RETO 1- ESTRUCTURA DE DATOS Y ALGORITMOS

Integrantes:

- **Nombre:** Juan Sebastián Sánchez Delgado
- **Correo**: js.sanchezd1@uniandes.edu.co
- **Código:** 2020135577

- **Nombre:** Nicolás Alexander rodríguez Pinilla
  Correo: na.rodriguezp1@uniandes.edu.co
  Código: 20202250

ANALISIS DE COMPLEJIDAD Y PRUEBAS DE EJECUCION:

REQUISITO 1:

La función listarCronologicamente crea una lista nueva y agrega a esta ultimo
todos los artistas que estén en el rango. Para ello se debió haber recorrido la lista completa una vez, por lo que la complejidad hasta este punto es O(N), siendo N el numero de elementos de la lista original.

Luego, se crea una segunda lista donde se agregarán los elementos de la primera lista creada ya ordenados. En el peor caso, todos los elementos de la lista estarán situados también en la primera lista creada, así que la complejidad se seguirá expresando en términos de N. El programa recorrerá la lista completa un numero definido por la siguiente expresión: añoFinal - añoInicial.

En conclusión, la complejidad temporal del programa es N + (añoFinal - añoInicial)N, que en notación O vendria siendo: O(N).En el peor caso, la cantidad de memoria extra que se utiliza seria de 2N.

PRUEBA

* Las pruebas mostradas a continuacion se hicieron con los archivos de extension “large”

Sistema operativo: Mac OS
Procesador: 1,6 GHz Intel Core i5 de dos núcleos
Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1:

Año inicial: 1920
Año final: 1985
Tiempo de ejecucion aproximado: 0,93s

Prueba 2:

Año inicial: 1850

Año final: 2015
Tiempo de ejecucion aproximado: 1,58s

Prueba 3:
Año inicial: 1700
Año final: 2021
Tiempo de ejecucion aproximado: 3,22s

La complejidad del apartado anterior es congruente con los tiempos registrados para las 3 pruebas. A sí mismo, es evidente que la solución que se implemento es no solo funcional, sino que también es altamente efectiva. Con todo lo anterior se concluye que vale la pena utilizar memoria adicional en este caso para aumentar la eficiencia del programa final.

REQUISITO 2:
La función listarAdquisiciones crea una lista nueva y agrega a ella todas las obras que se encuentren en el rango de fechas. Para ello, recorre la lista una vez, por lo que la complejidad es O(N).
Luego, se crea una segunda lista donde se agregarán los elementos de la primera lista creada ya ordenados. En el peor caso, todos los elementos de la lista estarán situados también en la primera lista creada, así que la complejidad se seguirá expresando en términos de N. El programa recorrerá la lista completa un numero definido por la siguiente expresión: fechaInicial- fechaFinal.

En conclusión, la complejidad temporal del programa es N + (fechaFinal - fechaInicial)N, que en notación O vendria siendo: O(N).En el peor caso, la cantidad de memoria extra que se utiliza seria de 2N.
PRUEBA

* Las pruebas mostradas a continuacion se hicieron con los archivos de extension “large”

Sistema operativo: Windows
Procesador: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1:

fechaInicial: 1920-10-09
fechaFinal: 1985-11-11
Tiempo de ejecucion aproximado: 0,41s

Prueba 2:

fechaInicial: 1850-09-10
fechaFinal: 2015-11-11
Tiempo de ejecucion aproximado: 0,91s

Prueba 3:

fechaInicial: 1700-09-10
fechaFinal: 2021-11-11
Tiempo de ejecucion aproximado: 2,12s

REQUISITO 4(Juan Sebastián Sánchez Delgado):

El requisito en cuestión fue implementado utilizando en total 3 funciones en el model.py. La primera y principal se denomina nacionalidadCreadores, en ella primero se recorre primero la lista de los artistas una sola vez, esto con el objetivo de crear un diccionario dentro que tenga como llave el id del artista y como valor otro diccionario con el nombre y la nacionalidad del mismo. Lo anterior se hace para cada uno de los artistas.

Dentro del primer for también se identifican las diferentes nacionalidades existentes y se agregan a un diccionario, esto con el fin de realizar el conteo de las obras por nacionalidad mas adelante. Como el primer ciclo solo recorrerá la lista una vez independientemente del caso, se puede afirmar que tiene una complejidad de O(N), siendo N el numero de elementos de la lista.

