Documento de Análisis Reto 2

GRUPO 3:

Requerimiento 3
- Ana Sofía Villa Benavides, 201923361, as.villa@uniandes.edu.co

Requerimiento 4
- Daniela Alejandra Camacho Molano, 202110974, d.camachom@uniandes.edu.co

NA = número artistas
NO= número obras
M= tamaño tabla

Análisis de complejidad

Requerimiento 0 (carga de datos)

`initCatalog(estructura)` esta función en view llama a otra del mismo nombre que a su vez llama a la función

- **`newCatalog(estructura):`**
- Esta función es **O (1)**, ya que se genera un catálogo a manera de diccionario con dos llaves, una para artistas y otra para obras. Luego para cada categoría se le crea y asigna una lista vacía con la función de TAD lista newlist(). En este reto decidimos realizar la carga inicial de los datos con ARRAY LIST como estructura ya que en varias ocasiones utilizamos funciones como get.element() sublist() y algoritmos de ordenamiento específicos que tienen una complejidad menor cuando se trata de esta estructura.

`loadData(catalog)` esta función en view llama a otra función del mismo nombre en controlador que a su vez llama a dos funciones:

- **`loadArtistas`**Para cada artista en el archivo csv ósea **O(NA)** llama a la función en modelo:
  - **`addArtistb`**Esta función almacena en un diccionario solo la información necesaria de cada artista. A la función solo entra un artista y se accede a su información por llave por lo tanto no hay que recorrer y es **O (1).**
  - Adicionalmente, esta función crea una categoría de obras para cada artista con new.list(ARRAY_LIST) (que es **O (1)**) ya que mantenemos nuestra decisión de utilizar esta estructura, sobre todo porque esta lista de obras para cada artista luego será utilizada con funciones que resultan de menor complejidad con arreglos.
- **`loadObras`** Para cada artista en el archivo csv ósea **O(NO)** llama a la función en modelo:
  - **`def addObra:`** Esta función almacena en un diccionario solo la información necesaria de cada obra. A la función solo entra una obra cuya información se accede por llaves por lo tanto no hay que recorrer y es **O (1).**
  - Adicionalmente, esta función crea una categoría de artistas para cada obra con new.list(ARRAY_LIST) (que es **O (1)**) ya que mantenemos nuestra decisión de utilizar esta estructura, sobre todo porque esta lista de artistas para cada obra luego será utilizada con funciones que resultan de menor complejidad con arreglos.
  - Para realizar la referencia entre artistas y obras en las listas vacías creadas anteriormente, hacemos planteamos una sección de código en la que primero con un for se recorren los artistas de una obra **o(#artitstasenobra)**, asumimos que este número es mucho menor que O(NA) ya que una obra, aunque puede tener varios artistas, según los datos revisados no pasa de 100 artistas por obra. También asumimos que este número entonces es una constante **e** para cada obra. Luego hacemos otro for en el cual se recorren todos los artistas del catálogo O(NA), para buscar una coincidencia con el código de interés y entonces añadir la obra a la lista de obras del artista y el artista a la

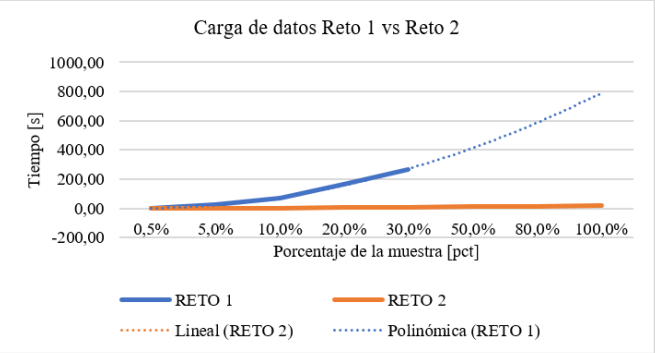
lista de artistas de la obra. Esto ultimo se realiza con lt.addlast() que es O(1). La mezcla de estos dos for no alcanza a ser NA^2 ya que **e** es un valor mucho menor, se trataría entonces de una complejidad O(NA*e) ≈ O(NA).

- En total la función **addObras** sería O(NO)*O(NA)= O(NO*NA), sabemos que NO>>NA, entonces O(NO*NA) es mayor que O(NA^2) y menor O(NO^2).

