OBSERVACIONES DEL LA PRACTICA

Gregorio Salazar 202022085

Valentina Uribe Salcedo 201817485

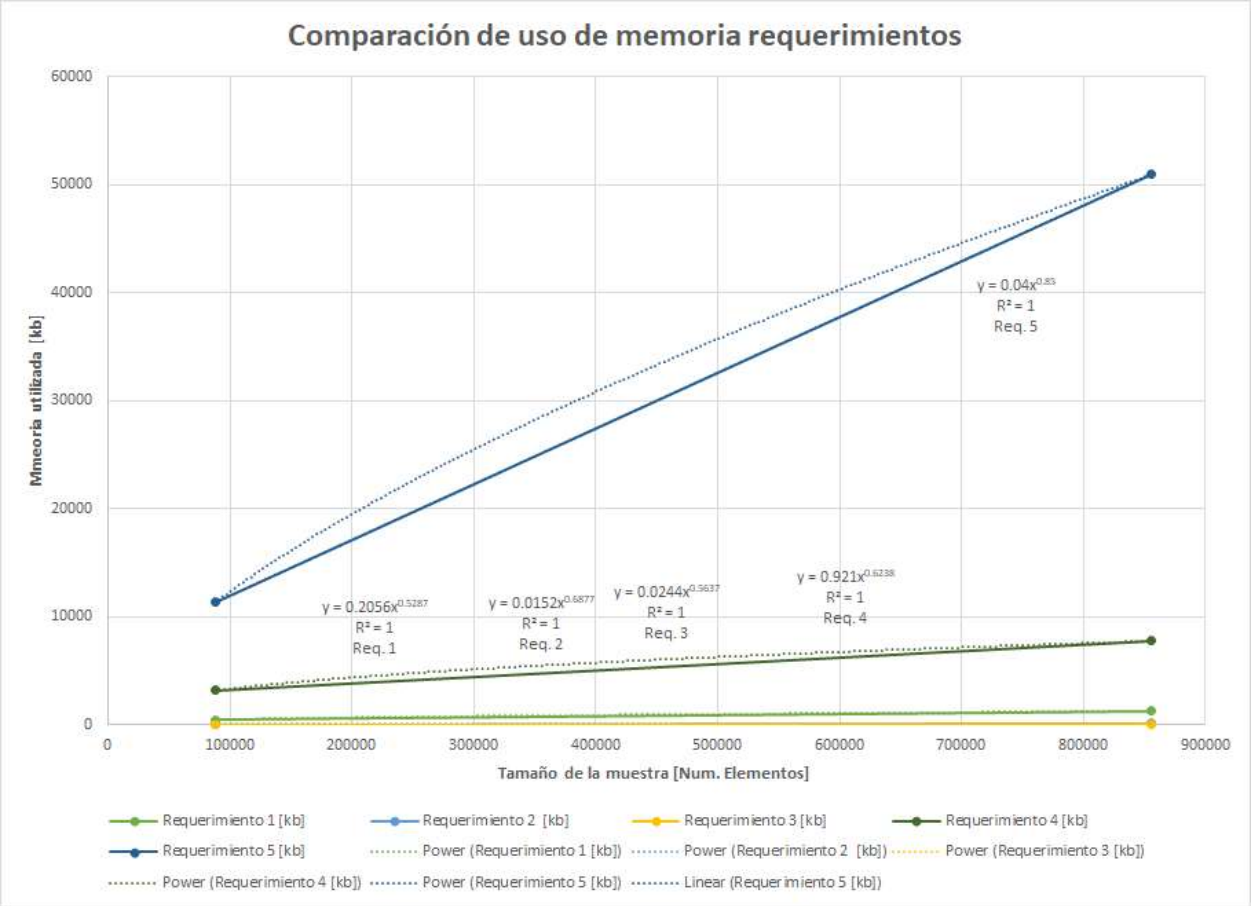
	Máquina 1	Máquina 2
Procesadores	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz	AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx
Memoria RAM (GB)	8	16
Sistema Operativo	Windows 10	Windows 10

