ANÁLISIS DEL RETO

Estudiante 1: Esteban Castelblanco 202214942 e.castelblanco@uniandes.edu.co

Estudiante 2: Cesar Avellaneda 200214746 c.avellanedac@uniandes.edu.co

Estudiante 3: Nicolas Casas 202212190 n.casasi@uniandes.edu.co

Requerimiento 1

```python
# Funciones utilizadas para comparar elementos dentro de una lista
def requerimiento1(desde, hasta, catalog):
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    sortcatalog(catalog, comparerequerimiento168)
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            if show["release_year"]>=desde and show["release_year"]<=hasta and show["type"]=="Movie":
                temp={}
                temp["type"]=show["type"]
                temp["release_year"]=show["release_year"]
                temp["title"]=show["title"]
                temp["duration"]=show["duration"]
                temp["stream_service"]=show["stream_service"]
                temp["director"]=show["director"]
                temp["cast"]=show["cast"]
                lt.addLast(respuesta, temp)
    return respuesta
```

Descripción

Para todos los requerimientos se tenia que hacer una tabla, con valores específicos, así que se hizo una función que recibiera un datastructure y con la librería de pretty table generara la tabla y también se hizo una función que tomara un datastructure y devolviera uno que tuviera solo los valores que se quieren en la tabla. Por eso todos los retornos de los requerimientos son un datastructure nuevo con la respuesta esperada para la tabla.

Para el primer requerimiento se explora el catalogo y cuando se encuentra una película, es decir que el atributo “type” del contenido es “Movie” y además su año de lanzamiento esta dentro del intervalo que se pasa por parámetro, agrega la película a la datastructure de la respuesta. Antes de esto se organiza el catalogo con las instrucciones de la guía, esto se hace con un merge sort y con una compare function que se usa posteriormente para el requerimiento 6 también.

En el view se usa una función de sacar los primeros y últimos 3 contenidos de un adt structure para la segunda tabla, a su vez se crean en el view las tablas para luego imprimirlas, esto se repite en casi todas las funciones.

Entrada	Año inicial, año final y el catálogo (diccionario que tiene como llave las plataformas y como valor un array list con los contenidos)
Salidas	Devuelve un array list que tiene como valores las películas que se estrenaron en ese tiempo.
Implementado (Sí/No)	Sí, Cesar Avellaneda, Esteban Castel, Nicolas Casas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Requerimiento 1	
Pasos	**Complejidad**
Paso 1: crea estructura	o(1)
Paso 2: Sort	o(nlogn)
Paso 3: recorrer catalogo por plataformas	o(4n) = n
Paso 4: recorrer cada show por plataforma	o(n)
Paso 5: condición	o(3)
Paso 6: crear diccionario	o(1)
Paso 7: agregar contenido al diccionario	o(1)
Paso 8: agregar contenido al diccionario	o(1)
Paso 9: agregar contenido al diccionario	o(1)
Paso 10: agregar contenido al diccionario	o(1)
Paso 11: agregar contenido al diccionario	o(1)
Pase 12: agregar contenido al diccionario	o(1)
Paso 13: agregar contenido al diccionario	o(1)
Paso 14: AddLast	o(1)
Paso 15: Retrun	o(1)
Total	o(n^3logn) = o(n^2)

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	849.16 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator y newlist) Recorrido por medio de 2 loop Condicionales y crear un diccionario y agregar valor a las llaves
Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

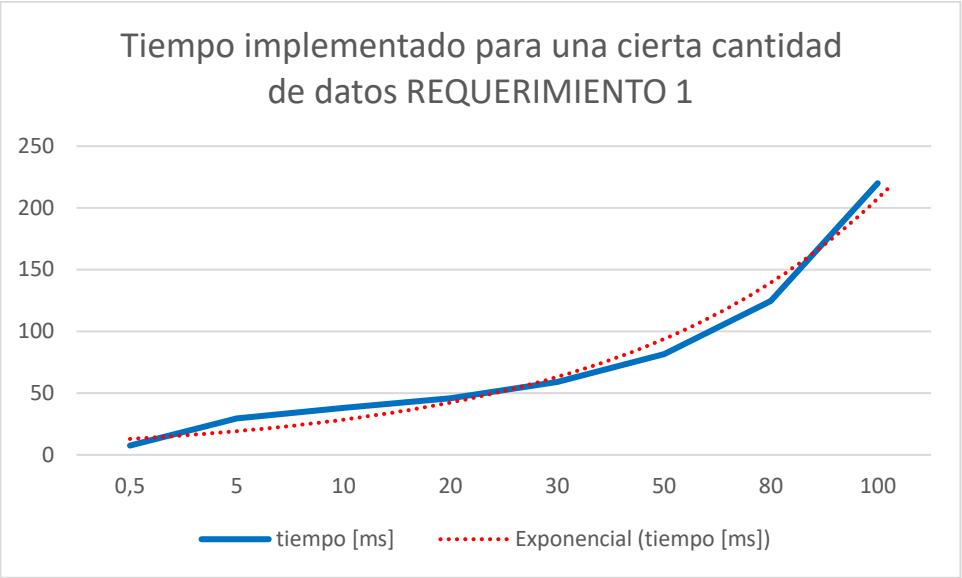
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	17,47
5	77,29
10	85,89
20	232,13
30	253,23
50	424,93
80	685,32
100	849,15

Graficas

Las gráficas con la representación de las pruebas realizadas.

