Informe: Taller Algoritmos de ordenamiento

Autores:
Juan Nicolás Diaz Salamanca – 20232020059
Juan Esteban Galindo – 20232020095

Universidad Distrital Francisco José De Caldas
Facultad de Ingeniería

Ciencias de la computación 1
Docentes:
Roberto Albeiro Pava Diaz
Helio Henry Ramirez Arévalo

Fecha:

09/09/25

1. Introducción
   La Asociación de Profesores Universitarios Totalitarios y Anarquistas Socialistas definirá su proceso de elección, ASO - SIN SIGLA (La junta directiva no se ha puesto de acuerdo

con la sigla de la organización), con esta finalidad plantean los siguientes criterios de elección:

- Distancia total recorrida en marchas
- Horas de clase perdidas por bloqueos
- Valor total de prebendas sindicales recibidas
- Número de políticos de los que han recibido sobornos
- Valor total atribuido a actos de corrupción en los que han participado.

Esta información se presenta en orden ascendente al comité según el criterio escogido y se selecciona al mejor candidato.

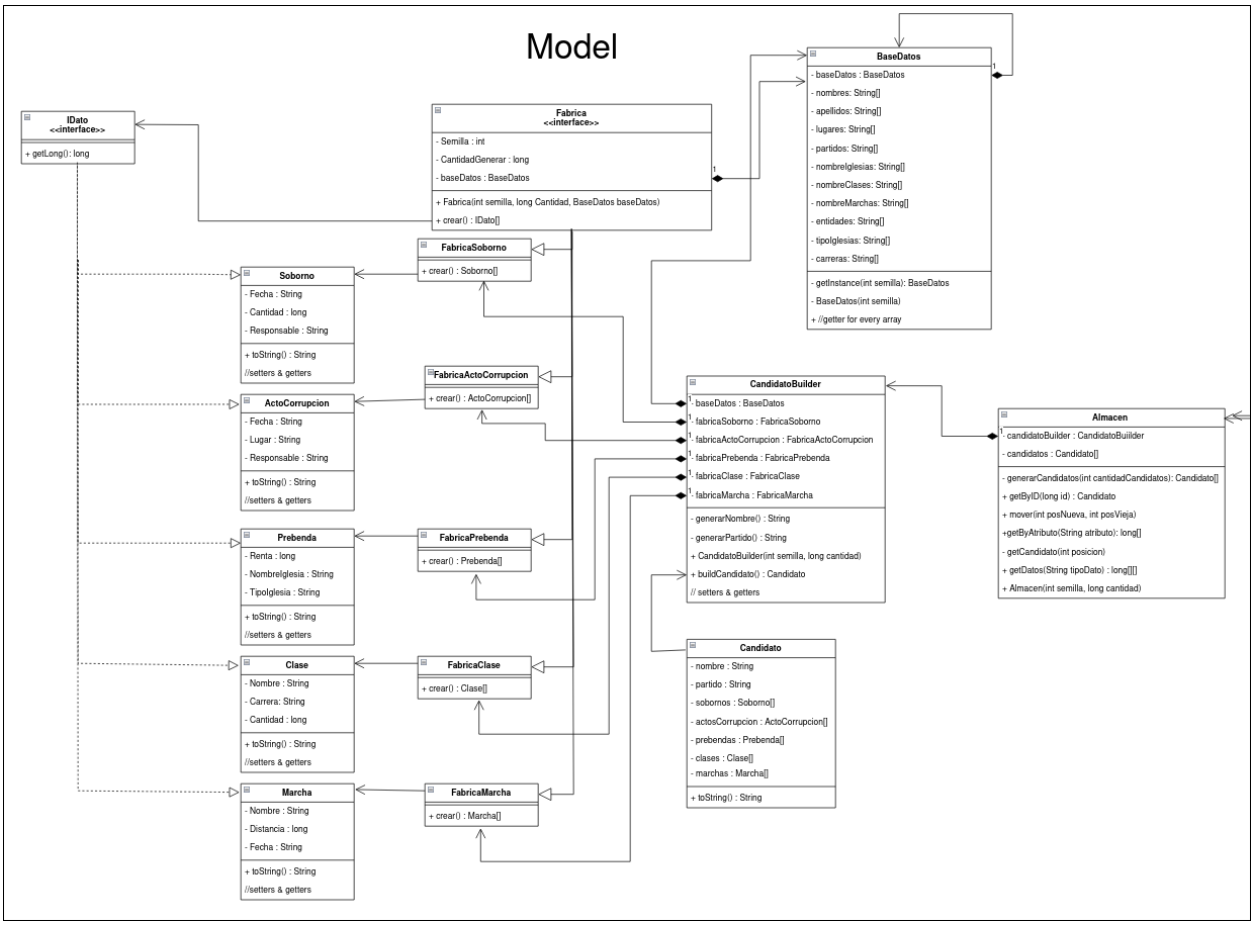
Con esta finalidad, se plantea el uso de los siguientes 5 algoritmos de ordenamientos:

- Burbuja
- Inserción
- Selección
- Merge
- Quick

2. Planteamiento de problema

Para resolver este problema se plantea el uso de una arquitectura Modelo-Vista-Controlador escrito en el lenguaje JAVA, dividiendo las necesidades de la solución como se muestra a continuación:

- Modelo



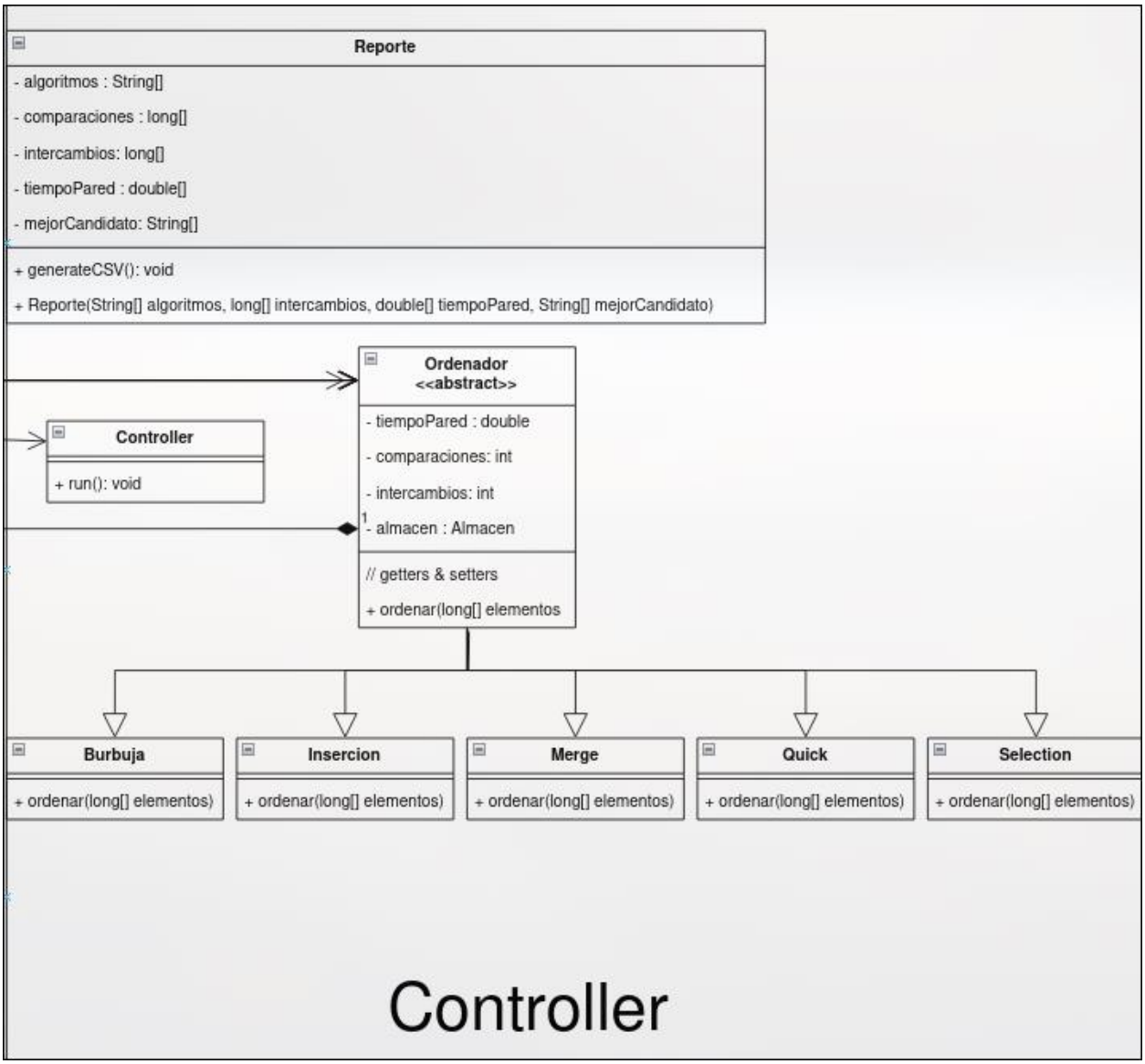
El paquete modelo partiendo del objetivo de almacenar las entidades del problema, se plantea el uso del patrón de diseño *abstract factory* y *factory method,* para la creación de los atributos de los candidatos dado que son objetos complejos y esta última entidad debe almacenar un arreglo de cada atributo; al hacer uso de estos patrones de diseño se facilita la organización de la creación del objeto candidato a su vez que remarca el principio SOLID de *single-responsability*, cada fabrica está hecha para generar un tipo de atributo.

Posteriormente para la creación del objeto candidato se basa en el patrón de diseño *builder* dado que el candidato es un objeto compuesto por otros, los anteriormente mencionados atributos.

Además para la generación de estos atributos se hizo uso de la clase BaseDatos cuyo diseño se basa en el patrón *singleton*, su implementación surge de la necesidad de cada fabrica y la clase CandidatoBuilder de adquirir datos generados automáticamente para los candidatos, de no aplicarse el patrón *singleton*, se generaría en memoria un objeto de este tipo para cada una de las clases mencionadas aumentando el consumo de memorias lo cual debe ser evitado todo lo posible dado la necesidad de generar la mayor cantidad posible de candidatos.

