OBSERVACIONES DE LA PRÁCTICA

Ambientes de pruebas

	Máquina 1
Procesadores	12th Gen Intel® Core™ i5-12450H × 12
Memoria RAM (GB)	16
Sistema Operativo	Ubuntu 25.04

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

Ordenamientos Iterativos

Resultados para ordenamientos iterativos con Array List

Porcentaje de la muestra	Insertion Sort (Array List)	Selection Sort (Array List)	Shell Sort (Array List)
0.50%	0,342	0,246	0,217
5.00%	27,946	3,406	2,450
10.00%	97,364	9,999	5,554
20.00%	399,400	40,775	13,756
30.00%	931,771	77,002	24,830
50.00%	2634,739	169,800	44,822
80.00%	7173,521	394,351	70,260
100.00%	12075,863	588,156	90,611

Tabla 2. ***Resultados máquina 1 para oredenamientos iterativos con Array List.***

Resultados para ordenamientos iterativos con Linked List

Porcentaje de la muestra	Insertion Sort (Linked List)	Selection Sort (Linked List)	Shell Sort (Linked List)
0.50%	0,339	0,519	0,615
5.00%	33,793	47,249	124,654
10.00%	94,824	160,136	726,363
20.00%	391,051	655,538	4559,054
30.00%	918,793	1488,898	13722,682
50.00%	2613,323	4289,206	61388,365
80.00%	7216,697	11554,502	234149,501
100.00%	11970,927	18755,973	378520,135

Tabla 3. ***Resultados máquina 1 para oredenamientos iterativos con Single Linked List.***

Ordenamientos Recursivos

Resultados para ordenamientos recursivos con Array List

Porcentaje de la muestra	Merge Sort (Array List)	Quick Sort (Array List)
0.50%	0,217	0,189
5.00%	2,450	2,304
10.00%	5,554	6,078
20.00%	13,756	6,394
30.00%	24,830	9,815
50.00%	44,822	20,869
80.00%	70,260	42,858
100.00%	90,611	46,711

Tabla 4. ***Resultados máquina 1 para oredenamientos recursivos con Array List.***

Resultados para ordenamientos recursivos con Linked List

Porcentaje de la muestra	Merge Sort (Linked List)	Quick Sort (Linked List)
0.50%	0,222	0,202
5.00%	2,442	2,083
10.00%	5,605	2,826
20.00%	14,176	6,176
30.00%	23,415	8,952
50.00%	48,386	22,153
80.00%	70,450	35,390
100.00%	89,663	47,729

Tabla 5. ***Resultados máquina 1 para oredenamientos recursivos con Single Linked List.***

Comparación de tiempo mejores algoritmos de ordenamiento

Maquina 1

Porcentaje de la muestra	Algoritmo recursivo (Quick) Array List	Algoritmo iterativo (Shell) Array List
0.50%	0,189	0,246
5.00%	2,304	3,406
10.00%	6,078	9,999
20.00%	6,394	40,775
30.00%	9,815	77,002
50.00%	20,869	169,800
80.00%	42,858	394,351
100.00%	46,711	588,156

