**Análisis del reto 1:**

**Estudiante 1:** José Daniel Montero Gutiérrez, código 202012732, correo Uniandes [j.monterog@uniandes.edu.co](mailto:j.monterog@uniandes.edu.co)  
**Estudiante 2:** David Ernesto Zamora Cortés, código 202113407, correo Uniandes [d.zamorac@uniandes.edu.co](mailto:d.zamorac@uniandes.edu.co)

|  |  |  |
| --- | --- | --- |
|  | **Máquina 1** | **Máquina 2** |
| **Procesador** | AMD Ryzen 5 2600X Six-Core Processor 3.60 GHz |  |
| **Memoria RAM (en GB)** | 8,00 GB |  |
| **Sistema operativo** | Windows 10 Pro 64-Bits |  |

**Justificación del uso de la estructura de datos:**

Dentro del programa, se decidió por solo hacer uso de listas basadas en arreglo (ARRAY\_LIST) ya que esta específica estructura de datos es más rentable para no hacer una modificación en el catálogo tan a menudo, además de que la búsqueda dentro del catálogo sería más sencilla al no estar repartido aleatoriamente dentro de la memoria, sino de manera más organizada, al contrario que cuando se usan listas simplemente encadenadas (SINGLE\_LINKED), ya que aquí ralentiza la carga de datos al catálogo, al mismo tiempo que la búsqueda de elementos dentro de la lista.

Por tanto, se debe tener en cuenta que **todos los cálculos** de complejidad temporal realizados en este documento fueron hechos **usando solo listas basadas en arreglo**.

**Análisis de complejidad de los requerimientos:**

Hechos por José Daniel:  
**Requerimiento 1:**

|  |  |
| --- | --- |
| def listChronologically(catalog, stYear, fnYear):      artistList = lt.newList('ARRAY\_LIST')      for artist in lt.iterator(catalog['artists']):          if (artist['BeginDate'] >= stYear) and (artist['BeginDate'] <= fnYear):              lt.addLast(artistList, artist)      return artistList | |
| artistList = lt.newList(‘ARRAY\_LIST’) | **O(1)** |
| for artist in lt.iterator(catalog[‘artists’]): | **O(n)** |
| if (artist[‘BeginDate’] >= stYear) and (artist[‘BeginDate’] <= fnYear): | **O(1)** |
| lt.addLast(artistList, artist) | **O(n)** |
| Este algoritmo toma: **1 + n + 1 + n = 2 + 2n**  Que se puede re-escribir como: **2n**  El orden de crecimiento temporal de este algoritmo es: **~O(n)** | |

