OBSERVACIONES DE LA PRÁCTICA

Samuel Villamil Alvarez - 202421118
Juan Diego García – 202423575
Tomas Aponte - 202420148

Ambientes de pruebas

	Máquina 1	**Máquina 2**	**Máquina 3**
Procesadores	Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz 2.30 GHz	AMD Ryzen 7 PRO 2700U w/ Radeon Vega Mobile Gfx	AMD Ryzen 5 3600 3,6 GHz – 4,2GHz
Memoria RAM (GB)	16,0 GB (15,7 GB usable)	16.0 GB (14.9 GB usable)	16 GB (15,7 GB usable)
Sistema Operativo	Windows 11	Windows 10	Windows 11

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Máquina 1

Resultados para Insertion Sort

TIEMPOS DE EJECUCIÓN INSERTION SORT[ms]			
Porcentaje de la muestra [pct]	**Tamaño de la muestra (ARRAY_LIST)**	**Array List**	**Linked List**
0,50%	50,00	**1.072**	**1.055**
5,00%	500,00	**115.264**	**1.342**
10,00%	1000,00	**273.724**	**1.812**
20,00%	2000,00	**1.123.871**	**1.964**
30,00%	3000,00	**3.523.236**	**2.652**
50,00%	5000,00	**6.549.118**	**6.092**
80,00%	8000,00	**18.359.474**	**4.060**
100,00%	10000,00	**34.499.564**	**10.333**