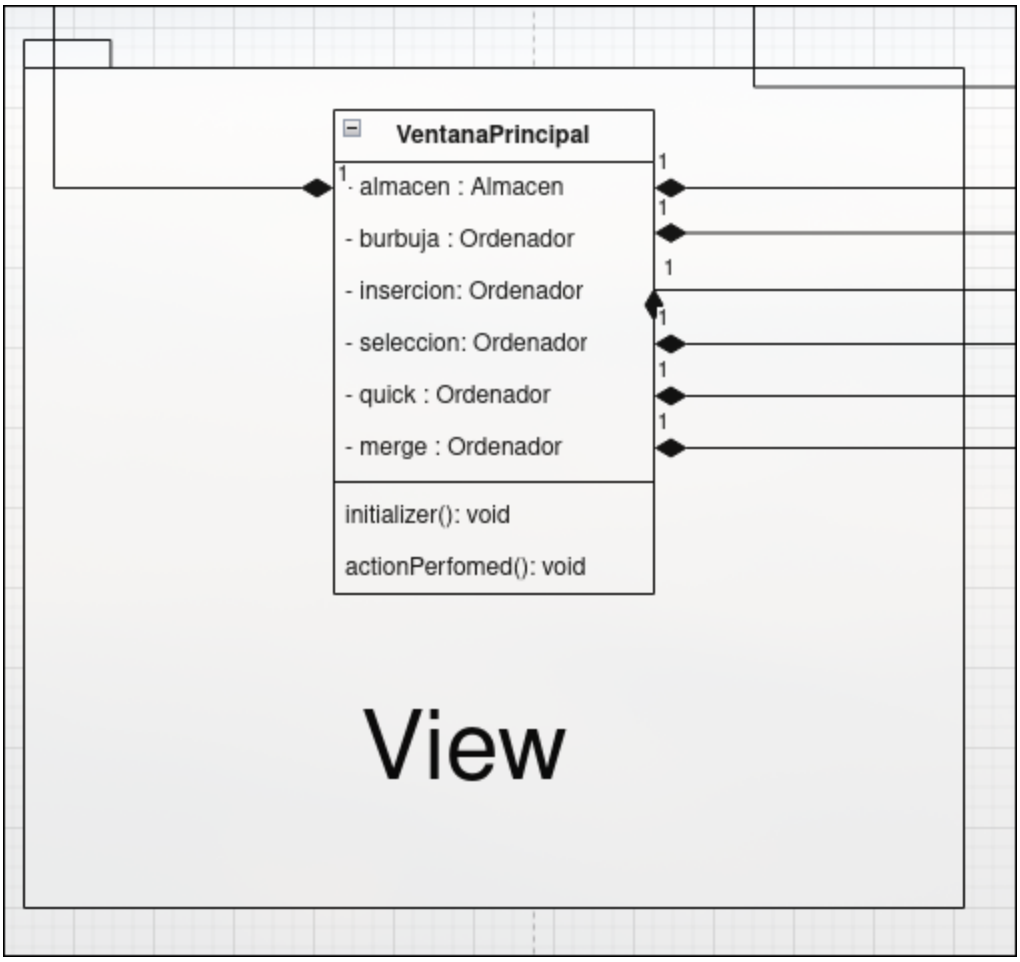
- Controlador

En el apartado del controlador partiendo del objetivo de almacenar los objetos que manipulen el modelo, se generó el siguiente diagrama:

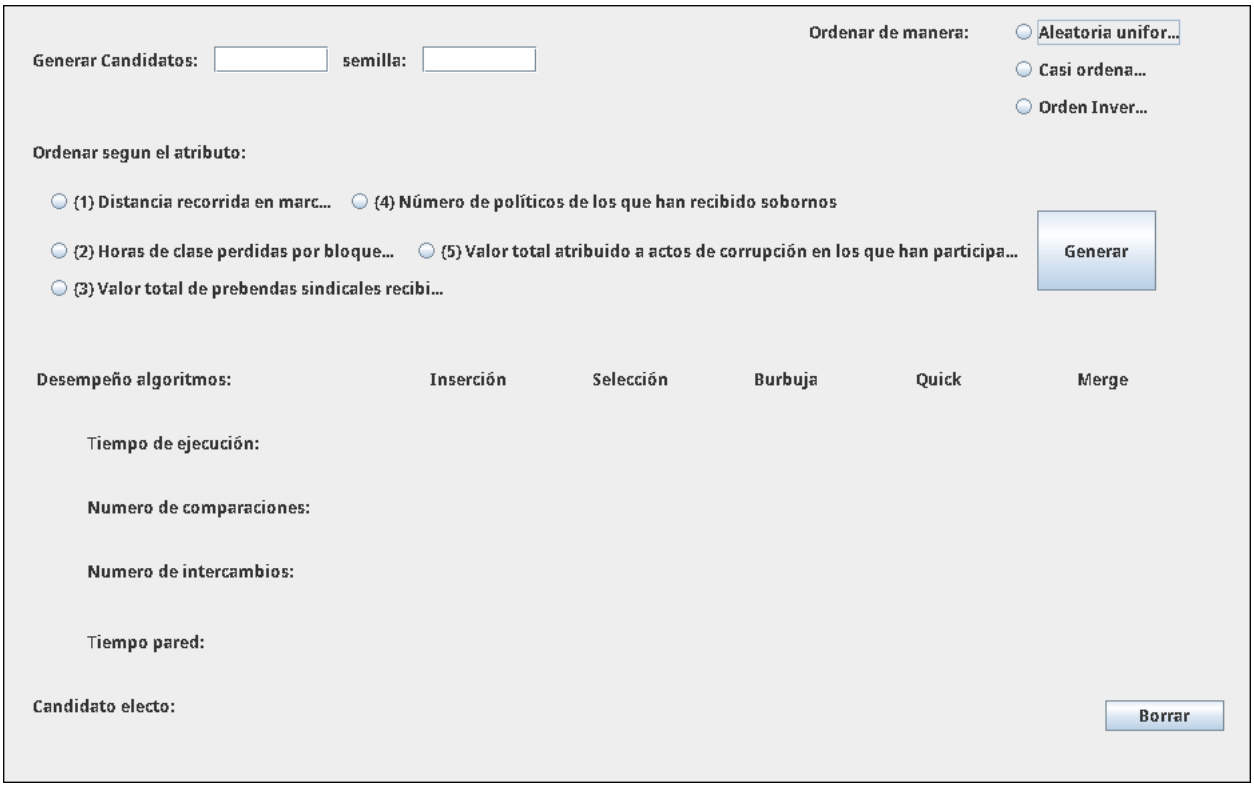


Siguiendo con los requerimientos planteados, se genera una clase Ordenador por la cual se heredan todas las clases que contienen los algoritmos de ordenamiento, estas manipulan el modelo por medio de la clase Almacén y dado que se requiere la generación de csv con la información creada, esta responsabilidad la contiene la clase Reporte; por último, la clase C*ontroller* es la encargada de ejecutar todo el programa por medio de la función run().

- Vista

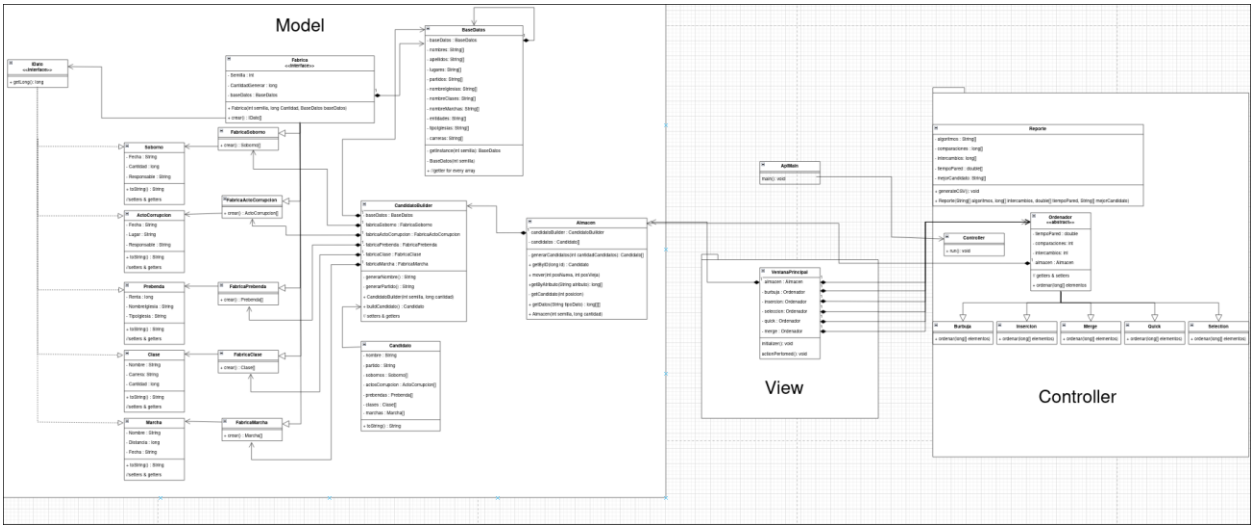


Dentro de la vista se almacena las clases de más alto nivel tanto del modelo como del controlador con la finalidad de facilitar la información presentada en la siguiente interfaz gráfica:



Ademas se incluye métricas de tiempo de ejecución, comparación, intercambio y tiempo pared de cada algoritmo por los siguientes objetivos.