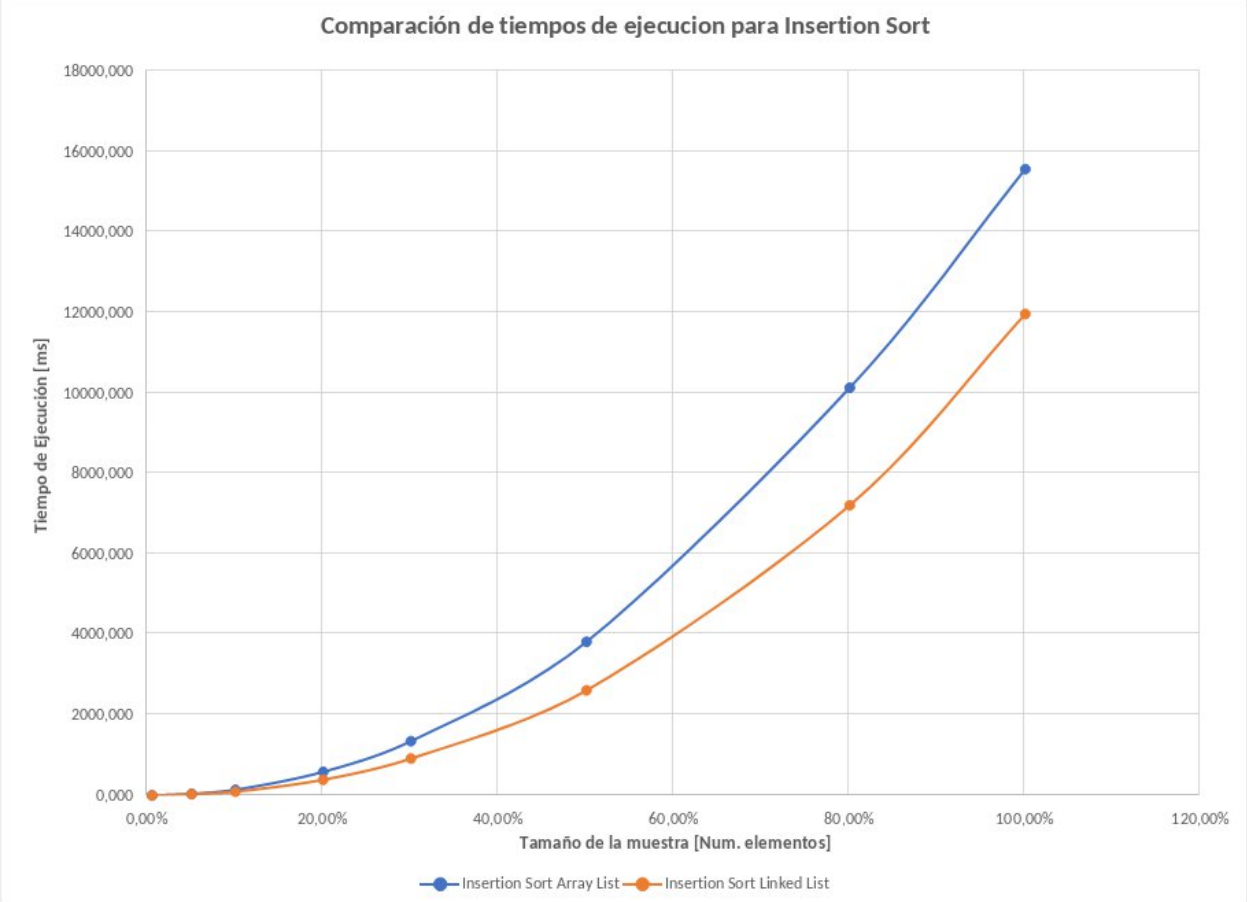
Tabla 6. ***Resultados máquina 1 mejores algoritmos***

Preguntas de análisis parte 1

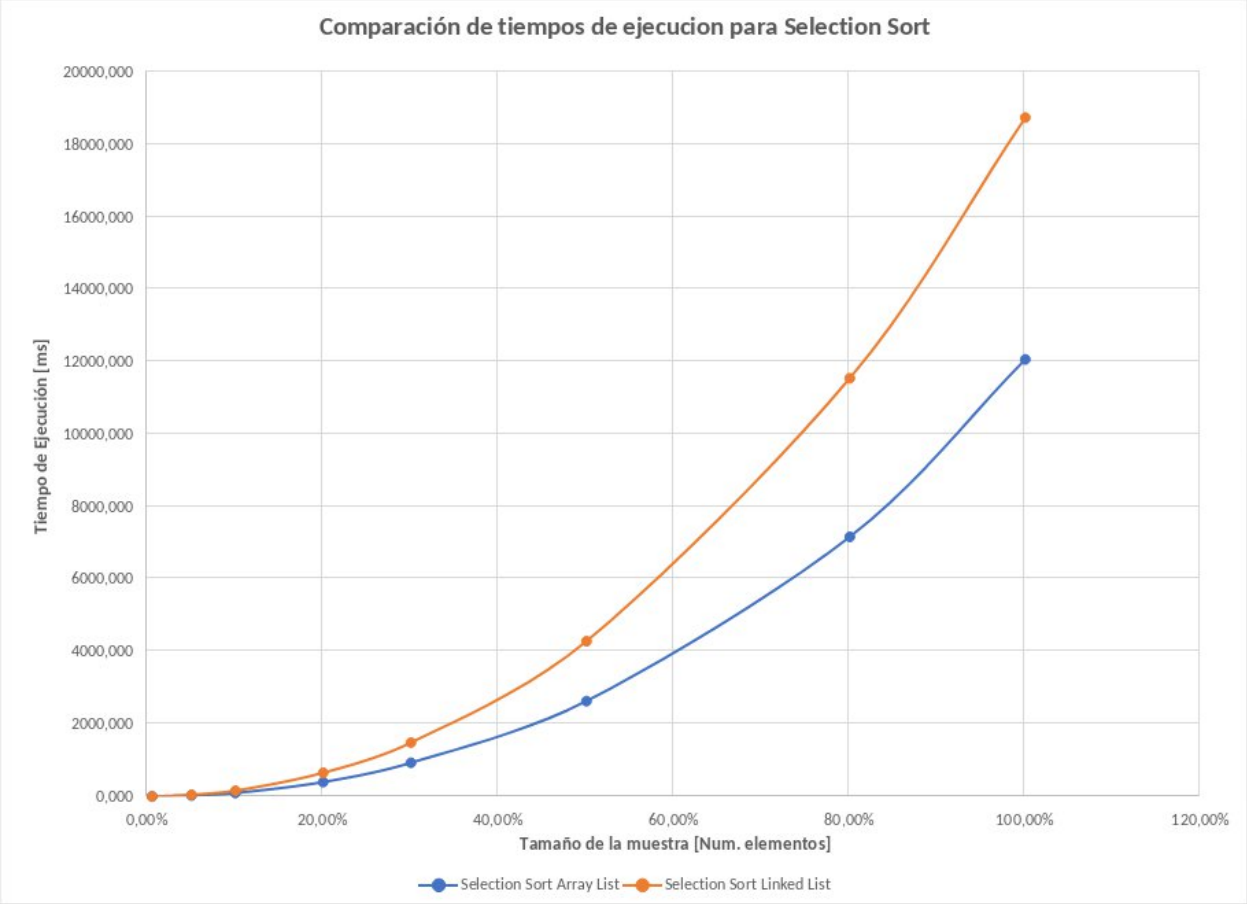
Nota: Se mencionarán los algoritmos de ordenamiento sin el sufijo Sort y empezando con mayúscula.