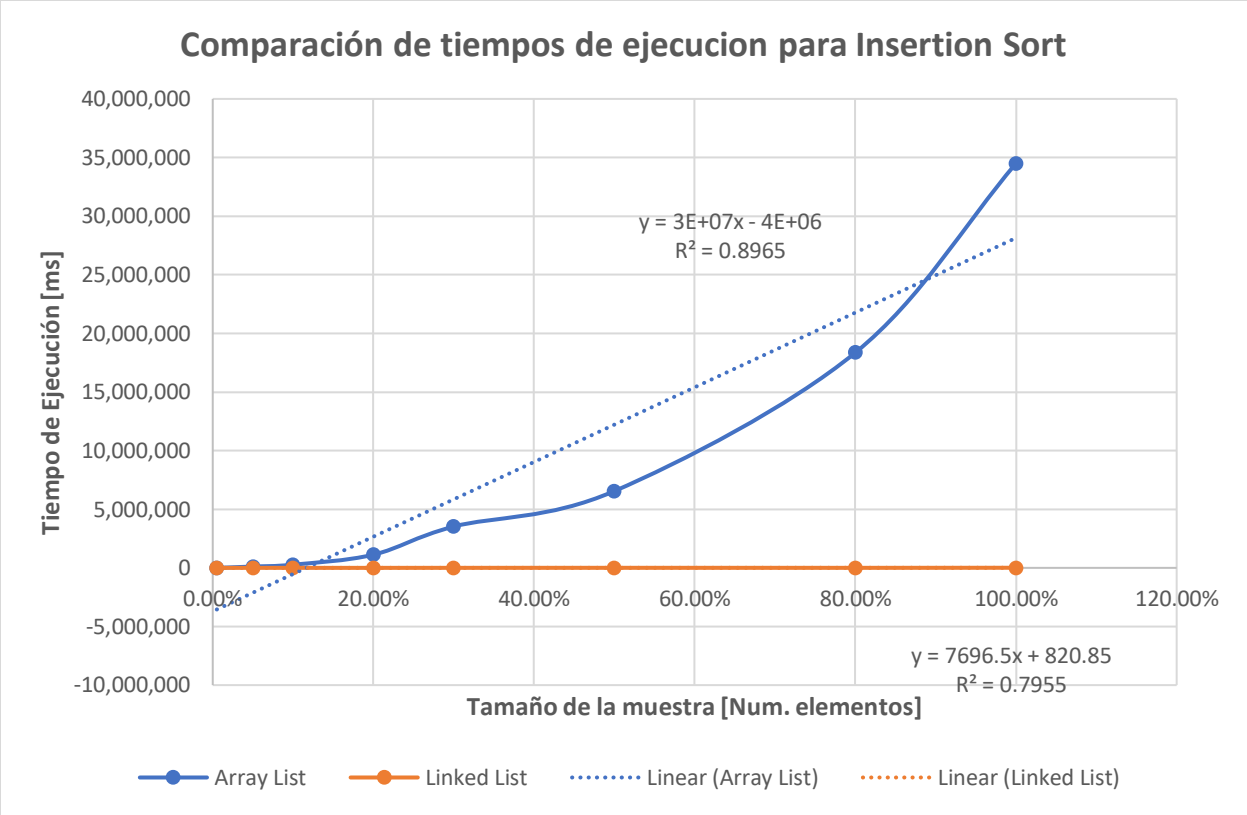


Tabla 2. ***Resultados máquina 1 para insertion Sort***

Resultados para Selection Sort

TIEMPOS DE EJECUCIÓN SELECTION SORT[ms]			
Porcentaje de la muestra [pct]	**Tamaño de la muestra (LINKED_LIST)**	**Array List**	**Linked List**
0,50%	50,00	**1.638**	**1.044**
5,00%	500,00	**136.477**	**1.212**
10,00%	1000,00	**394.010**	**1.641**
20,00%	2000,00	**1.124.549**	**1.064**
30,00%	3000,00	**2.489.934**	**1.275**
50,00%	5000,00	**7.000.043**	**0.963**
80,00%	8000,00	**29.702.695**	**1.060**
100,00%	10000,00	**40.013.216**	**1.064**