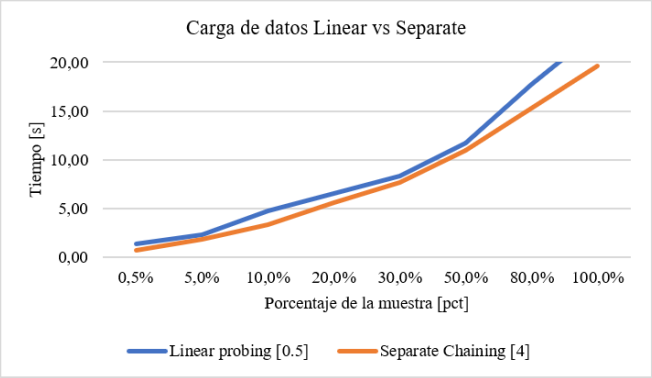
En total, el requisito sería O(NA)+O(NO*NA), como se toma el mayor en la suma sería O(NA*NO)

Pruebas de tiempos de ejecución:

Carga		
Porcentaje de la muestra [pct]	**Tiempo [s]**	
	RETO 1	**RETO 2**
0,5%	1,11719	0,70
5,0%	26,57031	1,89
10,0%	68,69531	3,38
20,0%	163,98438	5,59
30,0%	268,68750	7,67
50,0%		10,97
80,0%		16,26
100,0%		19,62



Carga		
Porcentaje de la muestra [pct]	**Tiempo [s]**	
	Linear probing [0.5]	**Separate Chaining [4]**
0,5%	1,34375	0,70
5,0%	2,35938	1,89
10,0%	4,76000	3,38
20,0%	6,53438	5,59
30,0%	8,36719	7,67
50,0%	11,73438	10,97
80,0%	17,71875	15,26
100,0%	23,21875	19,62



El Reto 1 se había planteado una complejidad de O(NA*NO) en cambio en este reto se plantea una complejidad de O(*). Esto se puede ver en la gráfica.....

Requerimiento 1

Análisis de complejidad:

`sortArtistInDateRange`:

Pruebas de tiempos de ejecución:

El Reto 1 se había planteado una complejidad de O(NA Log(NA)), en cambio en este reto se plantea una complejidad de O(*). Esto se puede ver en la gráfica.....

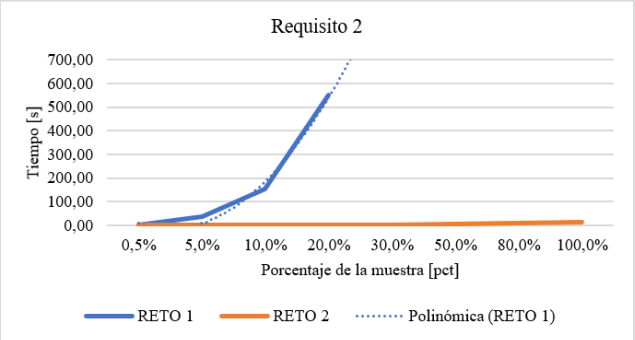
Requerimiento 2

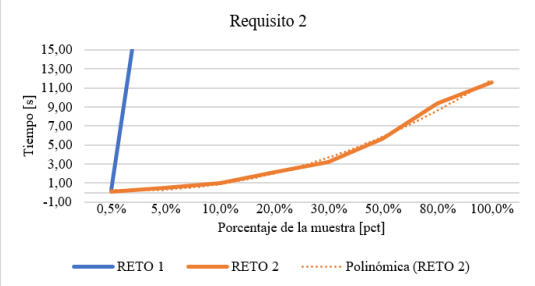
Análisis de complejidad:

`sortArtworksandRange(lista,inicial,final):` una lista tiene los primeros 3 elementos y la otra los últimos 3. Luego imprime las dos sublistas usando la función:

Pruebas de tiempos de ejecución:

Req 2		
Porcentaje de la muestra [pct]	**Tiempo [s]**	
	RETO 1	**RETO 2**
0,5%	0,29688	0,09375
5,0%	38,75000	0,53125
10,0%	155,25000	1,03125
20,0%	551,01563	2,12938
30,0%		3,23438
50,0%		5,64063
80,0%		9,34375
100,0%		11,61719





El Reto 1 se había planteado una complejidad de O (NO Log (NO), en cambio en este reto se plantea una complejidad de O(*). Esto se puede ver en la gráfica.....