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Maquina 1

Tamaño de la muestra	**Requerimiento 1 [ms]**	**Requerimiento 2 [ms]**	**Requerimiento 3 [ms]**	**Requerimiento 4 [ms]**	**Requerimiento 5 [ms]**
87589	84.36	38.18	14.88	1115.76	636.44
855422	281.47	183.01	53.76	4623.91	4416.12

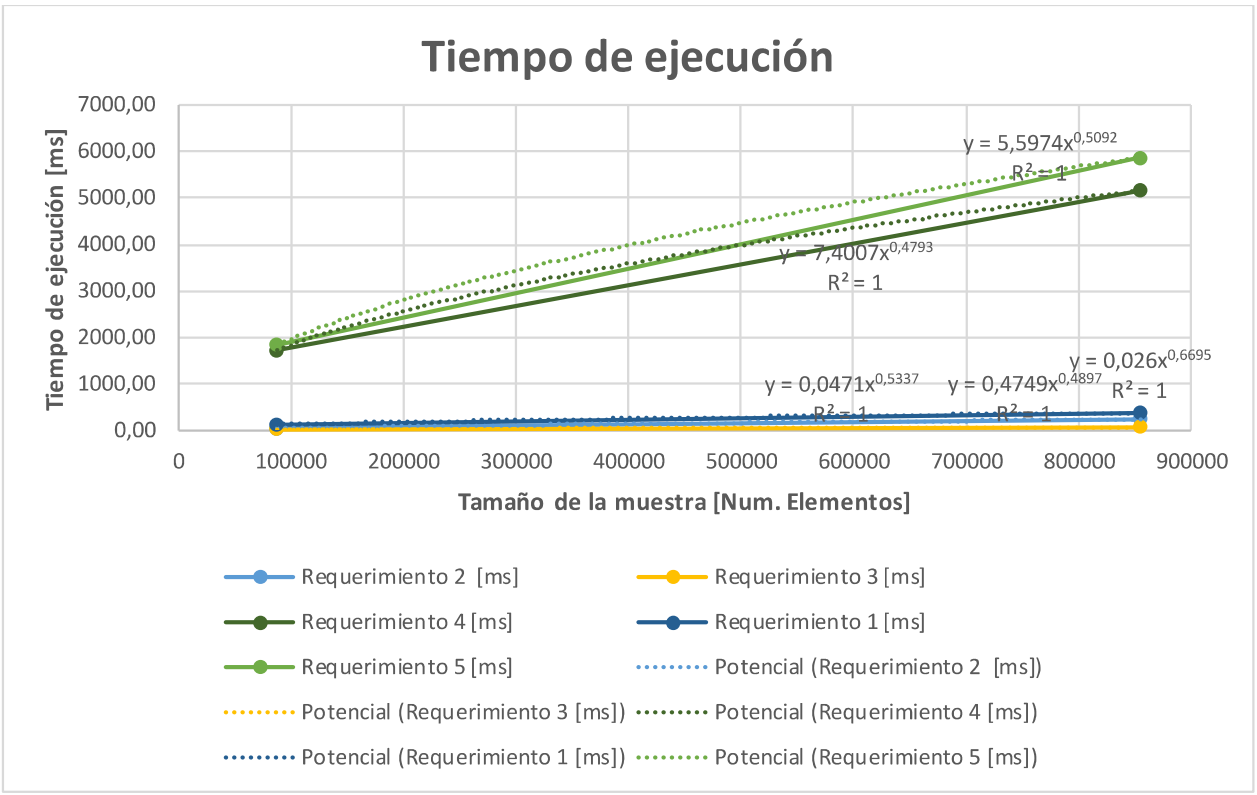
Tamaño de la muestra	Requerimiento 1 [kb]	Requerimiento 2 [kb]	Requerimiento 3 [kb]	Requerimiento 4 [kb]	Requerimiento 5 [kb]
87589	430.543	30.73	7.98	3142.90	11296.27
855422	1247.20	66.99	19.97	7731.73	50955.30