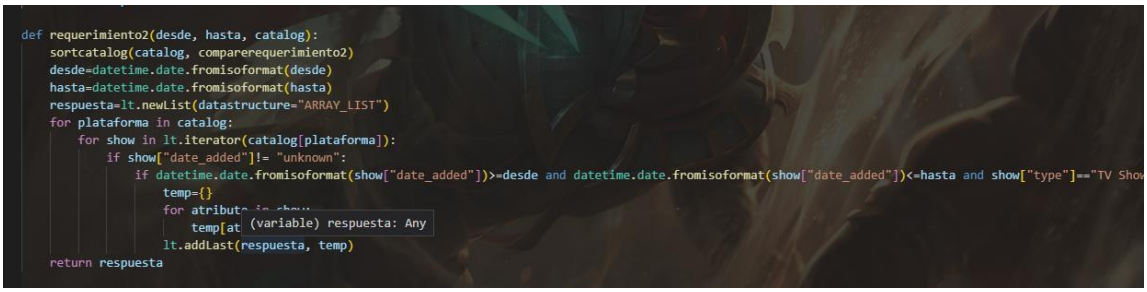


Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la grafica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.

Requerimiento 2



```
def requerimiento2(desde, hasta, catalog):
    sortcatalog(catalog, comparerequerimiento2)
    desde=datetime.date.fromisoformat(desde)
    hasta=datetime.date.fromisoformat(hasta)
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            if show["date_added"]!= "unknown":
                if datetime.date.fromisoformat(show["date_added"])>=desde and datetime.date.fromisoformat(show["date_added"])<=hasta and show["type"]=="TV Sho
                    temp={}
                    for atribut
                        temp[at (variable) respuesta: Any
                    lt.addLast(respuesta, temp)
    return respuesta
```

Descripción

El segundo requerimiento se parece mucho al primero, con la diferencia de que lo que se busca no son películas sino shows de televisión, además de esto el parámetro de búsqueda es la fecha en que se añadió, para lo que se tuvo que usar la librería datetime, en particular la función datetime.date.fromisoformat, la cual transforma los strings en forma (“YYYY-MM-DD”) en un valor que se pueda comparar, para que los shows que si cumplan con el condicional sean agregados a la datastructure que se da como retorno

Entrada	Fecha inicial, fecha final, y el catálogo.
Salidas	Array list con los shows que son estrenados en ese intervalo de tiempo.
Implementado (Sí/No)	Sí, Cesar Avellaneda, Esteban Castel, Nicolas Casas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Requerimiento 2	
Pasos	**Complejidad**
Paso 1: Sort	o(nlogn)
Paso 2: arreglos input	o(1)
Paso 3: arreglos input	o(1)
Paso 4: crear estructura	o(1)
Paso 5: recorrer catalogo por plataformas	o(4n) = n
Paso 6: recorrer cada show por plataforma	o(n)
Paso 7: condición	o(1)
Paso 8: condición dentro de la condicción anterior	o(1)
Paso 9: Crear diccionario	o(1)
Paso 10: Ciclo	o(n)
Paso 11: agregar contenido al diccionario	o(1)
Paso 12: AddLast	o(1)
Paso 13: Return	o(1)
Total	o(n^4logn) = o(n^3)

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	672,84 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator, newlist y addLast) import datetime (para modificar un str a un formato de fechas y trabajar más fácil) Recorrido por medio de 2 loop y un tercero después de 2 condiciones para agregar la información deseada a un diccionario
Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

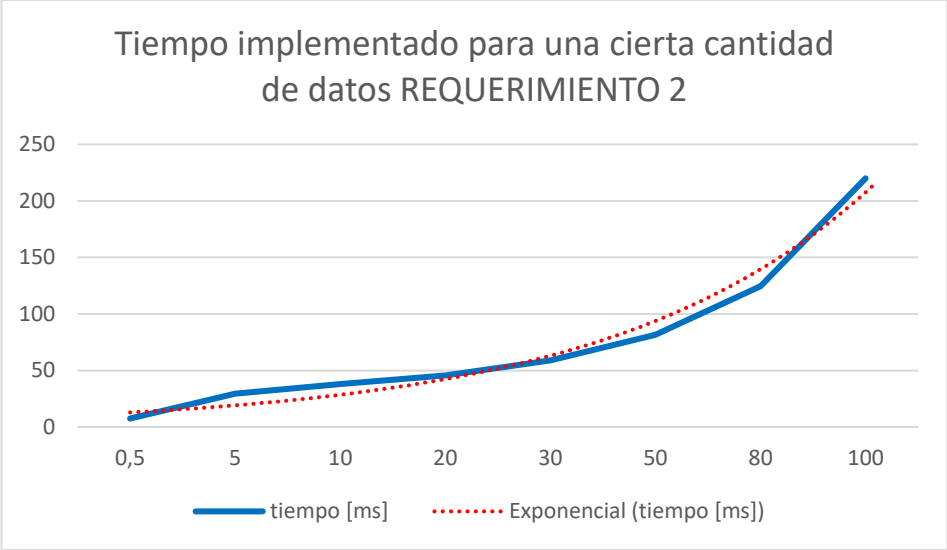
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	23,18
5	44,05
10	78,4
20	137,02
30	184,86
50	374,54
80	505,64
100	672,84

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la gráfica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.