Requerimiento 3 (Ana Sofía Villa Benavides)

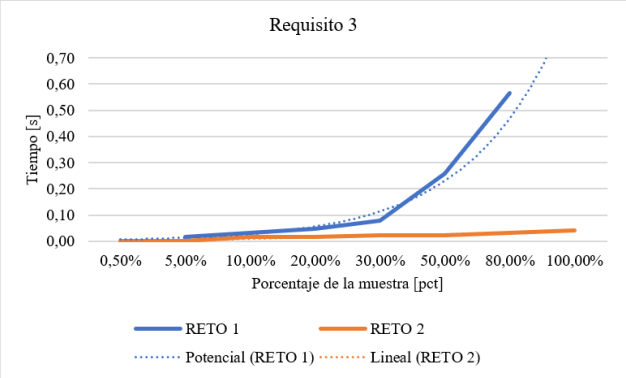
Análisis de complejidad:

`ObrasPorArtistaPorTecnica(catalog,nombre):`

`buscarTecnicaMasRep(Tecnicas):`

Pruebas de tiempos de ejecución:

Req 3		
Porcentaje de la muestra [pct]	**Tiempo [s]**	
	RETO 1	**RETO 2**
0,50%		0,00000
5,00%	0,01563	0,00001
10,00%	0,03125	0,01563
20,00%	0,04688	0,01563
30,00%	0,07813	0,02340
50,00%	0,25765	0,02343
80,00%	0,56543	0,03125
100,00%		0,04164



El Reto 1 se había planteado una complejidad de ≈ O (no*NA) que se asemeja a una complejidad potencial, en cambio en este reto se plantea una complejidad de O (1). Esto se puede ver en la gráfica claramente.

Requerimiento 4 (Daniela Alejandra Camacho)

Análisis de complejidad:

`RankingCountriesByArtworks`

Pruebas de tiempos de ejecución:

El Reto 1 se había planteado una complejidad de *****, en cambio en este reto se plantea una complejidad de O(*). Esto se puede ver en la gráfica.....

Requerimiento 5

Análisis de complejidad:

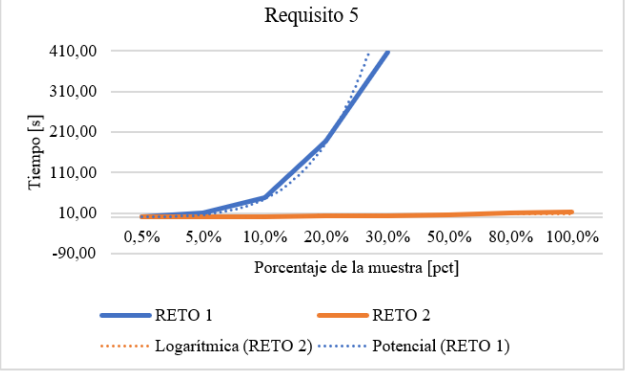
`OrdenarDepartamentoAsignarPrecioyPeso(catalog,departamento):` accede al valor de la pareja llave valor donde la llave es el departamento. Esto se logra ya que en la carga de datos se planteó un mapa con este índice. Acceder a este valor es O(1)

- **`def AsignarPrecio(object):`** asigna precio la obra que entra la función de acuerdo a los parámetros dados. Es O(1)

`SortArtworksByPrice y sortArtworksByDate:` Ordena las obras del departamento dado (que fueron seleccionadas con la función anterior) ambas usan el algoritmo de merge sort, por lo tanto tienen una complejidad de O(n log n) donde n corresponde al número de obras de un solo departamento.

Pruebas de tiempos de ejecución:

Req 5		
Porcentaje de la muestra [pct]	**Tiempo [s]**	
	RETO 1	**RETO 2**
0,5%	0,14063	0,04688
5,0%	10,75000	0,48374
10,0%	47,96875	0,94680
20,0%	187,71875	2,05300
30,0%	407,09375	3,15580
50,0%		5,34375
80,0%		8,94000
100,0%		11,50000



El Reto 1 se había planteado una complejidad de O(NO log NO), que en la gráfica se aproxima más a un n^2, en cambio en este reto se plantea una complejidad de O(n log n) donde n corresponde al número de obras

de un solo departamento. En este reto obtenemos un procedimiento con complejidad temporal mucho menor que el anterior ya que no hay que recorrer todas las obras para seleccionar las del departamento dado, esto se puede ver claramente en la tabla y gráfica.