En la misma función, existe un segundo ciclo for que recorre toda la lista de obras una sola vez. Este se encarga de evaluar el ConstituentID de cada una de las obras y verificar en el diccionario que contiene la información de los artistas para saber la nacionalidad de la obra dada. Finalmente dependiendo de la nacionalidad, se sumará 1 a su respectivo semejante en el diccionario de los países. Esta parte también tendrá tiene una complejidad de O(N).

Se puede concluir que la complejidad temporal para toda la función será O(N). Suponiendo que cada artista tiene una nacionalidad distinta y que una pareja llave-valor ocupa 1 espacio en memoria, en el peor caso, se estaría utilizando aproximadamente N + 3N.

La función sortPaises recorre 10 veces dictNacionalidades para encontrar las nacionalidades con el mayor número de obras. Por ello su complejidad temporal aproximada seria 10N sin importar el caso o O(N). La memoria extra que usa siempre es 10, aunque como el elemento que agrega a la lista lo va borrando del diccionario, se puede decir que la funcion en realidad no ocupa memoria adicional.

Por último, la función obrasPais recorre siempre una vez el diccionario dictNacionalidades, luego recorre la lista de obras también solo una vez, perdón dentro de este ciclo recorre a su vez la lista que contiene el id del artista y se llama ConstituentID. Por lo general, esta última lista suele tener de 1 a como mucho 5 elementos. Por lo anterior, sería erróneo pensar que la complejidad es n^2 ya que el ciclo al interior del otro se ejecuta muy pocas veces, sin importar el caso. Teniendo en cuenta lo anterior, la complejidad temporal es de O(N) para esta función y para el requisito en general.

PRUEBA

* Las prubas mostradas a continuacion se hicieron con los archivos de extension “large”

Sistema operativo: Mac OS
Procesador: 1,6 GHz Intel Core i5 de dos núcleos

Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1:
Tiempo de ejecución aproximado: 1,33s

Prueba 2:
Tiempo de ejecución aproximado: 1,18s

Prueba 3:
Tiempo de ejecución aproximado: 1,21s

Este requisito en concreto, sin importar el caso que se presente, siempre dará un tiempo muy similar siempre y cuando se utilice el mismo archivo. Esto se debe a que siempre realizara la misma ejecución con los mismos datos por defecto. El uso combinado de diccionarios y la estructura de datos vistas durante el curso resulto ser una solución realmente eficiente.

Normalmente, si se implementara solo con el TAD lista, sería necesario recorrer la lista artista una vez por cada id del artista o implementar un algoritmo de ordenamiento para luego hacer búsqueda binaria. En cuanto a la memoria extra empleada, al no ser tan grande y disminuir la complejidad en gran medida, es una medida adecuada para el programa diseñado.

REQUISITO 5:

La función transportar_obras hace un recorrido de todos los elementos en la lista, la primera vez, para calcular el peso, el número de obras y el precio estimado del departamento. Luego, se recorre la lista nuevamente otras 5 veces más para crear 2 listas que contengan las 5 obras más antiguas y las 5 más costosas. También se utilizan temporalmente algunos arreglos para albergar la informacion en dimensiones.

Teniendo en cuenta lo anterior, la complejidad temporal es de O(N) y la memoria extra utilizada sera aN + 10 aproximadamente, en donde el producto aN representa el número total de dimensiones en términos de cm que hay en toda la lista.

PRUEBA

* Las pruebas mostradas a continuacion se hicieron con los archivos de extension “large”

Sistema operativo: Mac OS
Procesador: 1,6 GHz Intel Core i5 de dos núcleos
Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1:
Departamento: Drawings & Prints
Tiempo de ejecución aproximado: 2,53s

Prueba 2:

Departamento: Gelatin silver print
Tiempo de ejecución aproximado: 2,68s

Prueba 3:
Departamento: Lithograph, printed in color
Tiempo de ejecución aproximado: 2,57s

El programa en cuestión es eficiente en términos de tiempo, pero podría mejorar con respecto al manejo de la memoria. También, algunos de los resultados arrojados para el precio estimado no concordaron con lo pronosticado, por lo que por obvias razones este programa puede mejorar. El principal problema a la hora de implementarlo fue determinar en qué casos se utilizaba la formula y cuales otros el precio por defecto. Existían casos en donde no daban una medida en concreta sino una proporción, lo que hacía difícil realizar el cálculo en ocasiones.

Otra problemática que surgió fue que no se sabe a ciencia cierta si da lo mismo utilizar las dimensiones de la lista que utilizar la altura, el ancho y el largo aparte. El resto de la información solicitada, con sus más y sus menos, se pudo implementar tal como pedia el enunciado.