1. **¿Cómo varía el comportamiento de cada algoritmo con respecto al tamaño de los datos?**

   RTA:

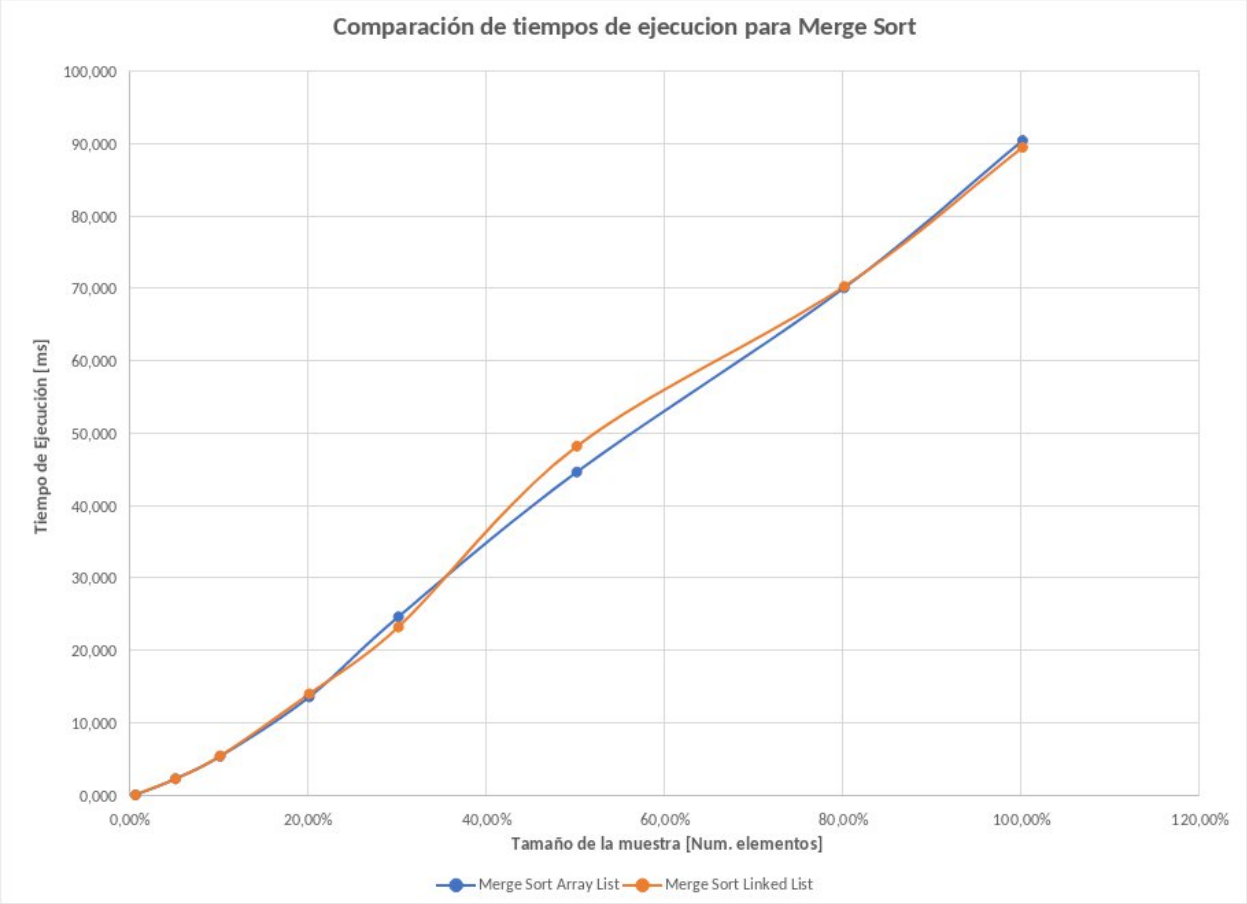


Insertion: El comportamiento es claramente O(n^2) para ArrayList y LinkedList.