Requerimiento 3

```
def requerimiento3(actorbuscar,catalog):
    sortcatalog(catalog, comparerequerimiento345)
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    tabla=lt.newList(datastructure="ARRAY_LIST")
    temp={"type":"count", "Movies":0, "TV Shows":0}
    actorbuscar=actorbuscar.lower().replace(" ", "")
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            cast=show["cast"].lower().replace(" ", "")
            cast=cast.split(",")
            for actor in cast:
                if actorbuscar
                    lt.addLast(respuesta,show)
                    if show["type"]=="Movie":
                        temp["Movies"]+=1
                    elif show["type"]=="TV Show":
                        temp["TV Shows"]+=1
    lt.addLast(tabla,temp)
    return (respuesta, tabla)

def requerimiento4(generobuscar,catalog):
```

(variable) respuesta: Any

Descripción

El tercer requerimiento tenia que encontrar los contenidos hechos por un actor pasado por parámetro, para lo que se recorre todo el diccionario y en cada show se sacan los diferentes actores usando la función. split(“,”) y para casi todos los strings que se tenían que comparar, se le quito los espacios y se puso todo en minúscula con las funciones .lower() y .replace(“ ”, “”).

Tras compararlo si el actor a buscar (al cual también se le hizo el mismo procedimiento de string) se encuentra en la lista de cast que se genera después del. split(), entonces añade el show a la respuesta y se actualiza el contador de tipo según el tipo del show que se esté analizando.

Esta función retorna una adtstructure para la tabla del contador y otra con los contenidos en los que participa el actor xd

Entrada	Actor a buscar y el catálogo.
Salidas	Array list con el conteo de tipo de contenido y otra array list la respuesta de los contenidos en que el actor participa.
Implementado (Sí/No)	Cesar Avellaneda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Requerimiento 3	
Pasos	**Complejidad**
Paso 1: Sort	o(nlogn)
Paso 2: crear estructura	o(1)
Paso 3: crear estructura tabla	o(1)
Paso 4: crear contador	o(1)
Paso 5: modificar str	o(1)
Paso 6: recorrer catalogo por plataformas	o(4n) = n
Paso 7: recorrer cada show por plataforma	o(n)
Paso 8: modificar str	o(2)
Paso 9: Recorrer el actor de cada show cast	o(n)
Paso 10: Condición si esta el actor que se desea buscar	o(1)
Paso 11: AddLast	o(1)
Paso 12: condiccion agregar al contador	o(4)
Pase 13: AddLast tabla	o(1)
Paso 14: Return	o(1)
Total	o(n^4logn) = o(n^3)

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	783,27 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator, newlist y addLast) Modificación de str (lower, replace y Split) Recorrido por medio de 3 loop luego una condición dentro otra condición para aumentar el contador (el contador es un diccionario con llaves y valores)
Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

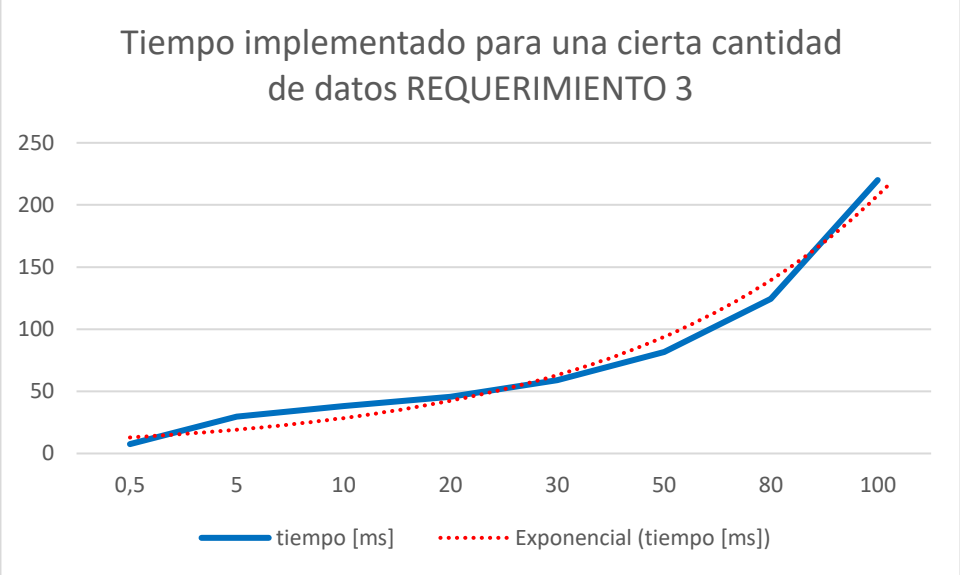
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	46,42
5	51,68
10	68,54
20	133,26
30	181,96
50	338,8
80	505,56
100	783,27

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la gráfica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.