- Diagrama de clases MVC



3. Objetivo General

    - Determinar el desempeño de los distintos algoritmos de ordenamiento

4. Objetivos específicos

    - Determinar la eficiencia relativa en arreglos pequeños vs. grandes
    - Analizar el efecto del orden inicial
    - Determinar el impacto de la distribución

5. Métricas
   Con la finalidad de comprender la eficiencia de cada algoritmo y como se presentó en el diagrama de clases, se hace uso de las siguientes métricas:

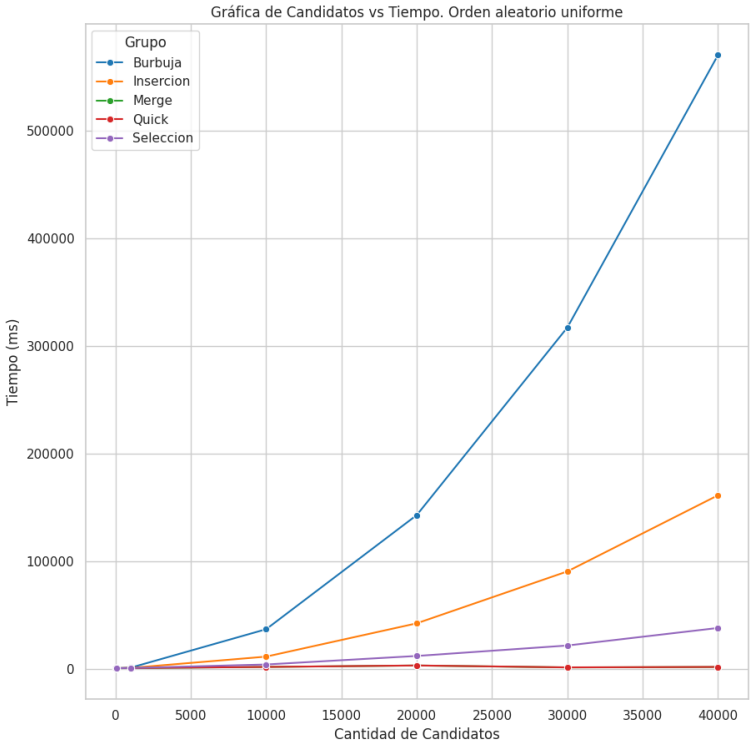
    - Tiempo ejecución: Intervalo de tiempo desde el inicio hasta la finalización de la ejecución del programa,
    - Tiempo pared: Tiempo que toma desde el inicio de la ejecución del algoritmo hasta su finalización, esta diferencia con el tiempo de ejecución nos permite descartar el tiempo utilizado para la generación de datos y objetos.
    - Comparaciones: La cantidad de comparaciones realizadas por el algoritmo, a nivel de la maquina se podría indicar como la cantidad de condicional if realizados.
    - Intercambio: La cantidad de modificaciones realizadas por el algoritmo en el arreglo.

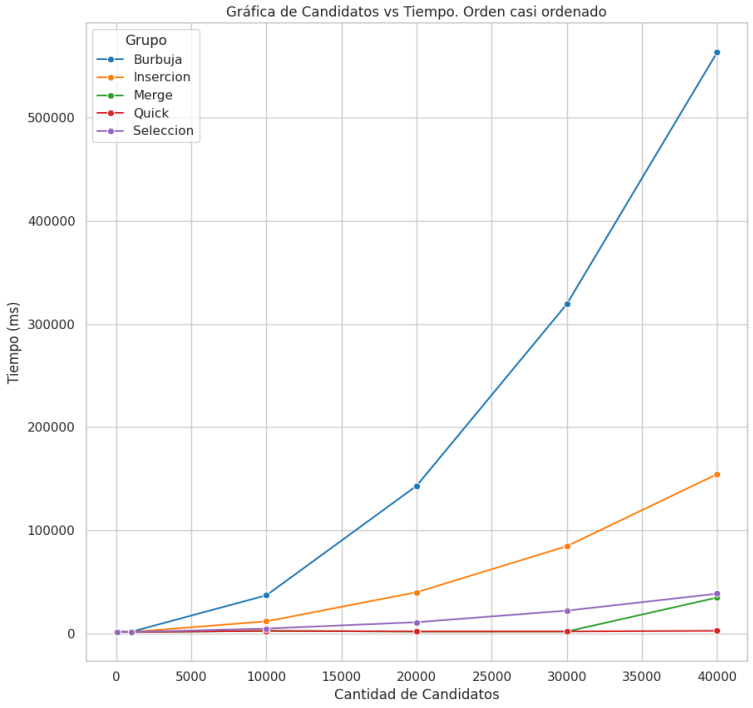
Además, se utiliza una semilla para la generación de números pseudoaleatorios para generar resultados replicables y generar contrastes por ejemplo en el orden inicial de los elementos, partiendo de aleatoriamente uniforme, casi ordenado y orden inverso.