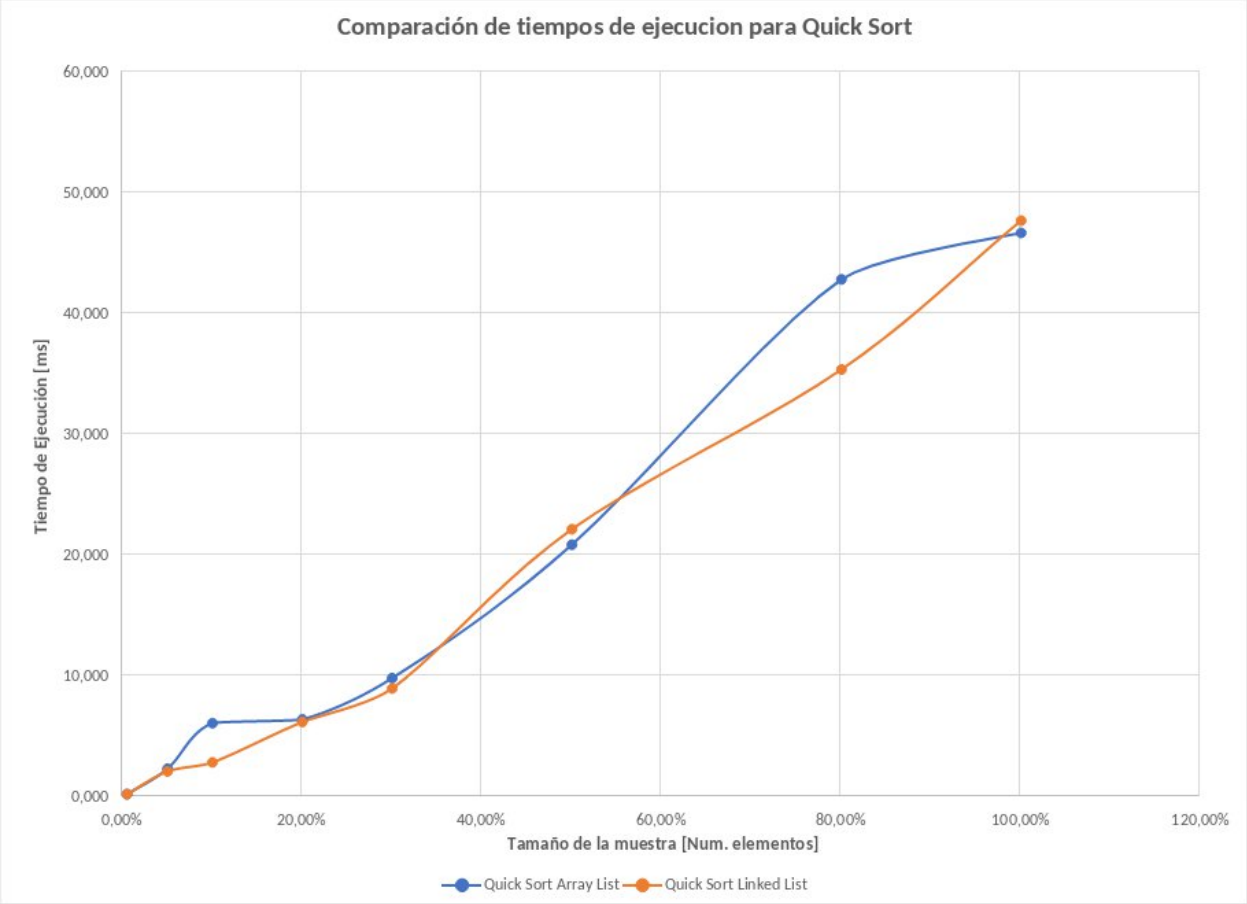


Selection: El comportamiento es claramente O(n^2) para ArrayList y LinkedList.

