.

Ahora pasaremos a describir los algoritmos de ordenamiento:

Selección:

Este algoritmo es no sensible y no estable.

34Este algoritmo busca el menor elemento de un array y lo intercambia con el primer elemento de la lista. Después de esto recorrerá de nuevo la lista buscando el segundo menor y lo intercambiará con el segundo elemento, siguiendo de esta forma hasta que se hayan intercambiado todos los elementos de la lista y la misma quede ordenada.

Complejidad Computacional:

Peor caso, $O(n^2)$ comparaciones, $O(n)$ intercambios

Mejor caso, $O(n^2)$ comparaciones, $O(1)$ intercambios

Caso Promedio, $O(n^2)$ comparaciones, $O(n)$ intercambios

Burbujeo:

Este algoritmo es sensible y estable.

Este algoritmo empieza con los primeros dos elementos, intercambiándolos de posición si el primero es mayor al segundo, se repetirá esta comparación ahora con el segundo y el tercero, y así hasta que el mayor elemento quede en el tope de la lista. Este proceso se repetirá hasta que ya no se produzca ningún intercambio. Lo cual significa que la lista ya quedo ordenada.

Complejidad Computacional:

Peor caso, $O(n^2)$ comparaciones, $O(n^2)$ intercambios

Mejor caso $O(n)$ comparaciones, $O(1)$ intercambios

Caso Promedio, $O(n^2)$ comparaciones, $O(n^2)$ intercambios

Inserción:

Este algoritmo es sensible y estable.

Este algoritmo empieza tomando el segundo elemento de la lista y lo compara con el primero, si el primero es mayor al segundo el primero se moverá una casilla a la derecha y el segundo se insertará en la posición liberada por primero. Luego, se tomará al tercer elemento y se lo comparará con los dos anteriores, moviendo a aquellos números anteriores que sean mayores al tercero una casilla a la derecha e insertando el tercero en la posición libre encontrada. Esto se repetirá por cada uno de los elementos de la lista, "extrayendo/tomando" dicho número y comparando con los anteriores, moviendo los mismos a la derecha a aquellos que sean mayor al número extraído hasta llegar al final de la lista o encontrar un número menor al extraído, entonces el numero extraído se insertará en la posición libre. Y así hasta ordenar por completo la lista.

Complejidad Computacional:

Peor caso, O(n^2) comparaciones e intercambios

Mejor caso O(n) comparaciones, O(1) intercambios

Caso Promedio, $O(n^2)$ comparaciones e intercambios

Selección doble:

Este algoritmo es no sensible y no estable.

Este algoritmo está basado en el algoritmo de selección, solo que en este caso en la primera pasada se busca tanto el menor como el mayor de la lista, y se intercambiaran sus posiciones con el primer y último elemento respectivamente de la lista. En la siguiente pasada, se buscará y el segundo menor y el segundo mayor de la lista, y se intercambiaran posiciones con el segundo y anteúltimo elemento de la lista. Se repetirá este proceso hasta que se llegue a la mitad de la lista. Ya que al poder intercambiar tanto el menor como el mayor de la lista en una iteración se ahorra la mitad de las iteraciones.

Complejidad Computacional:

Peor caso, $O(n^2)$ comparaciones, O(n) intercambios

Mejor caso $O(n^2)$ comparaciones, O(1) swap

Caso Promedio, $O(n^2)$ comparaciones, O(n) intercambios

Worst-case space complexity, O(1) auxiliary

Ahora procederemos a explicar cómo generamos los conjuntos de datos:

Lista de datos aleatorios:

Por cada una se las posiciones de la lista se crear e inserta un numero aleatorio con un objeto de la clase Random y ejecutando la función nextInt() que te devuelve un numero aleatorio según el rango especificado [0, rango máximo). Para poder tener números negativos a cero se realizará lo siguiente. Poniendo que rango mínimo es –5 y máximo es 10.

.nextInt(10-(-5))-5; si llegara a salir el número 0 de nextInt() este seria restado –5 y quedaría en el rango mínimo deseado. Si saliera el máximo 15 este seria restado –5 quedaría en el rango máximo deseado 10.