**Requerimiento 3:**

|  |  |
| --- | --- |
| def classifyByTechnique(catalog, authorName):      authorID = None      for artist in lt.iterator(catalog['artists']):          if artist['DisplayName'] == authorName:              authorID = artist['ConstituentID']      piecesByTechniques = {'techniques': None}      piecesByTechniques['techniques'] = lt.newList('ARRAY\_LIST', cmpfunction=comparetechniques)      for piece in lt.iterator(catalog['pieces']):          if authorID in piece['ConstituentID']:              techniquePos = lt.isPresent(piecesByTechniques['techniques'], piece['Medium'])              if techniquePos > 0:                  technique = lt.getElement(piecesByTechniques['techniques'], techniquePos)              else:                  technique = newTechnique(piece['Medium'])                  lt.addLast(piecesByTechniques['techniques'], technique)              lt.addLast(technique['pieces'], piece)      totalPieces = 0      totalTechniques = lt.size(piecesByTechniques['techniques'])      mostUsedTechnique = {'name': None, 'piecesList': None, 'mostPieces': 0}      for technique in lt.iterator(piecesByTechniques['techniques']):          ammPieces = lt.size(technique['pieces'])          totalPieces += ammPieces          if ammPieces > mostUsedTechnique['mostPieces']:              mostUsedTechnique['name'] = technique['name']              mostUsedTechnique['piecesList'] = technique['pieces']              mostUsedTechnique['mostPieces'] = ammPieces | |
| authorID = None | **O(1)** |
| for artist in lt.iterator(catalog[‘artists’]): | **O(n)** |
| if artist[‘DisplayName’] == authorName: | **O(1)** |
| authorID = artist[‘ConstituentID’] | **O(1)** |
| piecesByTechniques = {‘techniques’: None} | **O(1)** |
| piecesByTechniques[‘techniques’] = lt.newList(…) | **O(1)** |
| for piece in lt.iterator(catalog[‘pieces’]): | **O(n)** |
| if authorID in piece[‘ConstituentID’]: | **O(1)** |
| techniquePos = lt.isPresent(…) | **O(n)** |
| if techniquePos > 0: | **O(1)** |
| technique = lt.getElement(…) | **O(1)** |
| technique = newTechnique(…) | **O(1)\*** |
| lt.addLast(…) | **O(1)** |
| totalPieces = 0 | **O(1)** |
| totalTechniques = lt.size(…) | **O(1)** |
| mostUsedTechnique = {…} | **O(1)** |
| for technique in lt.iterator(…) | **O(n)** |
| ammPieces = lt.size(…) | **O(1)** |
| totalPieces += ammPieces | **O(1)** |
| if ammPieces > mostUsedTechnique[‘mostPieces’]: | **O(1)** |
| mostUsedTechnique[‘name’] = technique[‘name’] | **O(1)** |
| mostUsedTechnique[‘piecesList’] = technique[‘pieces’] | **O(1)** |
| mostUsedTechnique[‘mostPieces’] = ammPieces | **O(1)** |
| **\*** La función usada (newTechnique(…)) se toma como O(1), debido a que solo realiza la tarea de crear una nueva lista.  Este algoritmo toma: **1 + n + 4 + n + 1 + n + 7 + n + 6 = 19 + 4n**  Que se puede re-escribir como: **4n**  El orden de crecimiento temporal de este algoritmo es: **~O(n)** | |

**Requerimiento 4 hecho por David Zamora**

**Pruebas de tiempo de ejecución:**

**Requerimiento 1:**

Los datos usados para ejecutar este requerimiento fueron el rango de años de 1912, a 1919

|  |  |
| --- | --- |
| **Porcentaje de la muestra [pct]** | **Tiempo de ejecución (en ms)** |
| **small** | 0.0\* |
| **5.00%** | 0.0\* |
| **10.00%** | 0.0\* |
| **20.00%** | 0.0\* |
| **30.00%** | 0.0\* |
| **50.00%** | 0.0\* |
| **80.00%** | 0.0\* |
| **large** | 15.625 |

**\* El tiempo en esta máquina solo arroja 0.0 ms al usar ese rango de fechas (sin que el archivo sea *large*), mientras que también devuelve el resultado correcto que se introdujo, por lo que se concluye que el tiempo de ejecución fue tan bajo que el programa lo detectó como nulo.**

**Requerimiento 3:**

El dato usado para ejecutar este requerimiento fueron distintos artistas, ya que no siempre estaba el mismo en los distintos archivos

|  |  |  |
| --- | --- | --- |
| **Porcentaje de la muestra [pct]** | **Tiempo de ejecución (en ms)** | **Artista usado** |
| **small** | 0.0\* | Rick Prol |
| **5.00%** | 0.0\* | Umberto Nason |
| **10.00%** | 15.625 | Teresa Margolles |
| **20.00%** | 15.625 | Conrad Botes |
| **30.00%** | 15.625 | Lia Cook |
| **50.00%** | 31.25 | Nancy Rubins |
| **80.00%** | 46.875 | Quanta |
| **large** | 78.125 | Grapus |

**\* Al usar los archivos *small* y *5pct*, el resultado del tiempo solo da 0.0 ms, por lo que se concluye que el tiempo de ejecución fue tan bajo que el programa lo detectó como nulo.**