Shell: Evidentemente, la implementación es más eficiente en ArrayList. Lo anterior es esperado, ya que el algoritmo se basa en el intercambio de información entre gaps, lo cual solo es O(1) para ArrayList y O(n) para LinkedList, lo que implica que la complejidad de LinkedList es la de ArrayList multiplicada por n. Por otro lado, al comparar las razones entre el tiempo en 0.5% y 5%, 10% y 20%, 50% y 100%, se observa una razón aproximada a lo que se esperaría de una función con crecimiento O(n^3/2); por lo tanto, el orden de crecimiento es O(n^3/2), lo que es esperado, ya que el algoritmo se implementó con la secuencia de Knuth.



Merge: El comportamiento en ambas a primera vista es lineal, sin embargo al obtener las razones entre el tiempo en 0.5% y 5%,10% y 20%,50% y 100%; se observan las proporciones que se esperarían de una función con crecimiento O(nlogn), por lo tanto para ArrayList y LinkedList el orden de crecimiento es O(nlogn).



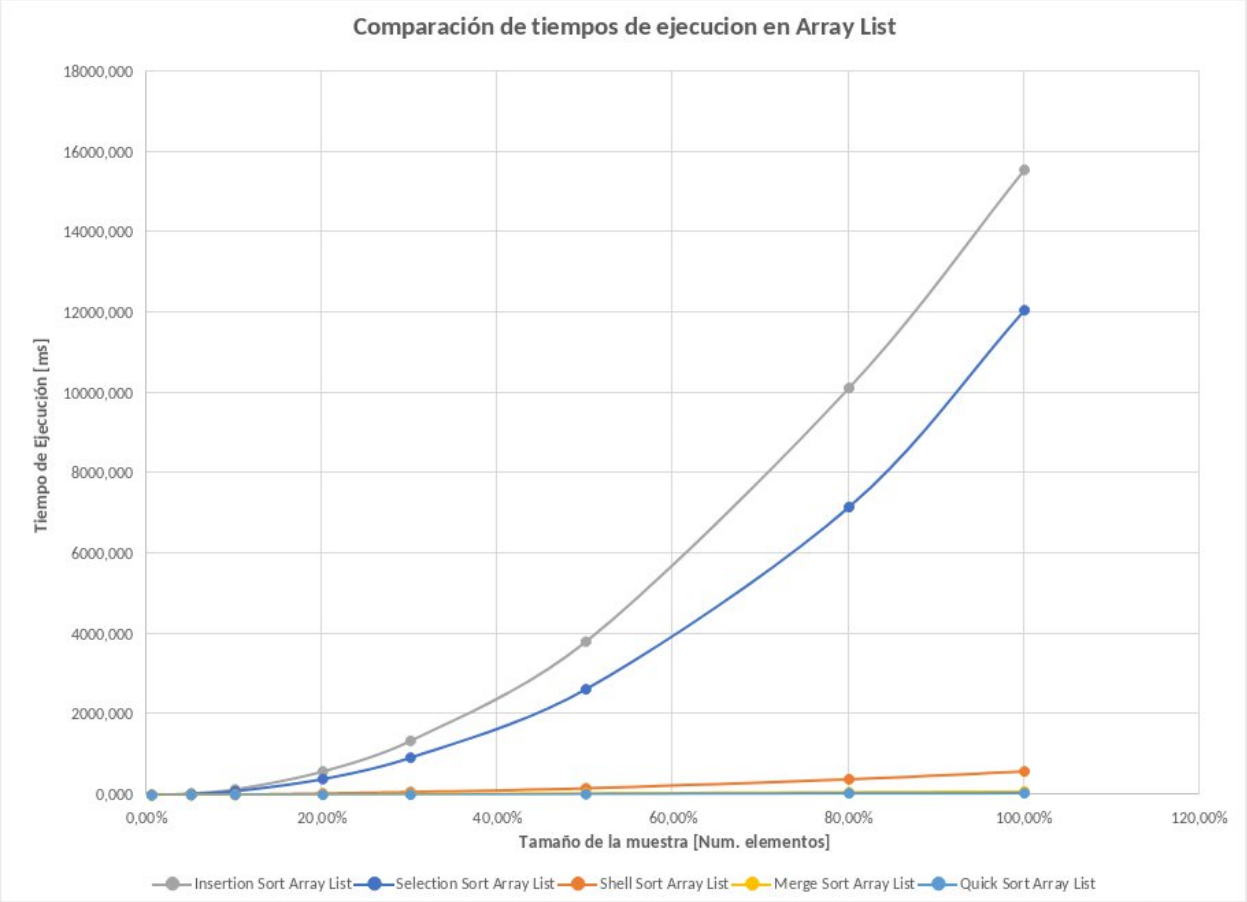
Quick: El comportamiento en ambas a primera vista es lineal; sin embargo, al obtener las razones entre el tiempo en 0.5% y 5%, 10% y 20%, 50% y 100%, se observan las proporciones que se esperarían de una función con crecimiento O(nlogn); por lo tanto, para ArrayList y LinkedList, el orden de crecimiento es O(nlogn).