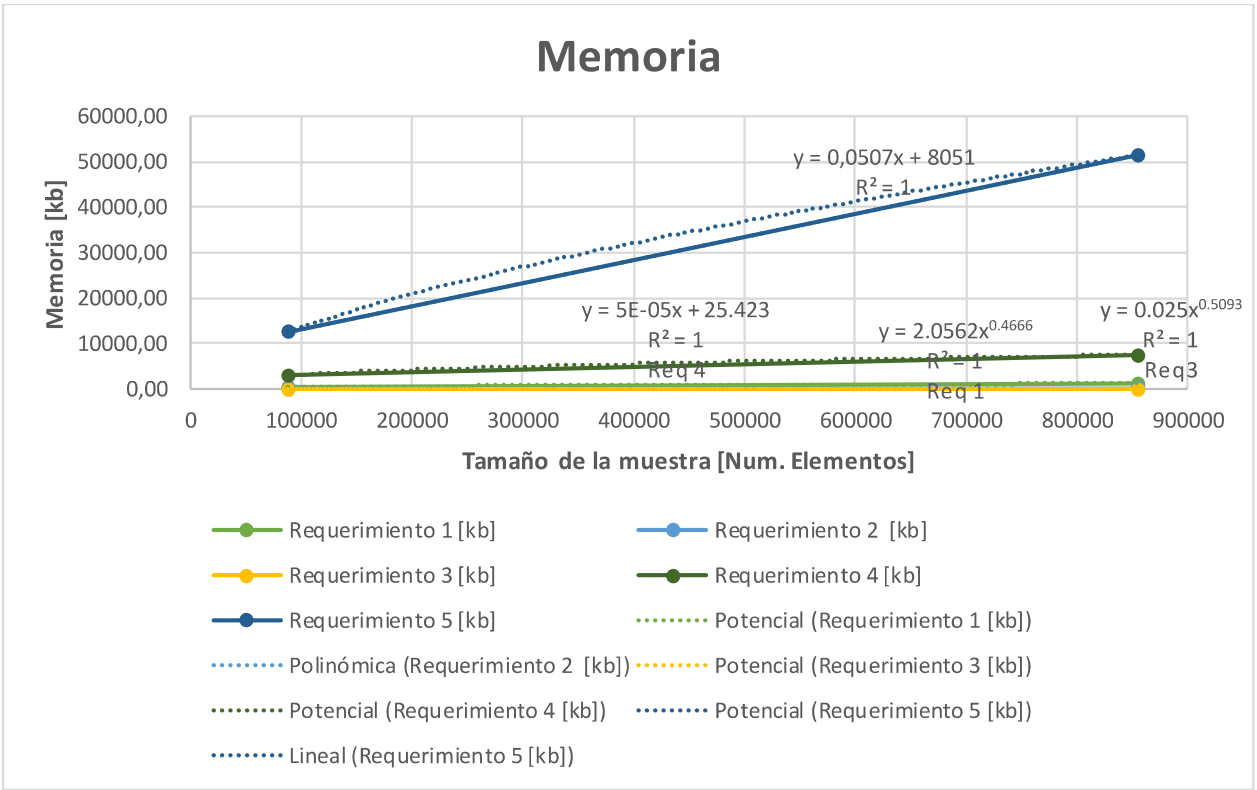


Maquina 2

Tamaño de la muestra (ARRAYLIST)	Requerimiento 1 [ms]	Requerimiento 2 [ms]	Requerimiento 3 [ms]	Requerimiento 4 [ms]	Requerimiento 5 [ms]
87589	125.057	52.87	20.47	1731.40	1840.34
855422	381.79	243.13	69.09	5162.00	5873.70

Tamaño de la muestra (LINKED_LIST)	Requerimiento 1 [kb]	Requerimiento 2 [kb]	Requerimiento 3 [kb]	Requerimiento 4 [kb]	Requerimiento 5 [kb]
87589	416.34	30.17	8.23	3039.39	12495.02
855422	1205.87	71.77	26.26	7480.83	51452.34