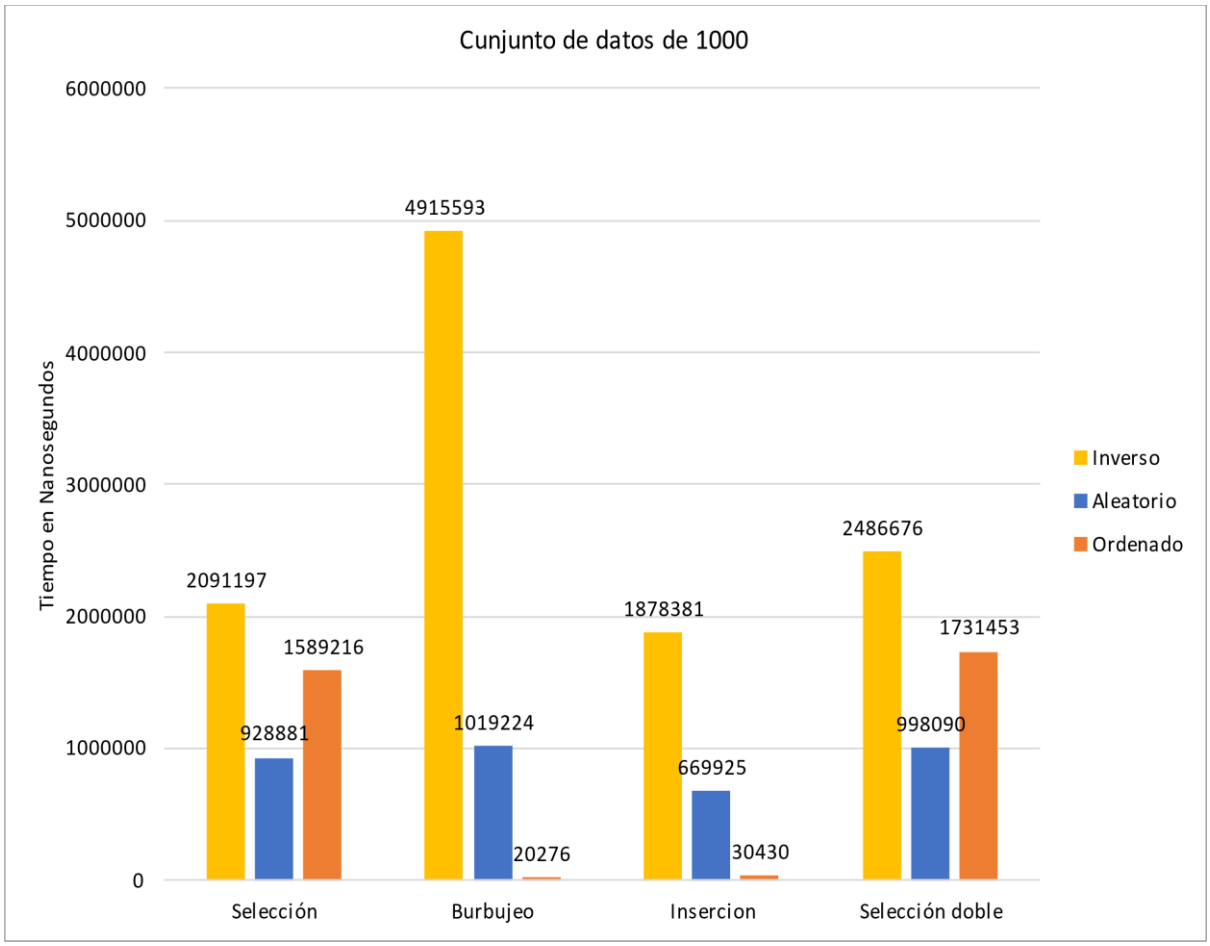
Lista de datos Ordenado:

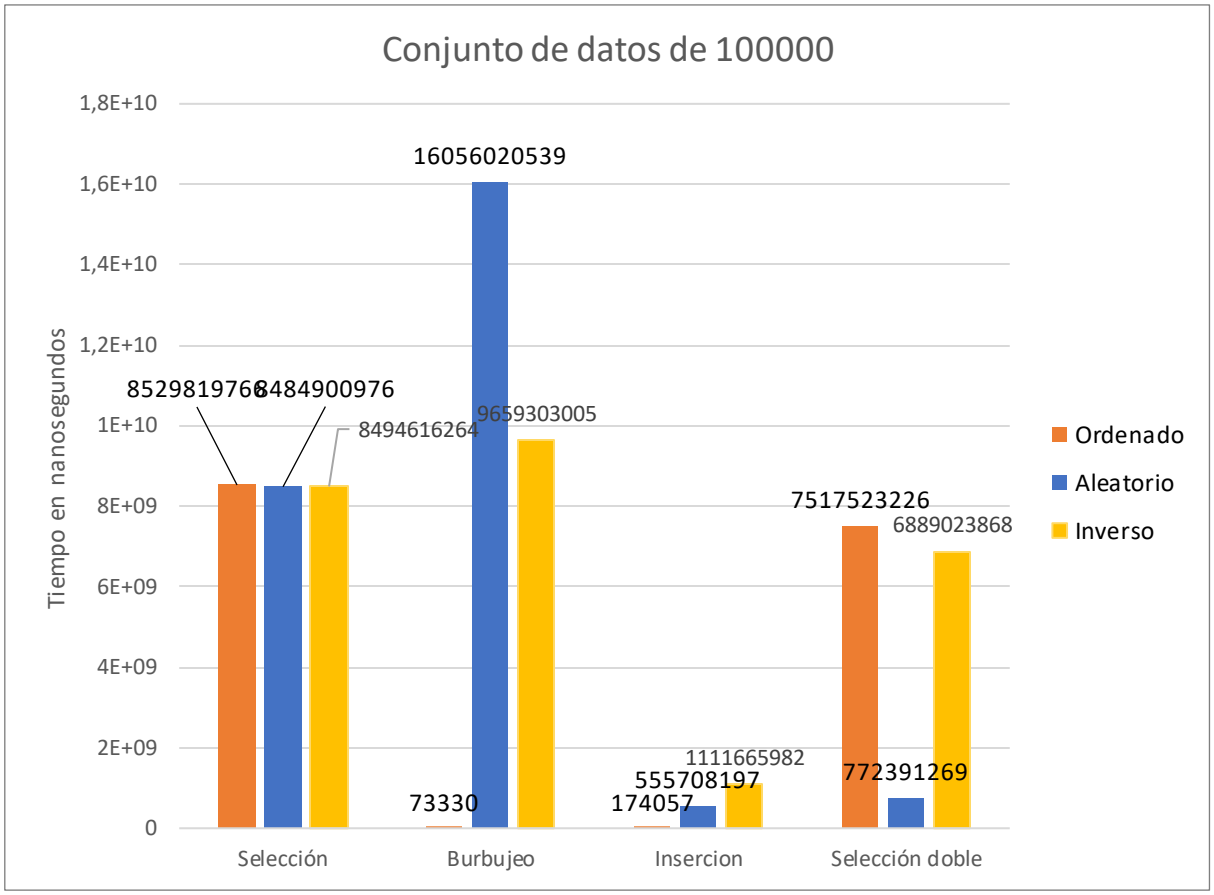
Se realizará el mismo paso que arriba solo que ahora se llamará a la función Arrays.sort(array) para ordenar la lista utilizando un método de ordenamiento optimizado. De esta forma podemos permitir que haya números repetidos