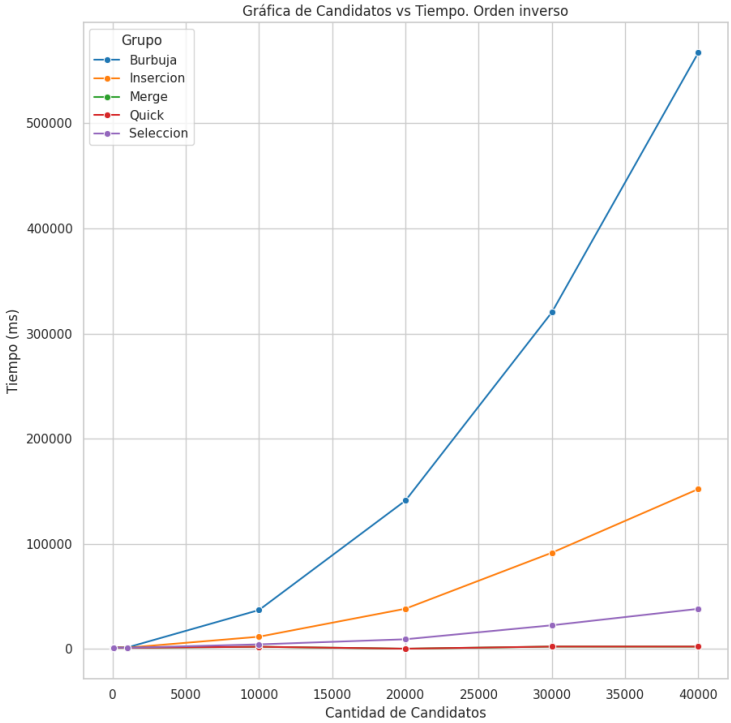
6. Resultados

Los resultados que se presentan a continuación se generaron partiendo de la semilla = 123 y variando la cantidad de candidatos de = {10, 100, 1000, 10000, 20000, 30000, 40000}.

    - Comparación de tiempo pared variando el orden inicial


Gráfica de Candidatos vs Tiempo. Orden aleatorio uniforme
Grupo
Burbuja
Insercion
Merge
Quick
Seleccion
Tiempo (ms)
0
100000
200000
300000
400000
500000
0
5000
10000
15000
20000
25000
30000
35000
40000
Cantidad de Candidatos


Gráfica de Candidatos vs Tiempo. Orden casi ordenado
Grupo
Burbuja
Insercion
Merge
Quick
Seleccion
Tiempo (ms)
0
100000
200000
300000
400000
500000
0
5000
10000
15000
20000
25000
30000
35000
40000
Cantidad de Candidatos



Estos resultados muestran la posible curva de complejidad temporal, el tiempo requerido según la enésima cantidad de elementos del arreglo para ordenarlo, siendo aproximadamente:

Burbuja: O(n*n)

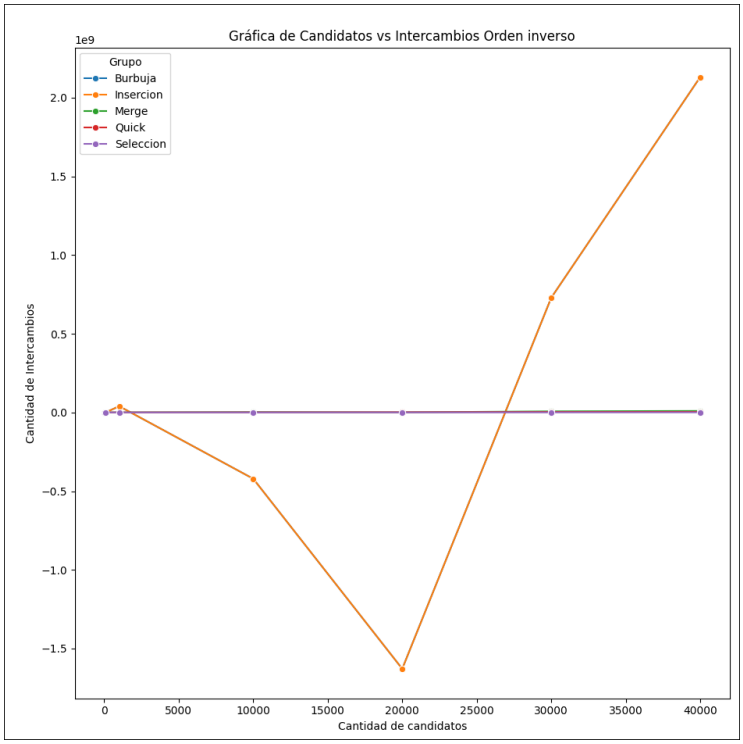
Inserción: O(n)