Tiempo de ejecución
Tiempo de ejecución [ms]
7000,00
6000,00
5000,00
4000,00
3000,00
2000,00
1000,00
0,00
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
Tamaño de la muestra [Num. Elementos]
y = 5,5974x^0,5092
R² = 1
y = 7,4007x^0,4793
R² = 1
y = 0,0471x^0,5337
R² = 1
y = 0,4749x^0,4897
R² = 1
y = 0,026x^0,6695
R² = 1
Requerimiento 2 [ms]
Requerimiento 3 [ms]
Requerimiento 4 [ms]
Requerimiento 1 [ms]
Requerimiento 5 [ms]
Potencial (Requerimiento 2 [ms])
Potencial (Requerimiento 3 [ms])
Potencial (Requerimiento 4 [ms])
Potencial (Requerimiento 1 [ms])
Potencial (Requerimiento 5 [ms])


Memoria
Memoria [kb]
60000,00
50000,00
40000,00
30000,00
20000,00
10000,00
0,00
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
Tamaño de la muestra [Num. Elementos]
y = 0,0507x + 8051
R² = 1
y = 5E-05x + 25.423
R² = 1
Req 4
y = 2.0562x^0.4666
R² = 1
Req 1
y = 0.025x^0.5093
R² = 1
Req3
Requerimiento 1 [kb]
Requerimiento 2 [kb]
Requerimiento 3 [kb]
Requerimiento 4 [kb]
Requerimiento 5 [kb]
Potencial (Requerimiento 1 [kb])
Polinómica (Requerimiento 2 [kb])
Potencial (Requerimiento 3 [kb])
Potencial (Requerimiento 4 [kb])
Potencial (Requerimiento 5 [kb])
Lineal (Requerimiento 5 [kb])

Analisis complejidad O.

Requerimiento 1.

```
def get_events_characteristic(catalog,characteristic_index,lo,hi):
    characteristic_tree=catalog[characteristic_index]
    lst=om.values(characteristic_tree,lo,hi)
    totevents=0
    artists=m.newMap(maptype='Probing',loadfactor=0.5)
    for entry in lt.iterator(lst):
        totevents+=lt.size(entry['lstevents'])
        for artist in lt.iterator(m.keySet(entry['artists'])):
            m.put(artists,artist,None)
    return totevents,m.size(artists),m.keySet(artists)
```

En el peor de los casos la complejidad de la función es O(n) ya que om.values() tiene una complejidad en el peor de los casos O(n) y los dos ciclos realizados tienen en conjunto una complejidad O(n) ya que aunque son un ciclo dentro de otro, solo recorren una vez la lista de los valores del árbol entre lo y hi con los respectivos artistas de cada evento.

Requerimiento 2.

```
def get_tracks_party(catalog,lo1,hi1,lo2,hi2):
    energy_tree=catalog['energy_index']
    lst=om.values(energy_tree,lo1,hi1)
    energy_dance_list=lt.newList('ARRAY_LIST')
    for entry in lt.iterator(lst):
        for track in lt.iterator(m.valueSet(entry['tracks'])):
            if lo2<=float(track['danceability'])<=hi2:
                lt.addLast(energy_dance_list,track)
    return energy_dance_list
```

En el peor de los casos la complejidad de la función es O(n) ya que om.values() tiene una complejidad en el peor de los casos O(n). De igual forma en el requerimiento 1, los dos ciclos realizados tienen en conjunto una complejidad O(n) ya que, aunque son un ciclo dentro de otro, solo recorren una vez la lista de los valores del árbol entre lo y hi con las respectivas canciones. Luego se revisa la danzahabilidad para ver si está entre los valores que entran por parámetro, esta operación es O(1).

Requerimiento 3.

```
def get_tracks_study(catalog,lo1,hi1,lo2,hi2):
    intrumentalness_tree=catalog['instrumentalness_index']
    lst=om.values(intrumentalness_tree,lo1,hi1)
    instru_tempo_list=lt.newList('ARRAY_LIST')
    for entry in lt.iterator(lst):
        for track in lt.iterator(m.valueSet(entry['tracks'])):
            if lo2<=float(track['tempo'])<=hi2:
                lt.addLast(instru_tempo_list,track)
    return instru_tempo_list
```

En el peor de los casos la complejidad de la función es O(n) ya que om.values() tiene una complejidad en el peor de los casos O(n). De igual forma en el requerimiento 2, los dos ciclos realizados tienen en conjunto una complejidad O(n) ya que, aunque son un ciclo dentro de otro, solo recorren una vez la lista de los valores del árbol entre lo y hi con las respectivas canciones. Luego se revisa el tempo para ver si está entre los valores que entran por parámetro, esta operación es O(1).

