**OBSERVACIONES – LABORATORIO 5**

Ana Sofía Castellanos Mosquera Cod 202114167

Martín Santiago Galván Castro 201911013

|  |  |  |
| --- | --- | --- |
|  | Máquina 1 (Ana Sofía) | Máquina 2 |
| Procesadores | Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz | AMD Ryzen 3 3200G with Radeon Vega Graphics 3.60 GHz |
| Memoria RAM (GB) | 16.0 GB | 16.0 GB |
| Sistema Operativo | Windows 10 Home Single Language | Windows 10 Pro |

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

# **Máquina 1**

## **Resultados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tamaño de la muestra (ARRAYLIST)** | **Insertion Sort [ms]** | **Selection Sort [ms]** | **Shell Sort [ms]** | **Quick Sort [ms]** | **Merge Sort [ms]** |
| 1000 | 462,49 | 403,12 | 21,67 | 27,3425 | 15,62 |
| 2000 | 1806,25 | 1587,5 | 46,88 | 62,5 | 39,065 |
| 4000 | 7253,12 | 6346,87 | 112,504 | 140,625 | 74,215 |
| 8000 | 29050 | 27334,38 | 265,62 | 277,34 | 160,1575 |
| 16000 | 125078,1 | 108010,4 | 671,88 | 550,7825 | 359,38 |
| 32000 | 541536,5 | 456750 | 1446,87 | 1160,1575 | 753,905 |
| 64000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 3428,12 | 2761,715 | 1640,625 |
| 128000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 8244,37 | 7636,72 | 3582,03 |
| 256000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 19250 | El programa se sale antes de completar ordenamiento | 7652,345 |

Tabla 2. Comparación de tiempos de ejecución para los ordenamientos iterativos en la representación arreglo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tamaño de la muestra (LINKED\_LIST)** | **Insertion Sort [ms]** | **Selection Sort [ms]** | **Shell Sort [ms]** | **Quick Sort [ms]** | **Merge Sort [ms]** |
| 1000 | 29369,79 | 25651,04 | 1411,46 | 1085,935 | 140,62 |
| 2000 | 233671,88 | 209942,73 | 6135,41 | 4941,4075 | 554,69 |
| 4000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 22093,75 | 22757,795 | 2199,215 |
| 8000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 139984,4 | 90644,5325 | 8765,625 |
| 16000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 414625 | 34980,4675 |
| 32000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 143812,6667 |
| 64000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | 589609,38 |
| 128000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) |
| 256000 | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) | Exceso de tiempo (>10 min) |

Tabla 3. Comparación de tiempos de ejecución para los ordenamientos iterativos en la representación lista enlazada.

|  |  |  |
| --- | --- | --- |
| Algoritmo | Arreglo (ARRAYLIST) | Lista enlazada (LINKED\_LIST) |
| Merge sort | Más eficiente | Menos eficiente |
| Quick sort | Más eficiente | Menos eficiente |

Tabla 4. Comparación de eficiencia de acuerdo con los algoritmos de ordenamientos y estructuras de datos utilizadas.

## **Graficas**

* Cinco gráficas generadas por los resultados de las pruebas de rendimiento en la **Máquina 1.**

Comparación de rendimiento ARRAYLIST.

Comparación de rendimiento LINKED\_LIST.

Comparación de rendimiento para Insertion Sort.

Comparación de rendimiento para Selection Sort.

Comparación de rendimiento para Shell Sort.

Comparación de rendimiento para MergeSort.

Comparación de rendimiento para QuickSort.

