**ANÁLISIS DEL RETO**

Jacobo Zarruk, 20222393, [j.zarruk@uniandes.edu.co](mailto:j.zarruk@uniandes.edu.co)

María José Amorocho, 202220179, [m.amorocho@uniandes.edu.co](mailto:m.amorocho@uniandes.edu.co)

# **Información máquinas**

|  |  |  |
| --- | --- | --- |
|  | Máquina 1 | Máquina 2 |
| Procesadores | AMD Ryzen 5 5625U with Radeon Graphics 2.30 GHz | Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.10 GHz |
| Memoria RAM (GB) | 16.0 GB | 8,00 GB |
| Sistema Operativo | Windows 11 Home Single Language | Windows 10 Pro – 64 bits |

# **Requerimiento 1**

Tipo de estructura de dato y forma de ordenamiento

## **Descripción**

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media Texto

Descripción generada automáticamente

Para la carga de datos dada una forma específica de lista, se añadió el parámetro data\_type, que da el tipo de estructura de dato que el usuario quiere crear en la función lt.newList. Esta función retorna la estructura de datos deseada por el usuario.

Para ordenar los datos se pide al usuario que escoja entre tres maneras de ordenamiento. Dependiendo de cual opción elija, se retorna una estructura de datos que ha sido organizada de diferente manera. Para el ordenamiento, se tiene como criterio el año del impuesto y el código de actividad económica (parámetros utilizados por la función sort\_criteria)

|  |  |
| --- | --- |
| **Función** | new\_data\_structs |
| **Entrada** | Tipo de estructura de datos (array list o linked list) |
| **Salidas** | Los datos organizados en la estructura seleccionada |
| **Implementado (Sí/No)** | Sí. Implementado por Jacobo Zarruk |

|  |  |
| --- | --- |
| **Función** | sort |
| **Entrada** | La estructura de datos y la manera en que se quiere ordenar (insertion, selection o shell sort) |
| **Salidas** | Los datos -impuestos- organizados de acuerdo con el año y actividad económica en forma ascendente. |
| **Implementado (Sí/No)** | Sí. Implementado por Jacobo Zarruk |

## **Análisis de complejidad**

Análisis de complejidad función new\_data\_structs

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Creación de un diccionario vacío | O(1) |
| Añadir una llave “data” con valor de una lista en forma de array list o linked list | O(1)\* |
| Retornar la estructura de datos | O(1) |
| ***TOTAL*** | ***O(1)*** |

\*El paso de añadir la llave “data” se realiza sólo una vez. Para la creación de la estructura de datos en forma de array list o linked list se recorre la matriz de datos y se agregan a la estructura, por lo que el proceso tiene una complejidad de O(n). Aún así, como cada impuesto es agregado al final de la lista, en cualquiera de los dos casos -array list o linked list- hay una complejidad de O(n).

Análisis de complejidad función sort

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Verifica si la forma de organizar la información es Shell. De ser así, retorna la estructura de datos organizada con el método Shell Sort; si no, continúa con el siguiente paso | O(1) |
| Verifica si la forma de organizar la información es Insertion Sort. De ser así, retorna la estructura de datos organizada con este método; de otra manera, continúa con el siguiente paso | O(1) |
| Retorna la estructura de datos organizada con el método Selection Sort | O(1) |
| ***TOTAL*** | ***O(1)\**** |

\*La función como tal tiene una complejidad O(1), pues cada paso sólo se ejecuta una vez; sin embargo, la complejidad de las maneras de ordenar la estructura de datos dependerá del método seleccionado y el tipo de estructura en el que se encuentren los datos.

## **Pruebas Realizadas**

**Máquina 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Porcentaje de la muestra [pct] | Tamaño de la muestra (ARRAY\_LIST) | Insertion Sort [ms] | Selection Sort [ms] | Shell Sort [ms] |
| 5.00% | 245 | 28.30 | 59.24 | 9.37 |
| 20.00% | 980 | 528.82 | 590.70 | 31.62 |
| 30.00% | 1470 | 1120.11 | 1185.01 | 100.81 |
| 50.00% | 2460 | 3055.38 | 3435.62 | 117.56 |
| 100.00% | 4093 | 12932.41 | 15449.34 | 209.21 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Porcentaje de la muestra [pct] | Tamaño de la muestra (LINKED\_LIST) | Insertion Sort [ms] | Selection Sort [ms] | Shell Sort [ms] |
| 5.00% | 245 | 704.17 | 632.11 | 91.13 |
| 20.00% | 980 | 44164.15 | 40653.06 | 2145.26 |
| 30.00% | 1470 | 152310.76 | 137827.14 | 5859.98 |
| 50.00% | 2460 | Más de 10 min | Más de 10 min | 19861.63 |
| 100.00% | 4093 | Más de 10 min | Más de 10 min | 94245.93 |

**Máquina 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Porcentaje de la muestra [pct] | Tamaño de la muestra (ARRAY\_LIST) | Insertion Sort [ms] | Selection Sort [ms] | Shell Sort [ms] |
| 5.00% | 245 | 30.35 | 40.65 | 9.16 |
| 20.00% | 980 | 497.73 | 508.37 | 53.73 |
| 30.00% | 1470 | 932.06 | 1064.10 | 69.95 |
| 50.00% | 2460 | 2775.94 | 3155.90 | 135.57 |
| 100.00% | 4903 | 13261.10 | 18593.92 | 180.45 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Porcentaje de la muestra [pct] | Tamaño de la muestra (LINKED\_LIST) | Insertion Sort [ms] | Selection Sort [ms] | Shell Sort [ms] |
| 5.00% | 245 | 803.16 | 703.54 | 93.17 |
| 20.00% | 980 | 46204.32 | 42792.24 | 2263.58 |
| 30.00% | 1470 | 155786.48 | 140953.75 | 6023.41 |
| 50.00% | 2460 | Más de 10 min | Más de 10 min | 21487.23 |
| 100.00% | 4903 | Más de 10 min | Más de 10 min | 97864.37 |

### **Graficas**

Gráficas máquina 1, Array List

Máquina 1, linked list

Máquina 2, array list

Máquina 2, linked list

## **Análisis**

Al observar los resultados obtenidos y las gráficas hechas nos damos cuenta de que, aunque existen diferencias entre los tiempos tomados por cada máquina (tal vez debido a la diferencia entre la memoria y el procesador de cada una), estas diferencias no son significativas, y el algoritmo se comporta de manera bastante similar en ambos casos. Así mismo, se puede notar que la complejidad de ordenamiento de datos coincide en gran medida con el valor teórico asignado; mientras que los ordenamientos usando selection e insertion sort tienden a tener una complejidad de O(n^2) -de forma parabólica, como se ve en las gráficas-, la función Shell sigue de manera más notoria la forma de una función constante, o por lo menos, su gráfica es menos pronunciada que la de los anteriores algoritmos de ordenamientos.

De la misma forma, se determina que el tipo de estructura de dato más efectivo, en términos de tiempo, para almacenar datos es el array list, pues la linked list demora considerable tiempo en ordenar los datos de las formas insertion y selection sort.