Requerimiento 4.

```
def add_genre(catalog,generos,name,lo,hi):
    catalog['tempo_generos_editable'][name]=(lo,hi)
    generos.append(name)


def get_events_by_genero(catalog,generos):
    total=0
    lista=lt.newList('ARRAY_LIST')
    for genero in generos:
        lo=catalog['tempo_generos_editable'][genero][0]
        hi=catalog['tempo_generos_editable'][genero][1]

ans_genero=get_events_characteristic(catalog,'tempo_index',lo,hi),genero,lo,hi
        lt.addLast(lista,ans_genero)
        total+=ans_genero[0][0]
```

```
    return total,lista
```

La primera función es claramente O(1). Después, como hay muy pocos generos (del orden de 10) el recorrido de generos no crea un O(n). Como la función get_events_characteristic tiene complejidad O(n), el requerimiento tendría complejidad O(10*n) (está dentro del ciclo) que es lo mismo que O(n).

Requerimiento 5.

```
def get_vader(catalog, arreglo,total):
    entry_most_rep = lt.getElement(arreglo,1)
    tracks = entry_most_rep[2]
    tracks = m.valueSet(tracks)
    lista=lt.newList('ARRAY_LIST')
    for track in lt.iterator(tracks):
        hashtags = track['hashtag']
        suma = 0
        mean = 0
        for hashtag in lt.iterator(hashtags):
            entry = m.get(catalog["values"], hashtag)
            if entry is not None:
                sentiments=me.getValue(entry)
                vader = sentiments["vader_avg"]
                if vader != "":
                    vader = float(vader)
                    suma += vader

        if lt.size(hashtags) != 0:
            mean = suma/lt.size(hashtags)
        lt.addLast(lista,(track["track_id"],lt.size(hashtags), mean))
    return lista,arreglo,total


def req5(catalog,lo,hi):
```

```
    lst=om.values(catalog['times'],lo,hi)
    arreglo=lt.newList('ARRAY_LIST')
    total=0
    for genero in catalog['tempo_generos']:
        mapa=m.newMap(10,maptype='Probing',loadfactor=0.5)
        entry=[genero,0,mapa]
        lt.addLast(arreglo,entry)
    for nodo in lt.iterator(lst):
        for event in lt.iterator(nodo['lstevents']):
            i=1
            total+=1
            for genero in catalog['tempo_generos']:
                lo=catalog['tempo_generos'][genero][0]
                hi=catalog['tempo_generos'][genero][1]
                if lo<=float(event['tempo'])<=hi:
                    entry=lt.getElement(arreglo,i)
                    entry[1]+=1
                    m.put(entry[2],event['track_id'],event)
                i+=1
    arreglo = merge.sort(arreglo,comparar_info_req5)
    return get_vader(catalog,arreglo,total)
```

La funcion get_vader toma el vader promedio de las canciones del genero más escuchado. Dentro del primer for in, hay un for in de hashtags, pero este no afecta ya que la cantidad de hashtags de cada canción no es del orden de n, donde n es la cantidad de canciones. La cantidad de hashtags es tan solo entre 1 y máximo 10 (incluso si fuera 100 no afectaría). Como todas las demás operaciones en el ciclo son O(1), la complejidad de la función O(n). Para la función principal, como explicamos antes, la función om.values y el doble recorrido de los nodos, son en realidad uno solo, por lo que hasta aquí llevaría complejidad O(n). A la hora de recorrer los generos, estos son solamente 9 y el for está por comodidad, por lo que no aumenta la complejidad. Todas las operaciones acá son O(1), por lo que la complejidad de todos los for es en conjunto O(n). La ultima fila ordena el arreglo, que tiene tan solo 9 entradas, por lo

que al ser un ordenamiento tan pequeño, y que no varía con n, es constante O(1). Finalmente el requerimiento tiene en conjunto una complejidad de O(n).