+-

# **Máquina 2**

## **Resultados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tamaño de la muestra (ARRAYLIST)** | **Insertion Sort [ms]** | **Selection Sort [ms]** | **Shell Sort [ms]** | **Quick Sort [ms]** | **Merge Sort [ms]** |
| 1000 | 645.83 | 609.38 | 31.25 | 26.04 | 31.25 |
| 2000 | 2567.71 | 2375 | 78.12 | 52.087 | 57.29 |
| 4000 | 10427.08 | 9562.5 | 171.88 | 119.793 | 114.59 |
| 8000 | 42666.67 | 40890.62 | 406.25 | 223.960 | 255.21 |
| 16000 | 184203.12 | 159109.38 | 968.75 | 531.250 | 531.25 |
| 32000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 2156.25 | 1135.417 | 1119.79 |
| 64000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 5031.25 | 2677.080 | 2442.71 |
| 128000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 12171.88 | 7437.500 | 5078.13 |
| 256000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 28687.5 | Error | 11020.83 |

Tabla 2. Comparación de tiempos de ejecución para los ordenamientos iterativos en la representación arreglo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tamaño de la muestra (LINKED\_LIST)** | **Insertion Sort [ms]** | **Selection Sort [ms]** | **Shell Sort [ms]** | **Quick Sort [ms]** | **Merge Sort [ms]** |
| 1000 | 30046.88 | 43984.38 | 1734.38 | 1546.87 | 192.71 |
| 2000 | 241250 | 344046.88 | 7406.25 | 7156.25 | 729.16 |
| 4000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 36218.75 | 35000.00 | 2994.79 |
| 8000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 165984.38 | 137130.21 | 12182.29 |
| 16000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 635937.50 | 49880.21 |
| 32000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | 206947.92 |
| 64000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) |
| 128000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) |
| 256000 | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) | Exceso de tiempo (> 10 min) |

Tabla 3. Comparación de tiempos de ejecución para los ordenamientos iterativos en la representación lista enlazada.

|  |  |  |
| --- | --- | --- |
| Algoritmo | Arreglo (ARRAYLIST) | Lista enlazada (LINKED\_LIST) |
| Merge sort | Mas eficiente | Menos eficiente |
| Quick sort | Mas eficiente | Menos eficiente |

Tabla 4. Comparación de eficiencia de acuerdo con los algoritmos de ordenamientos y estructuras de datos utilizadas.

## **Graficas**

* Cinco gráficas generadas por los resultados de las pruebas de rendimiento en la **Maquina 2.**

Comparación de rendimiento ARRAYLIST.

Comparación de rendimiento LINKED\_LIST.

Comparación de rendimiento para Insertion Sort.

Comparación de rendimiento para Selection Sort.

Comparación de rendimiento para Shell Sort.

Comparación de rendimiento para MergeSort.

Comparación de rendimiento para QuickSort.

# **Preguntas de análisis**

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

Para los resultados de la máquina 1 es posible analizar que el algoritmo MergeSort se comporta de mejor manera que el QuickSort lo cual está de acuerdo a la teoría en tanto que en el peor caso el algoritmo Merge se comporta de forma linearítmica con un orden de crecimiento temporal de la forma O(N log(N)) a comparación del algoritmo QuickSort que se comporta de forma O(N2).

Por otro lado cabe resaltar que hubo un incoveniente no esperado al probar el algoritmo Quicksort con un tamaño de datos de 256000 ya que el programa paraba de ejecutarse de forma inesperada a pesar de aumentar el límite de recursión.

Analizando los resultados de la maquina dos, este se ve limitado por la indisponibilidad de datos. Este problema se acentua para el analisis de resultados al usar la estructura de datos de lista encadenada. Adicionalmente, para el analisis de quicksort, no se obtuvo resultado de prueba de rendimiento para el tamaño de la lista de 256000 por un error no previsto al realizar las pruebas. Al eligir dicha opcion para una estructura de datos de Array list, el programa deja de ejecutarse. Se intento arreglar el problema aumentando el limite de recursión, pero siguio sucediendo lo mismo.