Requerimiento 4

```
def requerimiento4(generobuscar,catalog):
    sortcatalog(catalog, comparerequerimiento345)
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    tabla=lt.newList(datastructure="ARRAY_LIST")
    temp={"type":"count", "Movies":0, "TV Shows":0}
    generobuscar=generobuscar.lower().replace(" ", "")
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            generos=show["listed_in"].lower().replace(" ", "")
            generos=generos.split(",")
            for g in generos:
                if generobuscar in g:
                    lt.addLast(respuesta,show)
                    if show["type"]=="Movie":
                        temp["Movies"]+=1
                    elif show["type"]=="TV Show":
                        temp["TV Shows"]+=1
    lt.addLast(tabla,temp)
    return (respuesta, tabla)
```

Descripción

El segundo requerimiento se parece mucho al primero, con la diferencia de que lo que se busca no son películas sino shows de televisión, además de esto el parámetro de búsqueda es la fecha en que se añadió, para lo que se tuvo que usar la librería datetime, en particular la función datetime.date.fromisoformat, la cual transforma los strings en forma (“YYYY-MM-DD”) en un valor que se pueda comparar, para que los shows que si cumplan con el condicional sean agregados a la datastructure que se da como retorno

Entrada	Fecha inicial, fecha final, y el catálogo.
Salidas	Array list con los shows que son estrenados en ese intervalo de tiempo.
Implementado (Sí/No)	Nicolas Casas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Requerimiento 4	
Pasos	**Complejidad**
Paso 1: Sort	o(nlogn)
Paso 2: crear estructura	o(1)
Paso 3: crear estructura tabla	o(1)
Paso 4: crear contador	o(1)
Paso 5: modificar str	o(1)
Paso 6: recorrer catalogo por plataformas	o(4n) = n
Paso 7: recorrer cada show por plataforma	o(n)
Paso 8: modificar str	o(2)
Paso 9: Recorrer todos los generos	o(n)
Paso 10: Condición si esta el genero que se desea buscar	o(1)
Paso 11: AddLast	o(1)
Paso 12: condiccion agregar al contador	o(4)
Pase 13: AddLast tabla	o(1)
Paso 14: Return	o(1)
Total	o(n^4logn) = o(n^3)

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	686,69 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator, newlist y addLast) Modificación de str (lower, replace y Split) Recorrido por medio de 3 loop luego una condición dentro otra condición para aumentar el contador (el contador es un diccionario con llaves y valores)
Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

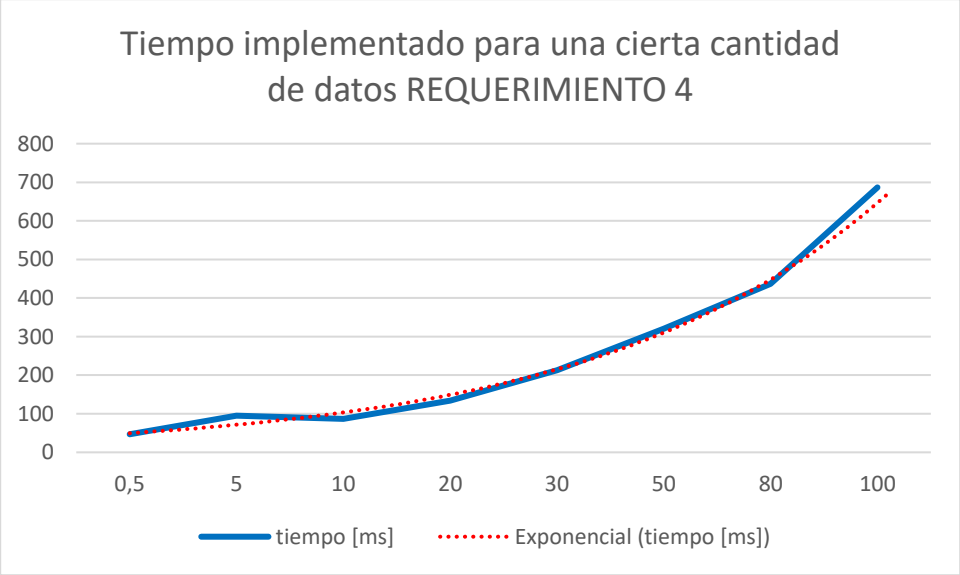
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	46,88
5	94,7
10	86,47
20	134,32
30	212,72
50	320,64
80	436,69
100	686,69

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la gráfica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.