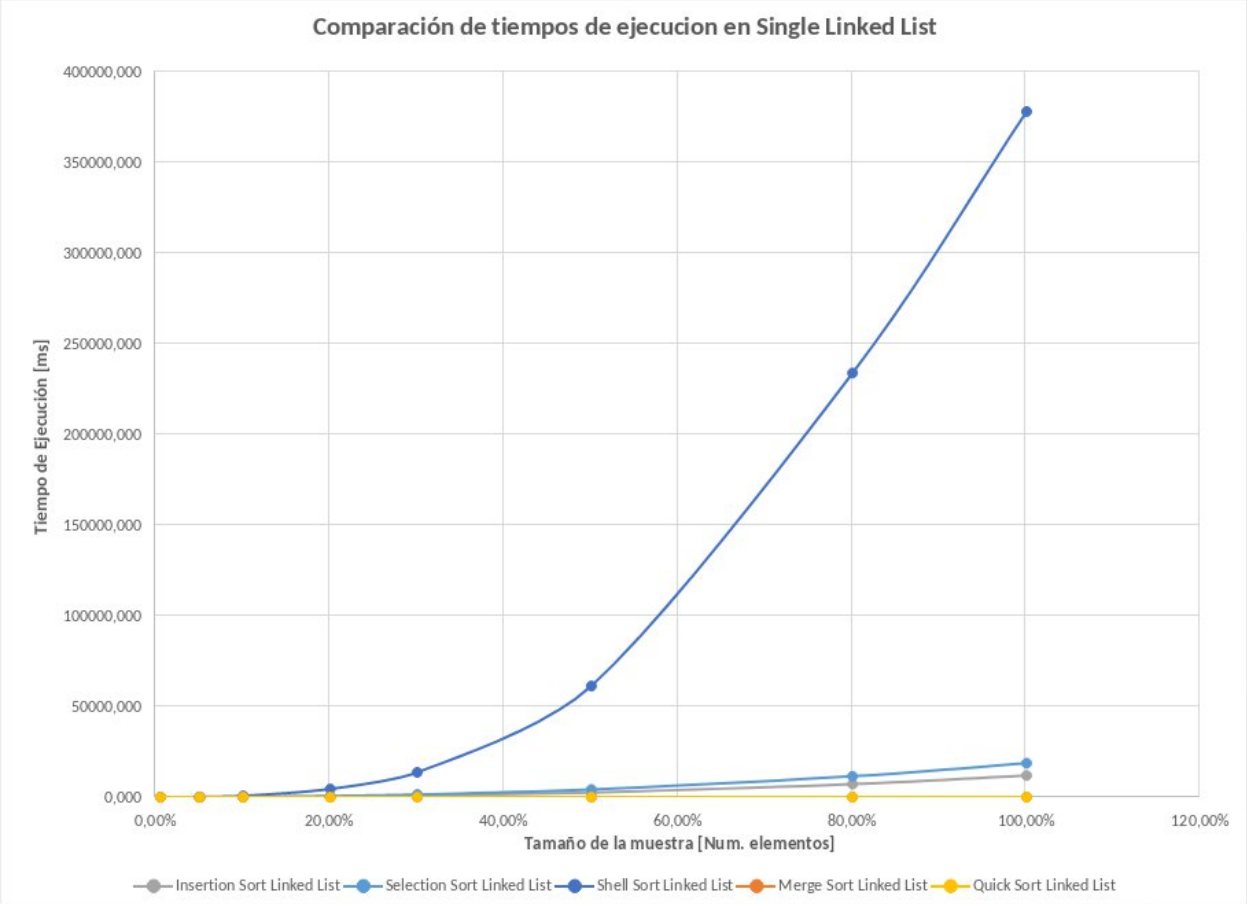
2. **¿Qué diferencias observas en el rendimiento de un mismo algoritmo al utilizar ArrayList frente a SingleLinkedList?**

RTA: El orden de crecimiento es igual para todos los ordenamientos, a excepción de Shell, en la cual la implementación en ArrayList es de menor crecimiento. Por otro lado, para los algoritmos de ordenamiento recursivos se observan tiempos muy cercanos, por lo que se infiere que su rendimiento es idéntico para ambos tipos de lista. En contraste, en los algoritmos iterativos (sin incluir Shell), se observa que Insertion tiene mejor rendimiento en LinkedList y Selection tiene mejor rendimiento en ArrayList. Lo anterior se debe a que Insertion utiliza O(n) intercambios hacia atrás en ArrayList, lo que es menos eficiente que insertar en la posición correcta (O(n)) en LinkedList. En Selection se utiliza una operación O(n) (delete_element en el ciclo while continue_sorting) en LinkedList en comparación con las operaciones O(1) (swap, get_element en el for de potential_index) en ArrayList.