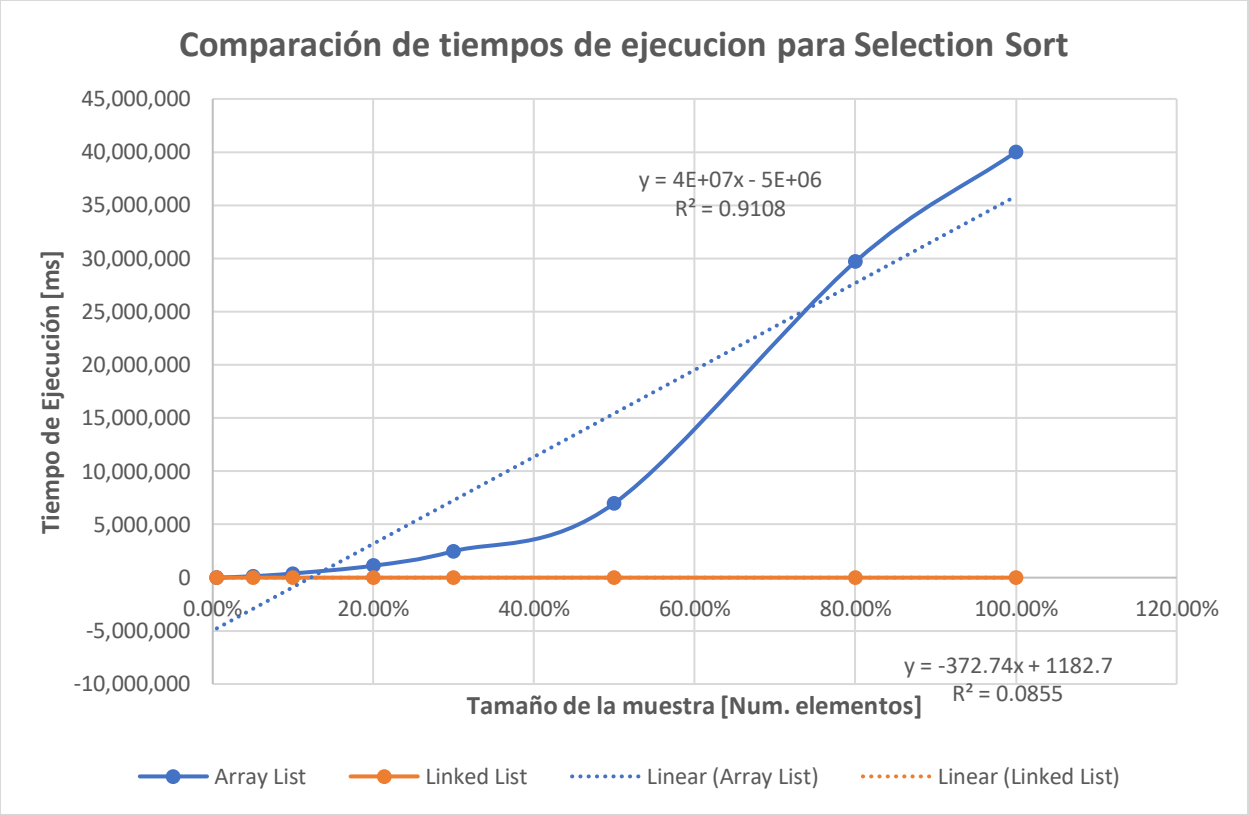


Tabla 3. ***Resultados máquina 1 para Selection Sort***

Resultados para Shell Sort

TIEMPOS DE EJECUCIÓN SHELL SORT[ms]			
Porcentaje de la muestra [pct]	**Tamaño de la muestra (especificar array o linked)**	**Array List**	**Linked List**
0,50%	50,00	**1.775**	**3.094**
5,00%	500,00	**86.800**	**3.876**
10,00%	1000,00	**388.982**	**3.430**
20,00%	2000,00	**1.018.589**	**6.291**
30,00%	3000,00	**2.231.269**	**6.424**
50,00%	5000,00	**6.582.562**	**6.390**
80,00%	8000,00	**18.741.444**	**3.399**
100,00%	10000,00	**38.876.693**	**3.387**