Requerimiento 5

```
def requerimiento5(paisbuscar,catalog):
    sortcatalog(catalog, comparerequerimiento345)
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    tabla=lt.newList(datastructure="ARRAY_LIST")
    temp={"type":"count", "Movies":0, "TV Shows":0}
    paisbuscar=paisbuscar.lower().replace(" ", "")
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            paises=show["country"].lower().replace(" ", "")
            paises=paises.split(",")
            for p in paises:
                if paisbuscar in p:
                    lt.addLast(respuesta,show)
                    if show["type"]=="Movie":
                        temp["Movies"]+=1
                    elif show["type"]=="TV Show":
                        temp["TV Shows"]+=1
    lt.addLast(tabla,temp)
    return (respuesta, tabla)
```

Descripción

El segundo requerimiento se parece mucho al primero, con la diferencia de que lo que se busca no son películas sino shows de televisión, además de esto el parámetro de búsqueda es la fecha en que se añadió, para lo que se tuvo que usar la librería datetime, en particular la función datetime.date.fromisoformat, la cual transforma los strings en forma (“YYYY-MM-DD”) en un valor que se pueda comparar, para que los shows que si cumplan con el condicional sean agregados a la datastructure que se da como retorno

Entrada	Fecha inicial, fecha final, y el catálogo.
Salidas	Array list con los shows que son estrenados en ese intervalo de tiempo.
Implementado (Sí/No)	Esteban Castel

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Requerimiento 5	
Pasos	**Complejidad**
Paso 1: Sort	o(nlogn)
Paso 2: crear estructura	o(1)
Paso 3: crear estructura tabla	o(1)
Paso 4: crear contador	o(1)
Paso 5: modificar str	o(1)
Paso 6: recorrer catalogo por plataformas	o(4n) = n
Paso 7: recorrer cada show por plataforma	o(n)
Paso 8: modificar str	o(2)
Paso 9: Recorrer todos los paises	o(n)
Paso 10: Condición si esta el pais que se desea buscar	o(1)
Paso 11: AddLast	o(1)
Paso 12: condiccion agregar al contador	o(4)
Pase 13: AddLast tabla	o(1)
Paso 14: Return	o(1)
Total	o(n^4logn) = O(N^3)

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	639,69 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator, newlist y addLast) Modificación de str (lower, replace y Split) Recorrido por medio de 3 loop luego una condición dentro otra condición para aumentar el contador (el contador es un diccionario con llaves y valores)
Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

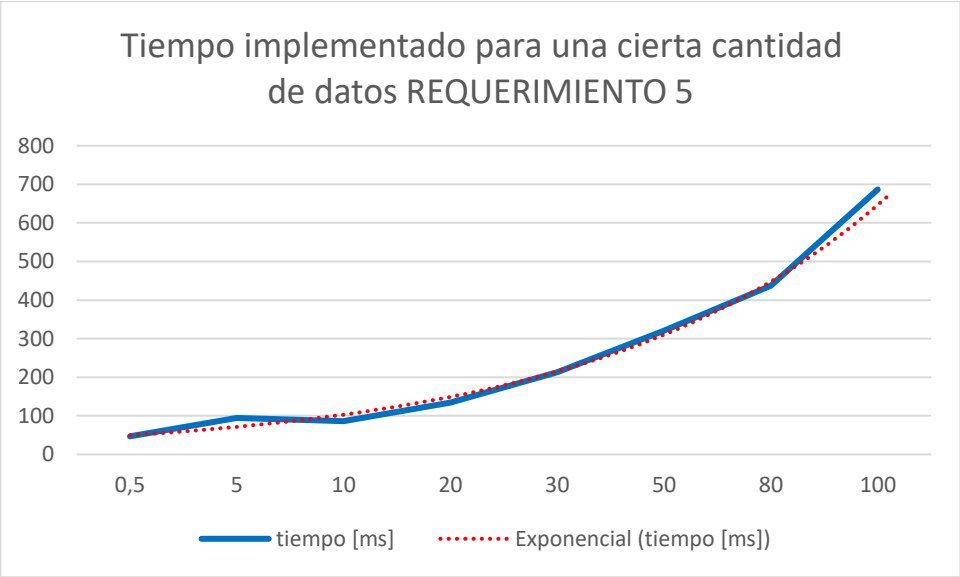
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	27,9
5	186,34
10	196,28
20	205,13
30	226,64
50	317,61
80	437,46
100	639,69

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la gráfica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.

Requerimiento 6

```
def requerimiento6(nombre, catalog):
    sortcatalog(catalog, comparerequerimiento168)
    tabla1=lt.newList(datastructure="ARRAY_LIST")
    tabla2=lt.newList(datastructure="ARRAY_LIST")
    tabla3=lt.newList(datastructure="ARRAY_LIST")
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    t1={"type":"count"}
    t2={"stream_service":"count"}
    t3={"listed_in":"count"}
    nombre=nombre.lower().replace(" ", "")
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            generos=show["listed_in"].lower().split(", ")
            directores= show["director"].lower().replace(" ", "").split(
            for director in directores:
                if director==nombre:
                    lt.addLast(respuesta, show)
                    if show["type"] not in t1.keys():
                        tipo=show["type"]
                        t1[tipo]=1
                    else:
                        tipo=show["type"]
                        t1[tipo]+=1
                    if show["stream_service"] not in t2.keys():
                        tipo=show["stream_service"]
                        t2[tipo]=1
                    else:
                        tipo=show["stream_service"]
                        t2[tipo]+=1
                    for genero in generos:
                        if genero not in t3.keys():
                            t3[genero]=1
                        else:
                            t3[genero]+=1
    lt.addLast(tabla1,t1)
    lt.addLast(tabla2,t2)
    lt.addLast(tabla3,t3)

    return tabla1,tabla2,tabla3,respuesta
```

Descripción

El requerimiento 6 tiene 4 retornos de adtstructure, 3 para las tablas de contadores de tipo, plataforma y genero de los contenidos del director que se está analizando y por ultimo los contenidos donde participa ese director.