3. **¿Qué algoritmo iterativo mostró el mejor comportamiento general en términos de tiempo y estabilidad en ambas estructuras de datos?**

RTA: Como es esperado, dada su orden de crecimiento, el algoritmo iterativo con mejor comportamiento fue Shell en ambas implementaciones de lista. Para apoyar lo anterior, observe la tabla de datos de rendimiento; más aún, observe la gráfica de rendimiento para ArrayList y LinkedList, que se muestra a continuación:





4. **¿Qué algoritmo recursivo presentó el mejor desempeño general según los tiempos registrados y la forma de la línea de tendencia?**

   RTA: El algoritmo recursivo con mejor rendimiento es Quick, ya que, a pesar de que tiene el mismo orden de crecimiento temporal (O(nlogn)) y espacial (O(n); la implementación de Quick no es in-place) que Merge, presenta tiempos menores. Lo anterior se explica con la mayor eficiencia en tiempo de la operación partition y combine con respecto a las operaciones sublist y merge, ya que las primeras son más simples y requieren menos pasos totales. Para justificar lo anterior, observe la tabla de rendimiento; más aún, observe que en las gráficas de la pregunta 3, los órdenes de crecimiento de Quick y Merge son idénticos; sin embargo, en las gráficas específicas de Merge y Quick, se nota que los tiempos de Merge son superiores en todos los casos.

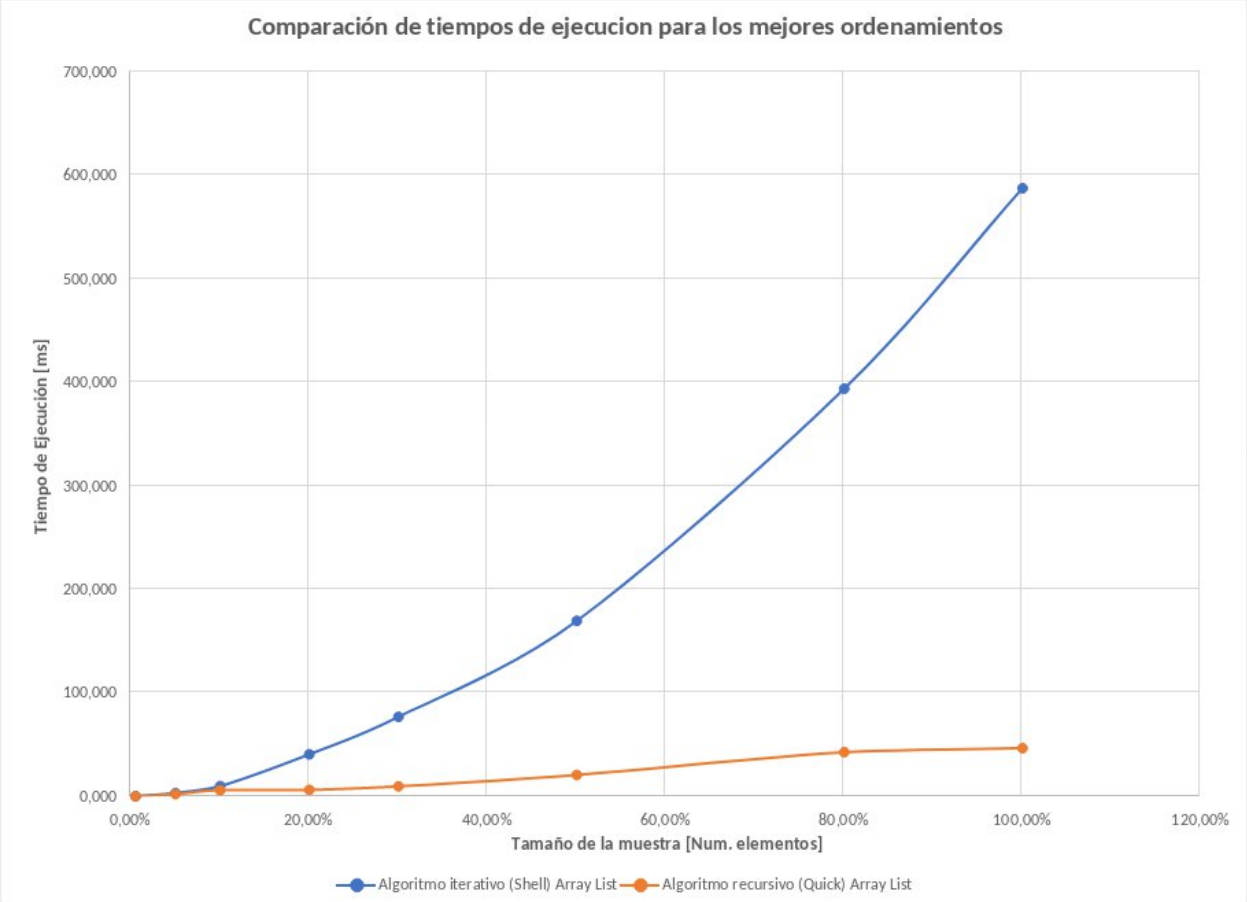
5. **Con base en su análisis visual y los datos recolectados, ¿cuáles algoritmos seleccionan como los “mejores” en cada categoría (iterativo y recursivo)?**