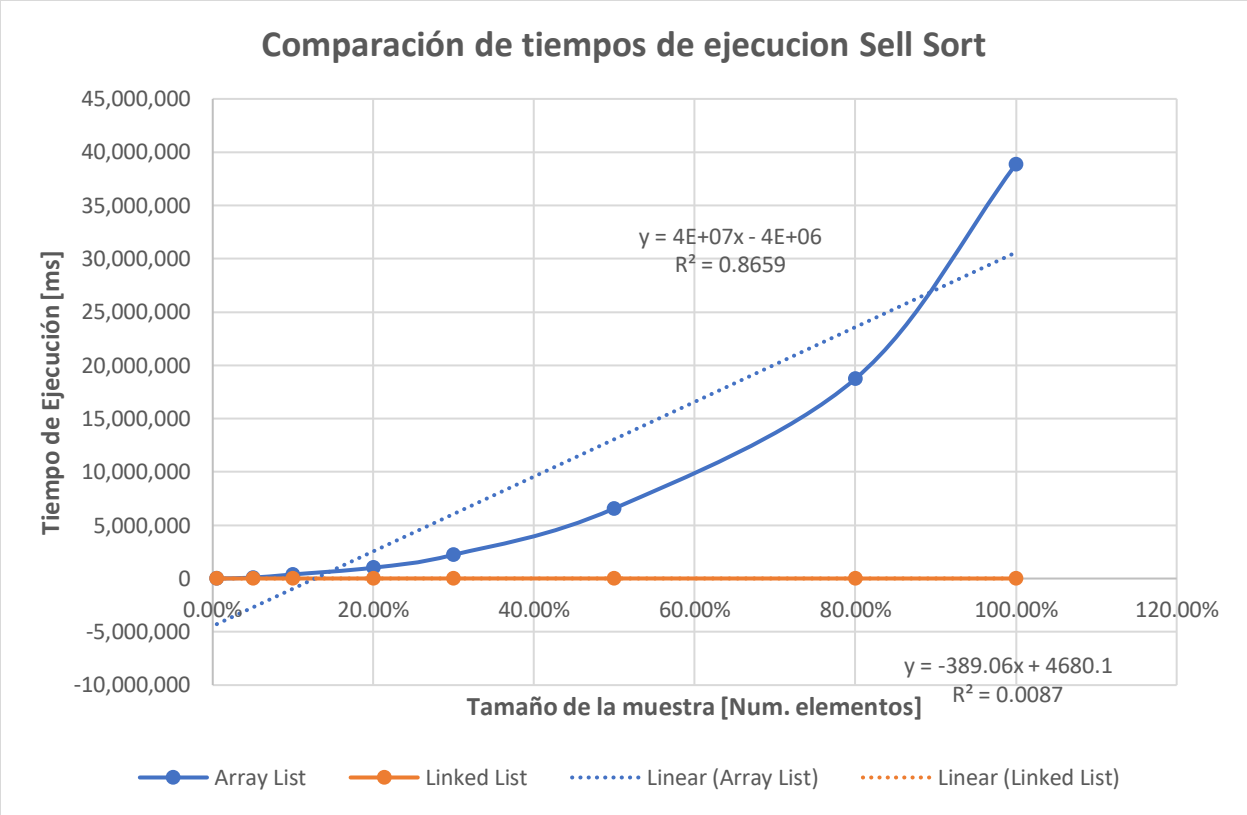
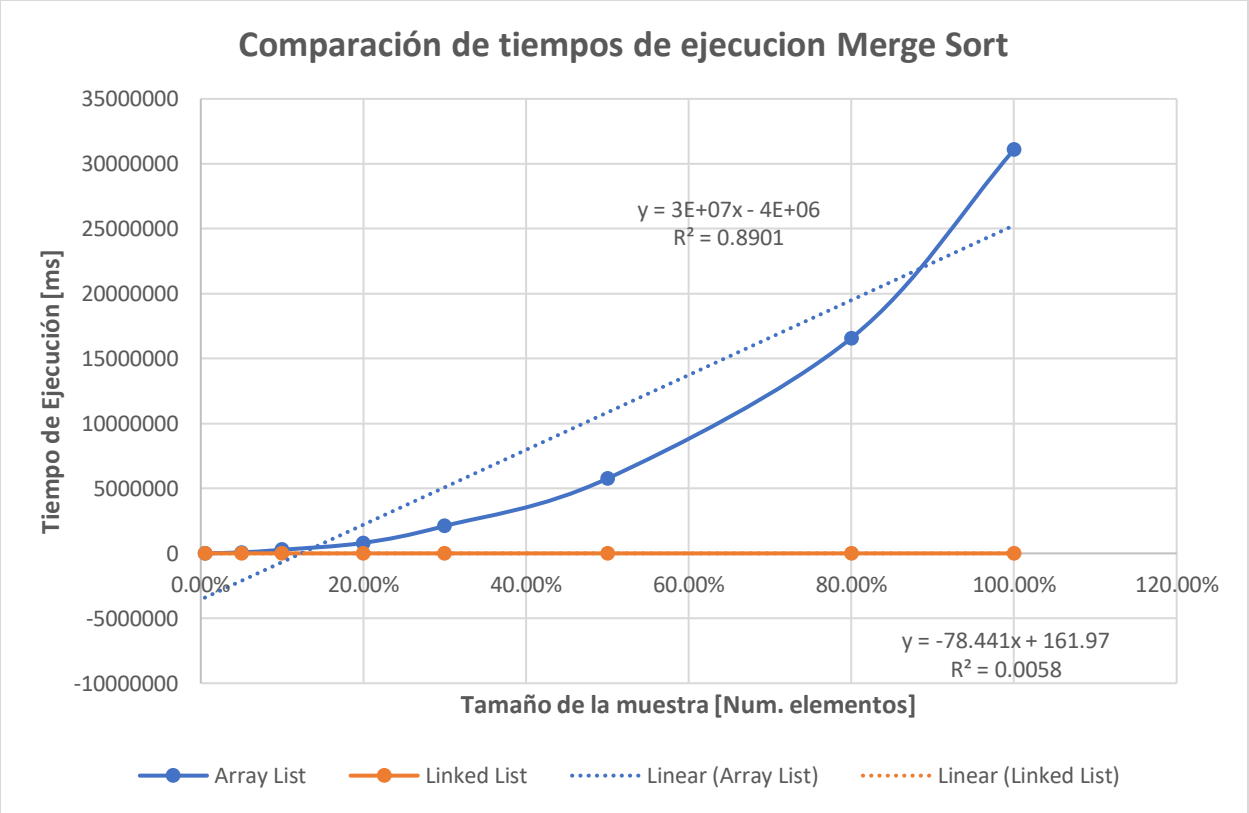