Para luego en el view armar las 3 tablas de contadores y la tabla de los 3 primeros y últimos contenidos del director.

para cada uno de los contadores se hace un diccionario, para el cual en cada show se revisa si el atributo (tipo, plataforma y genero) ya está en el diccionario, en cuyo caso le suma 1 y en caso de no estar lo inicializa en 1.

Para los géneros se ponen en minúscula y se separan por .split(“, ”).

Para este requerimiento se hace lo mismo que con los actores, se transforman los strings del director por parámetro y el de cada uno de los contenidos y se hace una lista con .split(“, ”) de los directores y luego si el que se busca está en los del show, se agrega a la respuesta.

Entrada	El catalogo
Salidas	3 array list de contadores de tipo, plataforma y género y el array list de los contenidos del director.
Implementado (Sí/No)	Sí, Cesar Avellaneda, Esteban Castel, Nicolas Casas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 sort	O(n log n)
Paso 2 crear adts	O(4)
Paso 3 crear dicts	O(3)
Paso 4 ajuste string	O(1)
Paso 5 recorrer plataformas del catalogo	O(4n)=O(n)
Paso 6 recorrer shows de las plataformas	O(n) está dentro del anterior
Paso 7 recorrer directores de los shows	O(n) está dentro del anterior
Paso 8 recorrer géneros	O(n) está dentro del anterior
Paso 9 condiciones	O(7)
Paso 9 añadir a las respuestas	O(3)
TOTAL	***O(***n log n+ n^4***)*** *= O(N^5)*

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	835,15 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator, newlist y addLast) Modificación de str (lower, replace y Split) Recorrido por medio de 3 loop luego varias condiciones dentro otras condiciones para aumentar el contador (el contador es un diccionario con llaves y valores)
Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

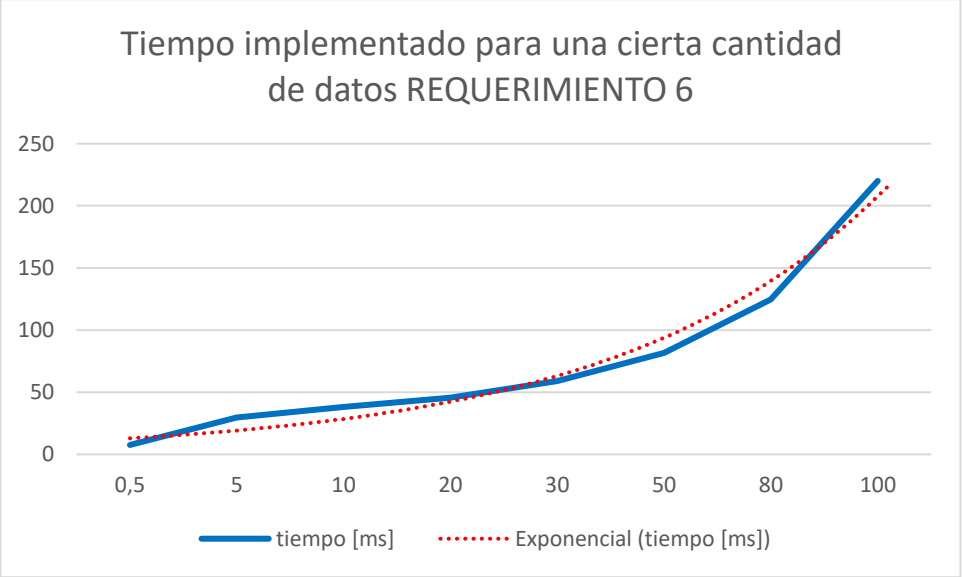
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	13,15
5	97,43
10	107,46
20	202,41
30	330,57
50	347,84
80	588,77
100	835,15

Graficas

Las gráficas con la representación de las pruebas realizadas.