   RTA: En iterativo, Shell en ArrayList es el mejor algoritmo; esto se puede ver visualmente o con la tabla de rendimiento. El resultado es de esperar, ya que para ArrayList el orden de crecimiento es el menor respecto a todos los algoritmos iterativos en ArrayList y LinkedList. Por otro lado, el mejor algoritmo de ordenamiento es Quick en ArrayList, 5/8 veces tiempo medido como menor (respecto a Quick en LinkedList); esto se debe a que, a pesar de que el mismo algoritmo en LinkedList tenga un rendimiento similar, el algoritmo en ArrayList elige un pivote aleatorio, lo que ocasiona que en la mayoría de las veces los pivotes elegidos sean más óptimos. Más aún, se observa que, a medida que la muestra crece, se repiten más los casos en que el algoritmo en ArrayList es superior; esto es consecuencia de que, a medida que la muestra crece, la elección de pivotes de ArrayList sea probablemente más óptima que la elección de LinkedList. Se elige también ArrayList porque Quick garantiza que, en promedio (independientemente del input), el tiempo es O(nlogn) para un mismo input, algo que no garantiza LinkedList, ya que este mismo garantiza O(nlogn) en promedio para un conjunto de muestras aleatorias del mismo tamaño.

Preguntas de análisis parte 2

1. **¿Cuál de los dos algoritmos seleccionados como “mejores” muestra un comportamiento más eficiente a medida que crece el tamaño de la muestra?**

   RTA: El que presenta un rendimiento más eficiente a medida que crece la muestra es Quick, ya que si se fija un dato, el siguiente dato crece poco respecto al anterior, contrario a Shell, que crece aceptablemente, pero mucho respecto al dato anterior. La justificación se halla en la tabla de rendimiento y en la siguiente gráfica:



2. **¿Qué estructura de datos resultó más favorable para cada algoritmo seleccionado como mejor?**

   RTA: Para Quick, ambas implementaciones tienen, para efectos prácticos, la misma eficiencia en todos los rangos. Sin embargo, para Shell, el uso de LinkedList es sumamente ineficiente; más aún, como ya se comentó, aumenta la complejidad temporal. Esto se debe a que no es posible adaptar este algoritmo de manera eficiente para LinkedList, ya que este mismo realiza muchos intercambios de información entre posiciones muy dispersas entre sí, y esta operación solo es O(1) en ArrayList dado su acceso directo. Se concluye que Quick para ArrayList es el algoritmo ganador.

3. **¿Qué ventajas prácticas podría tener implementar el algoritmo ganador en aplicaciones reales donde se manejan grandes volúmenes de datos?**

   RTA: El tiempo, fundamentalmente, y la garantía de que, a medida que los datos incrementen, el tiempo aumentará de manera pausible. Lo anterior facilitaría el ordenamiento en corto tiempo de grandes volúmenes de datos en aplicaciones reales donde el tiempo de ejecución es fundamental. Como nota extra, se comenta que Quick puede ser adaptado para que sea in-place, ahorrando memoria en aplicaciones críticas donde esta debe ser optimizada; además, Merge, aunque no es el algoritmo ganador, puede ser fácilmente paralelizable, lo que posibilita ventajas en otros contextos.

4. **Si se presentara una lista parcialmente ordenada, ¿cambiaría su elección de algoritmo? Explique por qué, con base en el comportamiento observado.**

   RTA: Dado que no se usaron muestras parcialmente ordenadas, sino muestras aleatorias, no se puede inferir la respuesta por comportamiento experimental; sin embargo, cambiaría la elección a Insertion, ya que este mismo es adaptativo; más aún, su orden de crecimiento es cuasilineal para listas parcialmente ordenadas. La anterior elección se basa en la observación de las teorías existentes de algoritmos de ordenamiento. Se puede modificar la práctica para verificar que esta afirmación es, en efecto, correcta.