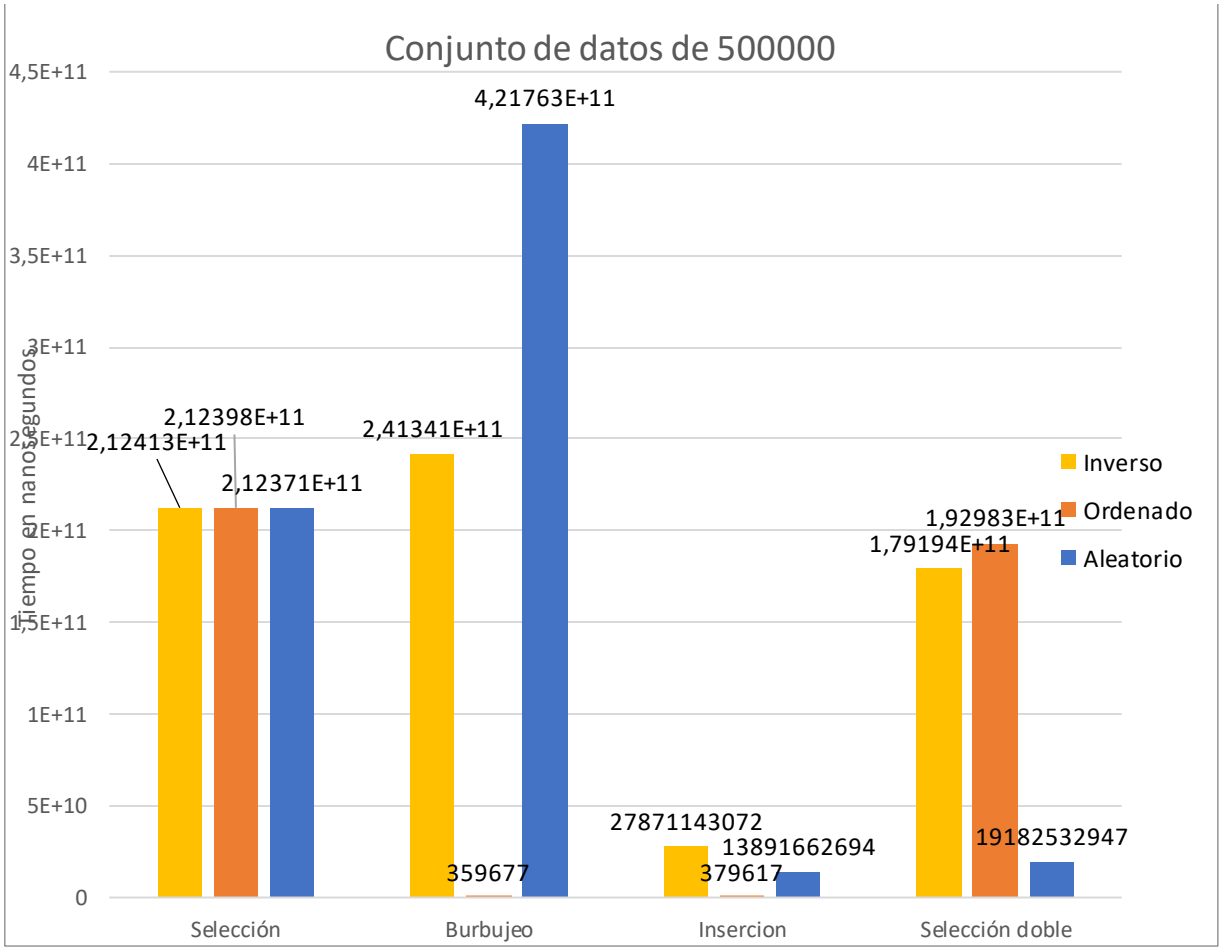
Lista de datos Inverso:

Se realizarán los mismos pasos que en la lista de datos Ordenado. Solo que por último se llamara a la función InvertirInt(array) que intercambiara las posiciones de todos los elementos de la lista.

Resultados de las mediciones:




Conjunto de datos de 100000
Tiempo en nanosegundos
1,8E+10
1,6E+10
1,4E+10
1,2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
8529819768
8484900976
8494616264
16056020539
9659303005
73330
174057
555708197
1111665982
7517523226
772391269
6889023868
Selección
Burbujeo
Insercion
Selección doble
Ordenado
Aleatorio
Inverso


Conjunto de datos de 500000
Tiempo en nanosegundos
4,5E+11
4E+11
3,5E+11
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
2,12413E+11
2,12398E+11
2,12371E+11
2,41341E+11
4,21763E+11
359677
27871143072
379617
13891662694
1,79194E+11
1,92983E+11
19182532947
Selección
Burbujeo
Insercion
Selección doble
Inverso
Ordenado
Aleatorio

Estas pruebas se realizaron en un procesador Intel I5-1135G7 de 4.20HZ

En el caso de 1 millón de elementos no nos fue posible obtener datos ya que el tiempo de ejecución para cada prueba tardaba aproximadamente 6 horas.

En el algoritmo de selección es notable observar que con 100.000 y 500.000 elementos los tiempos de ordenamiento para los 3 tipos de datos es prácticamente el mismo. Esto es lo esperable ya que es un algoritmo no sensible.

Tanto en el algoritmo de burbujeo como el de inserción al ser ambos estables se puede observar que el tiempo de ordenamiento cuando están ordenados es prácticamente instantáneo ya que no tienen que hacer ninguna operación de intercambio o movimiento de datos.

El algoritmo de burbujeo es el peor de los 4 en el caso de trabajar con arrays de gran tamaño cuando estos están en el orden aleatorio.

El algoritmo de selección doble esta apenas por debajo del de inserción en el ordenamiento aleatorio al trabajar con arrays de gran tamaño.

Pero al trabajar con los de arrays inverso o aleatorios es relativamente parecido al algoritmo de selección normal.

El tiempo de ejecución del algoritmo de inserción es el más bajo en los 3 tipos de ordenamiento.