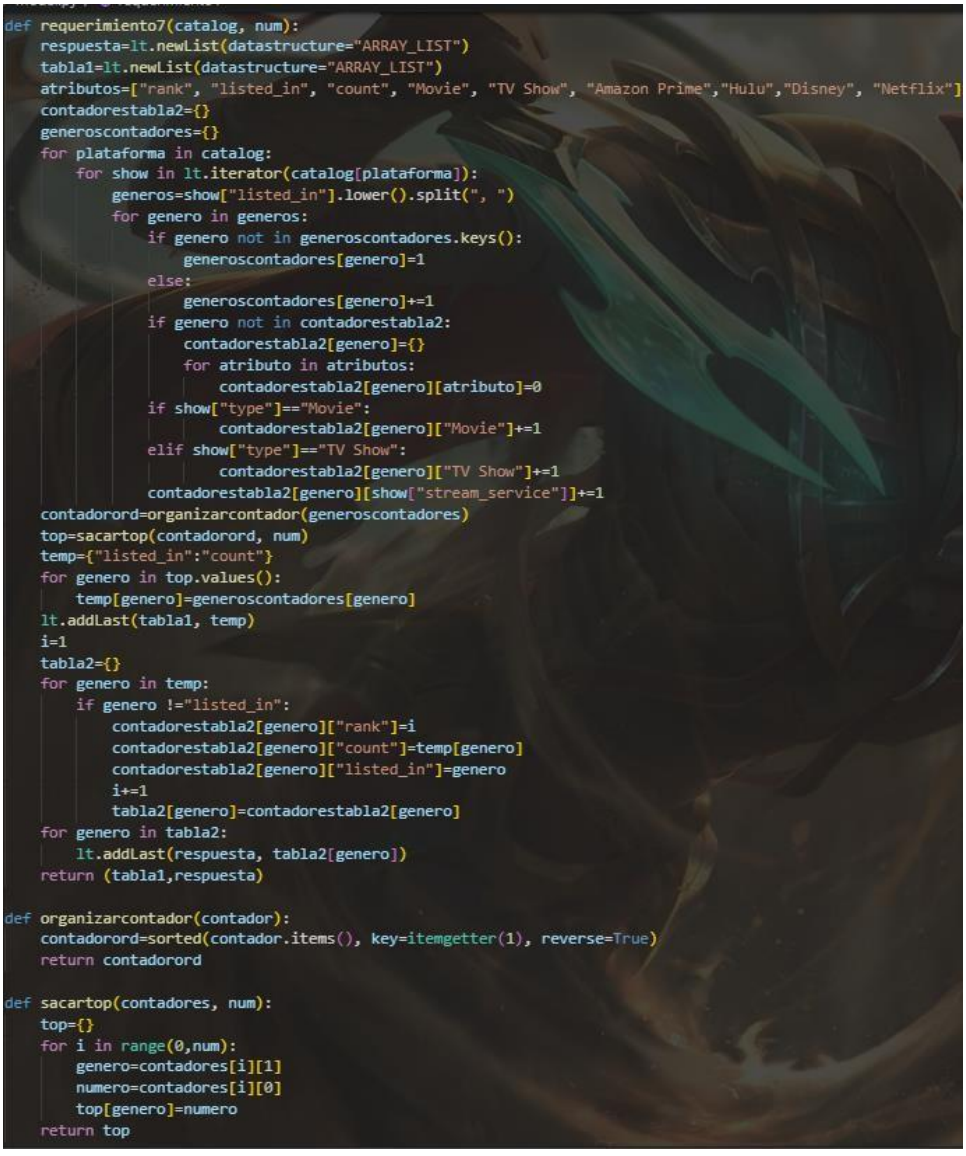


Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la gráfica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.

Requerimiento 7



```
def requerimiento7(catalog, num):
    respuesta=lt.newList(datastructure="ARRAY_LIST")
    tabla1=lt.newList(datastructure="ARRAY_LIST")
    atributos=["rank", "listed_in", "count", "Movie", "TV Show", "Amazon Prime","Hulu","Disney", "Netflix"]
    contadorestabla2={}
    generoscontadores={}
    for plataforma in catalog:
        for show in lt.iterator(catalog[plataforma]):
            generos=show["listed_in"].lower().split(", ")
            for genero in generos:
                if genero not in generoscontadores.keys():
                    generoscontadores[genero]=1
                else:
                    generoscontadores[genero]+=1
                if genero not in contadorestabla2:
                    contadorestabla2[genero]={}
                    for atributo in atributos:
                        contadorestabla2[genero][atributo]=0
                if show["type"]=="Movie":
                        contadorestabla2[genero]["Movie"]+=1
                elif show["type"]=="TV Show":
                        contadorestabla2[genero]["TV Show"]+=1
                contadorestabla2[genero][show["stream_service"]]+=1
    contadorord=organizarcontador(generoscontadores)
    top=sacartop(contadorord, num)
    temp={"listed_in":"count"}
    for genero in top.values():
        temp[genero]=generoscontadores[genero]
    lt.addLast(tabla1, temp)
    i=1
    tabla2={}
    for genero in temp:
        if genero !="listed_in":
            contadorestabla2[genero]["rank"]=i
            contadorestabla2[genero]["count"]=temp[genero]
            contadorestabla2[genero]["listed_in"]=genero
            i+=1
            tabla2[genero]=contadorestabla2[genero]
    for genero in tabla2:
        lt.addLast(respuesta, tabla2[genero])
    return (tabla1,respuesta)

def organizarcontador(contador):
    contadorord=sorted(contador.items(), key=itemgetter(1), reverse=True)
    return contadorord

def sacartop(contadores, num):
    top={}
    for i in range(0,num):
        genero=contadores[i][1]
        numero=contadores[i][0]
        top[genero]=numero
    return top
```

Descripción

Para este requerimiento lo que pensamos e hicimos fue un diccionario que tiene como llave cada uno de los diferentes géneros que se encuentran en el catálogo y cada uno tiene como valor las veces que

aparece, después de esto se organiza el diccionario de mayor a menor usando sorted e itemgetter, para luego sacar el top n de géneros que se pasa por parámetro. De esto se saca la tabla del top.
Por otra parte, usando las llaves del top y el contador de géneros se saca otro diccionario que tiene cada uno de los géneros del top con su contador.