Analizando los datos disponibles, se puede determinar que los algoritmos de organizamiento de Selection e Insertion, se obseva que sus crecimientos temporales se ven limitados por funciones cuadraticas. Dado a esto, se determina que para este laboratorio, se comportan de acuerdo a sus peores casos, delimitadas por la notacion big O O(n^2). Por el lado del algoritmo shell sort, se evidencia que es el mejor algoritmos dentro de los algoritmos iterativos, y que este se comporta de acuerdo o similar a una función linearitmica, delimitada por O(nlog(n)) o similar a O(n^1.5). Esto se asume cierto, pues al compararla a la linea de tendencia lineal, se puede ver que la linea de los datos tiende a superar la de tendencia en cierto punto, similar a la relación entre la lineal y la linearitmica.

Por parte de los algoritmos de organizamiento recursivos, se ve que estas tienen un rendimiento considerablemente mejor a que los iterativos. Referente a como estas se comportan acorde a lo enunciado teoreticamente, se puede determinar que si, puesto a que ambas tienden a tener formas de la función linearitmica. La justificación de lo anterior es la misma que para el algoritmo shell sort.

1. ¿Existe alguna diferencia entre los resultados obtenidos al ejecutar las pruebas en diferentes máquinas?

Si. Se puede estimar que la máquina uno fue capaz de organizar los datos en más o menos la mitad o en tiempos menores a comparación de la máquina 2. Adicionalmente, para el caso de array\_list. La maquina 1 fue capaz de procesar más datos que la máquina 2. En todo lo demás, no se encuentran diferencias significativas en el rendimiento de las dos máquinas

1. De existir diferencias, ¿A qué creen ustedes que se deben dichas diferencias?

Las diferencias entre los rendimientos de las máquinas se pueden deber a varios factores. Por un lado, se observa que entre las dos máquinas de cómputo, difiere el procesador de las máquinas. Una posible razón puede ser la diferencia de procesadores. Otra posible fuente de diferencia se puede deber a programas o servicios que corran en el fondo que no se hayan cerrado o tenido en cuenta. Por último, otra posible fuente de diferencia se puede deber a suspender el pc antes de las pruebas de cómputo. Si se pone en modo de suspensión durante mucho tiempo, la memoria podría no restablecerse correctamente. Como resultado, esto puede afectar el rendimiento del código.

1. ¿Cuál Estructura de Datos es mejor utilizar si solo se tiene en cuenta los tiempos de ejecución de los algoritmos?

Teniendo en cuenta los tiempos de ejecución de los algoritmos se hace mejor implementar la estructura de datos ArrayList tanto para el MergeSort como para el QuickSort, esto se debe a que estos algoritmos hacen uso dentro de su código en diversas ocasiones de la función getElement(..), la cual tiene un crecimiento temporal de O(N) en la lista encadenada a comparación del arreglo donde es de O(1).

1. Para el caso analizado de ordenamiento de los videos, teniendo en cuenta los resultados de tiempo reportados por todos los algoritmos de ordenamiento estudiados (iterativos y recursivos), proponga un ranking de los algoritmos de ordenamiento (de mayor eficiencia a menor eficiencia en tiempo) para ordenar la mayor cantidad de videos.

Considerando los resultados de tiempo para los algoritmos de ordenamiento en Array List y Linked list de mayor eficiencia a menor eficiencia en el tiempo, el ranking para la máquina N1 sería:

1. Merge Sort- Array List
2. Quicksort- Array List
3. ShellSort- Array List
4. MergeSort-Linked List
5. Insertion Sort- Array List
6. Selection Sort – Array List
7. QuickSort-Linked List
8. ShellSort-Linked List
9. SelectionSort-Linked List
10. InsertionSort-Linked List

Para el caso de la máquina N2 sería:

1. Merge Sort- Array List
2. Quicksort- Array List
3. ShellSort- Array List
4. MergeSort-Linked List
5. Insertion Sort- Array List
6. Selection Sort – Array List
7. QuickSort-Linked List
8. ShellSort-Linked List
9. InsertionSort-Linked List
10. SelectionSort-Linked List

Si no se considera la estructura de datos implementada, el ranking de los algoritmos de ordenamiento sería:

1. MergeSort
2. Quick Sort
3. ShellSort
4. SelectionSort
5. Insertion Sort