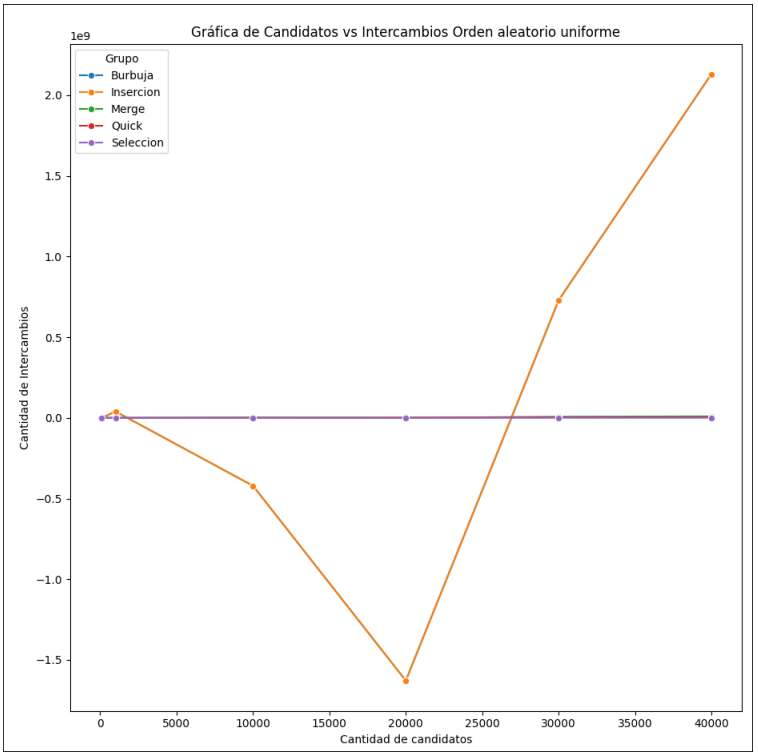
Merge : O(log(n))

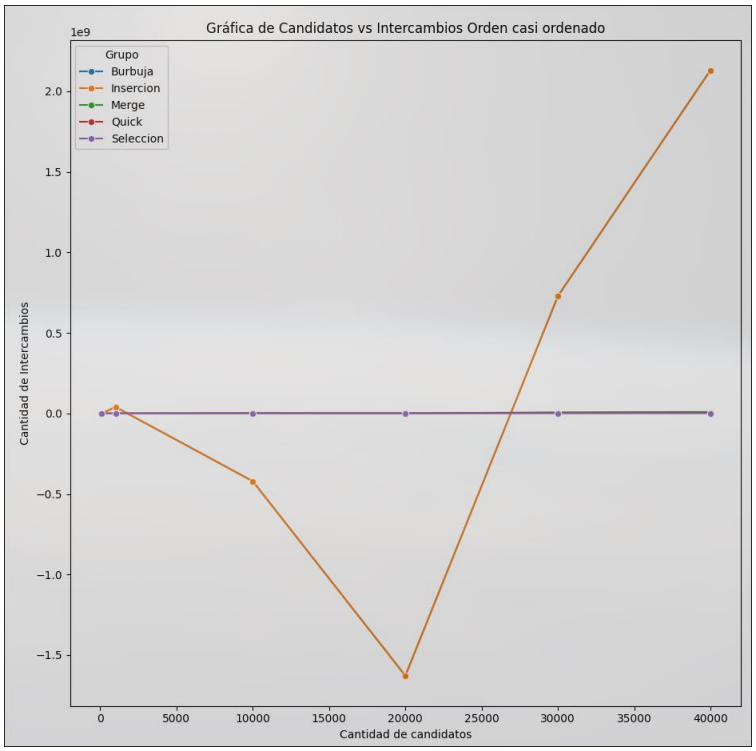
Selección: O(log(n))

Quick: $O(\log_2(n))$

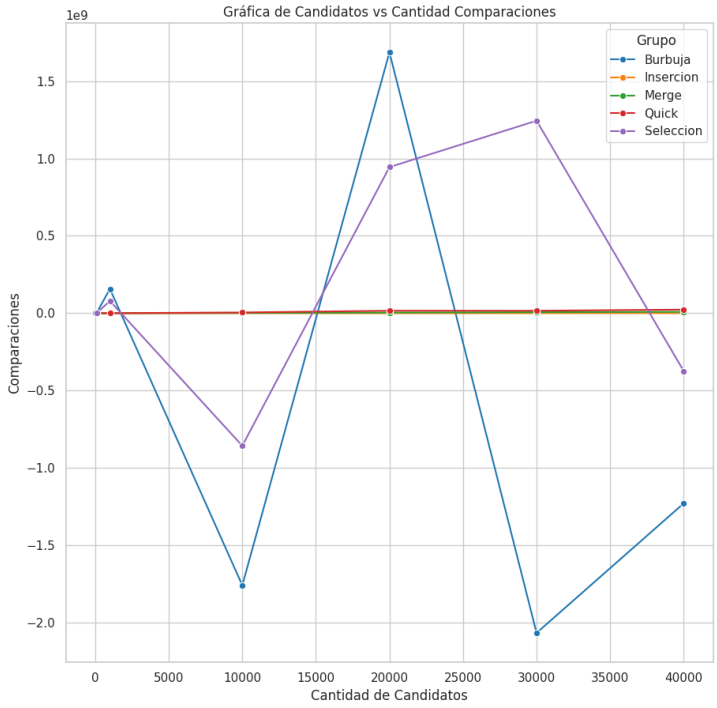
- Comparación de cantidad de intercambios variando cantidad de usuarios