Entrada	Numero de top y el catálogo.
Salidas	Datos para la tabla del contador del top y el top de géneros con todos los atributos que se piden para la tabla
Implementado (Sí/No)	Sí, Cesar Avellaneda, Esteban Castel, Nicolas Casas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 crear respuesta	O(1)
Paso 2 crear tabla1	O(1)
Paso 3 diccionario de atributos y contadores	O(3)
Paso 4 recorrer catalogo por plataformas	O(4n)= O(n)
Paso 5 recorrer géneros	O(n) está dentro del anterior
Paso 6 recorrer atributos	O(n) está dentro del anterior
Paso 7 condiciones	O(5)
Paso 8 organizar contador	O(1)
Paso 9 dict tabla 1	O(n)
Paso 10 sacar top	O(n)
Paso 11 agregar tabla 1 a adt	O(1)
Paso 12 llenar dict tabla2	O(n)
Paso 13 agregar dict tabla 2 a tabla 2	O(1)
TOTAL	***O(3n+n^3)= O(n^3)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución (large)	220,02 ms
Condiciones, herramientas y recursos utilizados:	DISClib.ADT import list (se usa iterator, newlist y addLast) Modificación de str (lower, replace y Split) Recorrido por medio de 3 loop luego varias condiciones que aumentan el diccionario el cual se ordena por medio de sorted e itemgetter para devolver el ranking

Computador donde se ejecuta:	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 GB Microsoft Windows 10 Home Single Language)

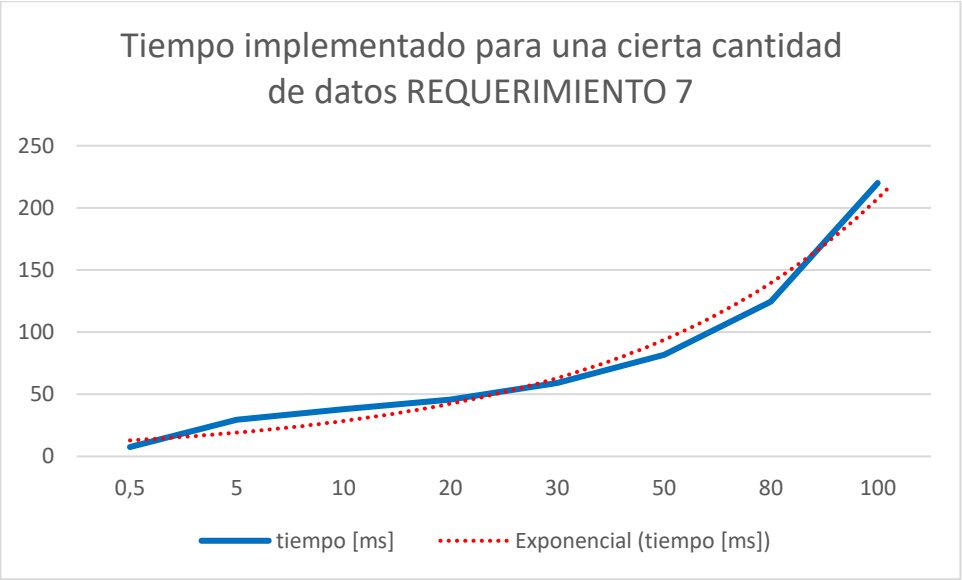
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

porcentaje datos	tiempo [ms]
0,5	7,54
5	29,58
10	38,08
20	45,72
30	59,04
50	81,71
80	124,54
100	220,02

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Analizando lo que realiza nuestra función y la manera como lo realizamos, junto al análisis de complejidad, el computador donde se ejecutaron las pruebas, el tiempo en el que transcurre la función con una cantidad de datos específicos y su respectiva gráfica. Nos damos cuenta de que todas estás descripciones van acorde a lo estudiando en el curso hasta el momento y todo se acota perfectamente de acuerdo con lo anteriormente estipulado en especial la gráfica con sus respectivos datos y el análisis de complejidad.

También analizamos que recorrer depende de la cantidad de elementos por eso es O(n), y esto es lo mismo que o(1) * n donde n es la cantidad de elementos que existen, el mejor de los casos seria que no encontrara lo que se desea buscar, también lo que se demora en sacar los elementos que deseo buscar es o(n) porque busca en todos los elementos y saca los que coincidan por ende la cantidad de ejecuciones del recorrido de los for es o(n). Finalmente guardar algo, modificar algo o crear algo en la memoria vale o(1) es constante.