Tabla 4. ***Resultados máquina 1 para Shell Sort***

Resultados para Merge Sort

TIEMPOS DE EJECUCIÓN MERGE SORT[ms]			
Porcentaje de la muestra [pct]	**Tamaño de la muestra (LINKED_LIST)**	**Array List**	**Linked List**
0,50%	50,00	**0.995**	**0.954**
5,00%	500,00	**77.611**	**0.536**
10,00%	1000,00	**280.858**	**0.563**
20,00%	2000,00	**812.573**	**0.537**
30,00%	3000,00	**2.112.914**	**1.064**
50,00%	5000,00	**5.762.327**	**0.584**
80,00%	8000,00	**16.584.472**	**0.586**
100,00%	10000,00	**31.087.177**	**0.692**

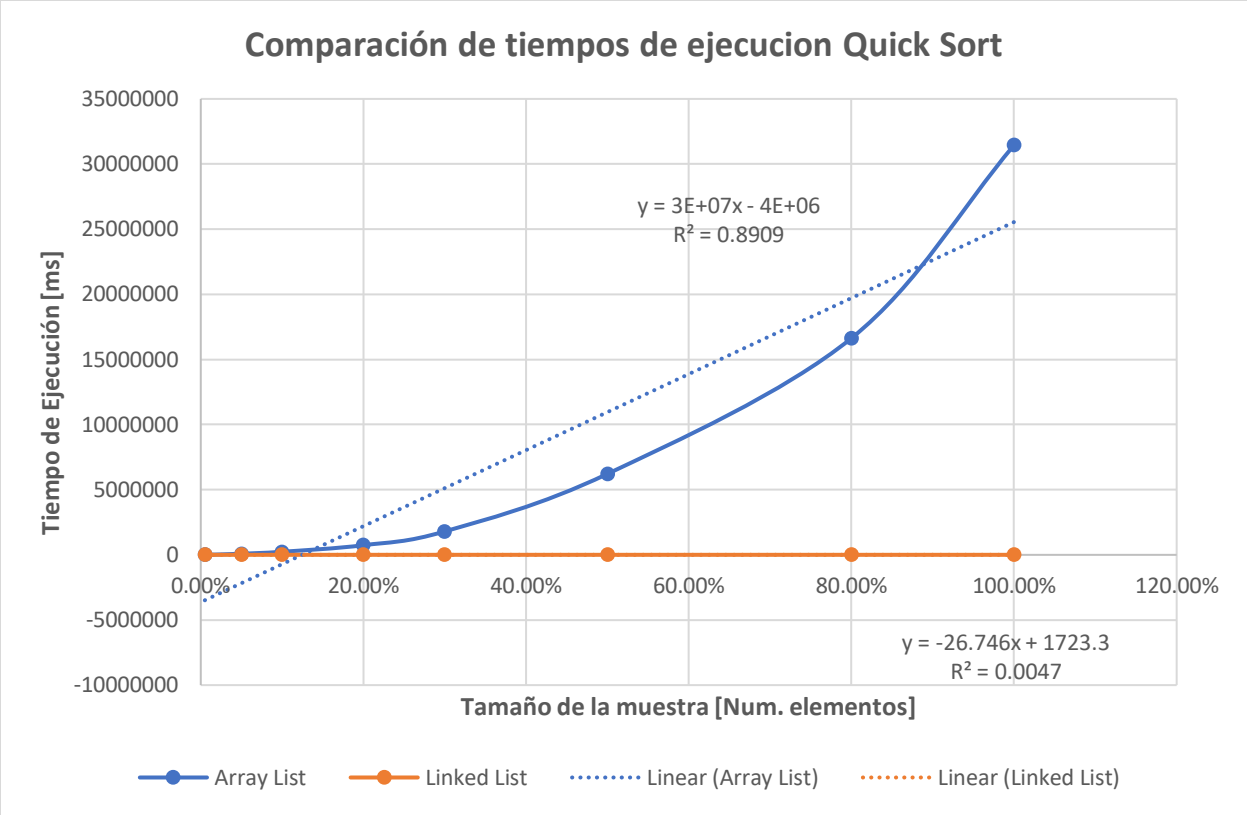
Tabla 5. ***Resultados máquina 2 para insertion Sort***



