OBSERVACIONES DE LA PRÁCTICA

# **Ambientes de pruebas**

|  | Computador 1 |
| --- | --- |
| Procesador | Intel Core i3 |
| Memoria RAM (GB) | 8.00 GB |
| Sistema Operativo | Windows 10 |

*Tabla 1. Especificaciones del computador para ejecutar las pruebas de rendimiento.*

Para realizar las siguientes pruebas, es importante que las ejecuten utilizando el archivo de datos **large**, ya que es con este conjunto donde realmente se puede observar el comportamiento y la eficiencia de los algoritmos en contextos más exigentes.

# **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% | **0.313** | **1.052** | **1.097** |
| 5.00% | **1.204** | **77.939** | **9.217** |
| 10.00% | **2.335** | **252.900** | **32.065** |
| 20.00% | **4.560** | **858.411** | **261.411** |
| 30.00% | **6.499** | **2.891.622** | **328.423** |
| 50.00% | **16.201** | **6.437.774** | **668.296** |
| 80.00% | **18.085** | **19.879.302** | **682.846** |
| 100.00% | **27.592** | **38.602.278** | **1.437.333** |

*Tabla 2.* ***Resultados en computador 1 para ordenamientos 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% | **0.637** | **1.293** | **2.640** |
| 5.00% | **35.647** | **92.988** | **246.220** |
| 10.00% | **293.385** | **385.497** | **1.378.938** |
| 20.00% | **788.229** | **1.080.941** | **7.023.798** |
| 30.00% | **1.682.217** | **3.052.617** | **21.928.093** |
| 50.00% | **3.765.102** | **8.347.488** | **80.597.496** |
| 80.00% | **9.688.908** | **26.113.773** | **227.654.128** |
| 100.00% | **14.079.654** | **56.749.741** | **446.896.767** |

*Tabla 3.* ***Resultados en computador 1 para ordenamientos 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% | **0.738** | **0.935** |
| 5.00% | **12.833** | **8.867** |
| 10.00% | **26.168** | **63.183** |
| 20.00% | **71.742** | **45.612** |
| 30.00% | **172.533** | **62.474** |
| 50.00% | **242.094** | **138.325** |
| 80.00% | **290.436** | **204.760** |
| 100.00% | **435.307** | **215.593** |

*Tabla 4.* ***Resultados en computador 1 para ordenamientos 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% | **1.125** | **1.256** |
| 5.00% | **38.616** | **11.041** |
| 10.00% | **100.889** | **27.953** |
| 20.00% | **330.770** | **54.390** |
| 30.00% | **865.622** | **60.932** |
| 50.00% | **2.130.393** | **98.435** |
| 80.00% | **6.105.347** | **277.946** |
| 100.00% | **11.068.895** | **373.779** |

*Tabla 5.* ***Resultados en computador 1 para ordenamientos recursivos con Single Linked List.***

Una vez alla llenado l

# **Comparación de tiempo mejores algoritmos de ordenamiento**

**Complete esta sección únicamente cuando se le indique en las instrucciones.** En este punto, deberá comparar el tiempo de ejecución entre el mejor algoritmo de ordenamiento recursivo y el mejor algoritmo de ordenamiento iterativo, los cuales debió haber identificado en el punto anterior.

Por favor, registre las mediciones en el espacio correspondiente de la tabla, especificando claramente:

* El nombre de cada algoritmo utilizado
* El tipo de estructura de datos empleada**(*ArrayList* o *SingleLinkedList*)**

Asegúrese de completar todos los campos solicitados en la tabla e incluir los nombres según corresponda, no es necesario que vuelva a correr las pruebas esta información la puede sacar de las tablas anteriores.