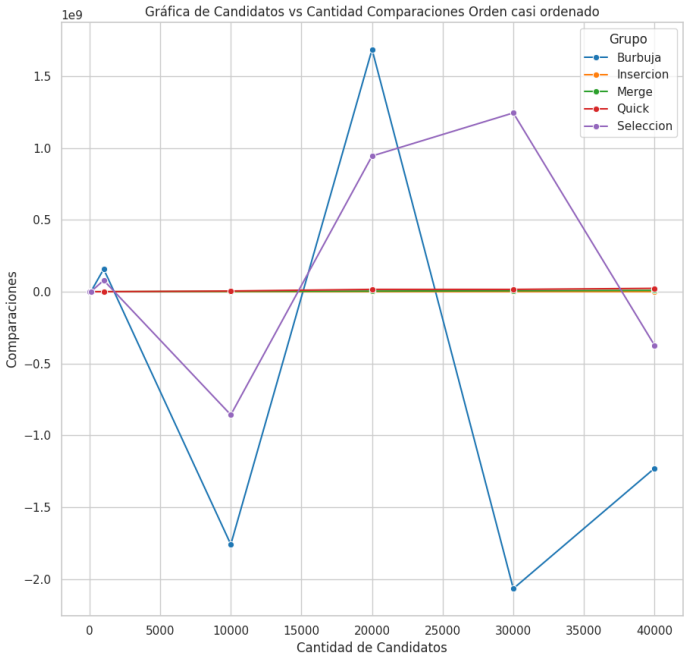

Gráfica de Candidatos vs Intercambios Orden inverso
1e9
Grupo
Burbuja
Insercion
Merge
Quick
Seleccion
Cantidad de intercambios
Cantidad de candidatos

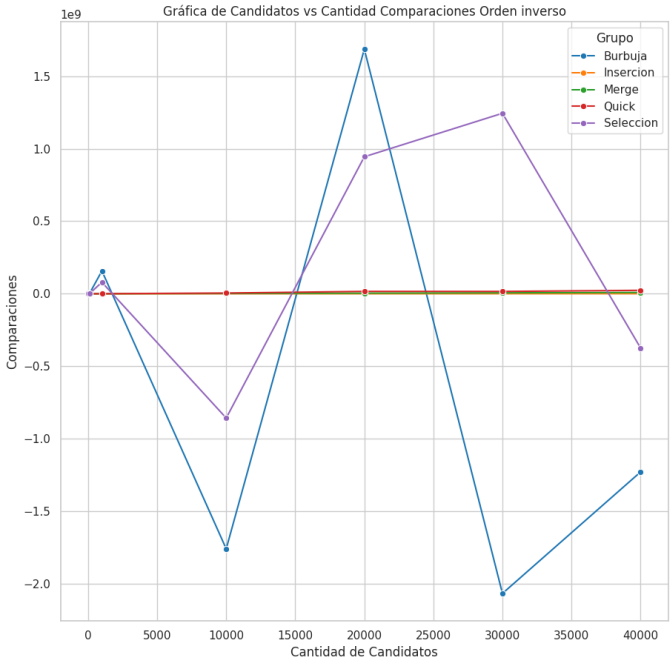

Gráfica de Candidatos vs Intercambios Orden aleatorio uniforme
1e9
Grupo
Burbuja
Insercion
Merge
Quick
Seleccion
Cantidad de intercambios
Cantidad de candidatos



- Comparación de cantidad de comparaciones variando cantidad de candidatos







La cantidad de comparaciones e intercambios conforme aumenta la cantidad de candidatos superan la longitud del tipo de dato *long*, por esta razón se representa erróneamente estas cantidades llegando a negativos, sin embargo, demuestra que aquellos

algoritmos que aparentan una línea recta en los gráficos, debido a la gran variación con los otros algoritmos, demuestran mayor eficiencia.

7. Conclusión

A partir de los resultados se evidencia la similitud en las métricas sobre el desempeño de cada algoritmo al manejar pequeñas cantidades de datos, marcando una desviación significativa a partir de los 10 mil datos, por consiguiente, el uso de algoritmos de ordenamiento clásicos como burbuja, selección e inserción, dado lo intuitivos de sus algoritmos, podrían preferirse para la creación de programas con manejo de pequeñas cantidades de datos y disminuir la complejidad de estos. Sin embargo, al aumentar la cantidad de datos, es decir, realizando una prueba de estrés al equipo, se evidencia una clara diferencia de desempeño entre los algoritmos clásicos y modernos, siendo la opción realista para el manejo de grandes cantidades de datos. Aunque la eficiencia de memoria se podría evidenciar con la cantidad de intercambios, dado que la generación de un intercambio requiere de la generación temporal de un objeto, no se consideró como criterio decisivo para esta conclusión ya que durante la ejecución de los algoritmos, se evidencio una incapacidad de procesar cantidades mayores a 40.000 candidatos a causa de la memoria, implicando con mayor seguridad un problema en la optimización del gasto en memoria de la arquitectura planteada más allá del rendimiento de los algoritmos.

El efecto de la distribución y el orden inicial de los datos no aparenta una repercusión a considerar en el desempeño de los algoritmos, manteniendo la cantidad de comparaciones e intercambios estables y un ligero cambio en los tiempos de ejecución, aumentando el comportamiento exponencialmente de la complejidad tiempo de los algoritmos de ordenamiento clásicos.