Resultados para Quick Sort

TIEMPOS DE EJECUCIÓN QUICK SORT[ms]			
Porcentaje de la muestra [pct]	**Tamaño de la muestra (LINKED_LIST)**	**Array List**	**Linked List**
0,50%	50,00	**0.760**	**1.491**
5,00%	500,00	**88.084**	**2.016**
10,00%	1000,00	**237.313**	**1.710**
20,00%	2000,00	**739.908**	**1.678**
30,00%	3000,00	**1.798.397**	**1.722**
50,00%	5000,00	**6.214.284**	**1.692**
80,00%	8000,00	**16.625.954**	**1.688**
100,00%	10000,00	**31.445.595**	**1.710**

Tabla 6. ***Resultados máquina 2 para Selection Sort***



Preguntas de análisis

1) **¿El comportamiento de los algoritmos es acorde a lo enunciado teóricamente?**

   No del todo, funciona perfectamente para array list, los resultados muestran un comportamiento entre cuadrático y lineal (a nuestro juicio más cuadrático, según Excel más lineal). Con un retorno correcto de los libros organizados por su rating promedio.

   El problema viene con SLL, que debería también tener un comportamiento similar, pero por alguna razón el programa lo está computando de manera constante. O encontramos un nuevo y perfeccionado método de ordenamiento o algo está mal cuando se está midiendo el tiempo. Lo que no llegamos a entender es: ¿Por qué si es así, el retorno es (a simple vista) correcto?

2) **¿Cuál Estructura de Datos es mejor utilizar si solo se tiene en cuenta los tiempos de ejecución de los algoritmos?**

   Tenemos dos conclusiones:

   a) Si nos guiamos por la teoría: Para los ordenamientos iterativos es mejor usar array list, al tener un índice y accesos directos, no sólo la implementación es más intuitiva y fácil, sino que estos ordenamientos son bien dependientes del índice, y cuando se adaptan a SSL hay que hacer acciones extra para adaptarlos. Los recursivos son mejores en SLL, no son tan dependientes

del índice, porque quick y merge lo que hacen es romper la lista y desde ahí reorganizar sin necesidad de recorrerla toda o de un índice.

b) Si nos guiamos según el tiempo conseguido: Para todas las funciones sería mejor utilizar SLL, pero este resultado muy posiblemente está defectuoso, por lo que no podemos concluir desde ahí.

3) ¿Cómo afecta el tamaño de los datos a la eficiencia de cada algoritmo en las diferentes estructuras de datos?

Los ordenamientos siempre tienen una eficiencia cuadrática (O(n^2)), el tamaño de los datos cambia el resultado abismalmente, y el tiempo de organización de estos se va haciendo cada vez mayor, con una pendiente cada vez más pronunciada. Sin importar la estructura, esta naturaleza se va a mantener.

4) ¿Cuál consideran que fue el mejor algoritmo iterativo y por qué?

Insertion sort. Al menos para array list tuvo la pendiente menos pronunciada, la más cercana que obtuvimos a una constante. Además, para las operaciones posteriores, como eliminar, añadir y buscar datos, array list siempre es muchísimo más manejable, que nos lleva a priorizar esta estructura de datos, y con ella los ordenamientos iterativos, y si de esos el más eficiente resultó ser Insertion, entonces ese fue el mejor.

5) Si quisieras ordenar una lista muy grande de datos, ¿qué algoritmo escogerías? ¿Por qué?

Depende del formato en que esté la lista. De ser un array list se utilizaría insertion sort, que ya se observó es la más eficiente para este tipo de listas. Para una SLL consideramos que merge sort puede ser un poco más eficiente que quick. En la documentación de la clase se nos explica que aunque el caso promedio de ambas es el mismo, el peor caso de merge es: O(n logn), que lo hace un poco más eficiente que quick sort: O(n^2). Por lo que escogeríamos merge para un volumen grande de datos.

6) Si quisieras ordenar una lista parcialmente ordenada, ¿cuál sería el mejor algoritmo?

Para una lista parcialmente ordenada puede ser mejor utilizar Shell. Al tomar valores separados por distancias específicas, puede ignorar grupos de valores que ya están organizados, y detectar anomalías o valores en desorden sin tener que recorrer la lista entera para hacerlo.