## **Computador 1**

| Porcentaje de la muestra | Algoritmo recursive (QUICK-SORT ARRAY-LIST) | Algoritmo iterativo (INSERTION-SORT – ARRAY\_LIST) |
| --- | --- | --- |
| 0.50% | **0.935** | **0.313** |
| 5.00% | **8.867** | **1.204** |
| 10.00% | **63.183** | **2.335** |
| 20.00% | **45.612** | **4.560** |
| 30.00% | **62.474** | **6.499** |
| 50.00% | **138.325** | **16.201** |
| 80.00% | **204.760** | **18.085** |
| 100.00% | **215.593** | **27.592** |

*Tabla 6.* ***Resultados en computador 1 mejores algoritmos***

# **Preguntas de análisis parte 1**

1. **¿Cómo varía el comportamiento de cada algoritmo con respecto al tamaño de los datos?**  
     
   Para responder, agregue líneas de tendencia en las gráficas de las pestañas 04 a 08**.** Observe si el crecimiento del tiempo de ejecución es lineal, cuadrático o de otro tipo, y relacione ese patrón con la complejidad teórica esperada de cada algoritmo.

Insertion Sort y Selection Sort muestran un aumento cuadrático: cuando el tamaño de la muestra se incrementa, el tiempo sigue una trayectoria parabólica. Shell Sort crece de forma cercana a O(n log n).Por otro lado, Merge Sort y Quick Sort presentan un crecimiento casi lineal-logarítmico, consistente con su complejidad O(n log n).

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

Analice esta comparación en las pestañas 04 a 08, donde se muestran ambos casos. Justifique su respuesta con base en las líneas de tendencia.

**RTA:**

Los algoritmos fueron más rápidos scon ArrayList que sobre SingleLinkedList, y la brecha se hace mayor a medida que aumenta el tamaño de los datos. Esto se explica porque, en una lista enlazada simple, cada acceso o intercambio requiere recorrer nodos, mientras que en un ArrayList el acceso a cualquier posición es directo.

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

El mejor algoritmo iterativo fue Insertion Sort, notablemente con ArrayList. Aunque teóricamente Shell Sort puede acercarse a O(n log n), en esta implementación y con estos datos su tiempo superó al de Insertion Sort.

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

Quick Sort tuvo el mejor comportamiento en términos de tiempo y pendiente. En casi todos los tamaños de muestra superó a Merge Sort tanto en ArrayList como en SingleLinkedList, y en la estructura de arreglo sus tiempos se mantuvieron claramente por debajo de los de Merge Sort.

1. **Con base en su análisis visual y los datos recolectados, ¿cuáles algoritmos**

En conclusión, el mejor algoritmo iterativo es Insertion Sort y el mejor recursivo es Quick Sort, ambos ejecutados sobre ArrayList. Quick Sort con ArrayList es el método más rápido para conjuntos de datos grandes, mientras que Insertion Sort en ArrayList resulta especialmente conveniente para conjuntos pequeños.

# **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?**

La comparación muestra que Quick Sort con ArrayList aumenta su tiempo de ejecución de forma mucho más lenta que Insertion Sort conforme se incrementa el tamaño de la muestra. Esta diferencia coincide con la teoría: Quick Sort tiene complejidad O(n log n) y Insertion Sort es O(n²). Por eso, a medida queel tamaño del conjunto de datos crece, , Quick Sort resulta claramente mejor.

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

Para Quick Sort ypara Insertion Sort, la estructura ArrayList se ejecut´de forma más rápida que la SingleLinkedList en todos los tamaños de prueba. La ventaja de ArrayList se mantiene estable a lo largo de toda la escala, porque el acceso en la memoria es directo y la facilidad de intercambio de elementos reduce el costo total del algoritmo

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

Implementar Quick Sort en un ArrayList ofrece tiempos mejores cuando se manejan grandes volúmenes de datos, lo que significa un menor consumo de CPU y mayor capacidad para procesar información.

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

Dado un conjunto de datos ya parcialmente ordenado,tenemos dos opciones: Insertion Sort y Quick Sort. Insertion Sort suele funcionar bien bajo este tipo de orden y puede acercarse a un tiempo casi lineal. Con este conjunto de datos, si su tamaño es pequeño, Insertion Sort podría ser [mejor. No](http://mejor.no) obstante,con un tamaño de datos mayor, Quick Sort sería la opción más segura y estable, ya que su desempeño no depende tanto del grado de orden inicial.