OBSERVACIONES RETO 1

Req. 3 - Abel Arismendy, 202020625, a.arismendy@uniandes.edu.co

Req. 2- David Pérez 202123314 d.perezc23@uniandes.edu.co

Análisis de complejidad de los requerimientos

Carga de datos:

O(N) debido a que solo se recorren los datos una vez por archivo.

```
def loadArtists(self, file_size):
        artistsfile = cf.data_dir + f'Spotify/spotify-artists-utf8-{file_size}.csv'
        input_file = csv.DictReader(open(artistsfile, encoding='utf-8'))
        for artist in input_file:
            ar = Model.createArtist(artist)
            ar.trackName(self.catalog.tracksId.mapa)
            self.catalog.artistsId.addElement(ar.id, ar)
            self.catalog.artistsName.addElement(ar.name, ar)
            self.catalog.artists.addLast(ar)
            self.catalog.artistsInPop.addArtistInPop(ar)
            self.catalog.artistsIdMarket.addArtistId(ar)
            self.catalog.artistsIdAlbums.addArtistId(ar)

    def loadAlbums(self, file_size):
        albumsfile = cf.data_dir + f'Spotify/spotify-albums-utf8-{file_size}.csv'
        input_file = csv.DictReader(open(albumsfile, encoding='utf-8'))
        for album in input_file:
            al = Model.createAlbum(album)
            al.artistName(self.catalog.artistsId.mapa)
            al.trackName(self.catalog.tracksId.mapa)
            self.catalog.albumsByYear.addAlbumYear(al)
            self.catalog.albumsId.addElement(al.id, al)
            self.catalog.albums.addLast(al)
            mapsToAdd = self.catalog.artistsIdMarket.getArtistsMap([al.artist_id])
            for artist in mapsToAdd:
                artist.addAlbumByMarket(al)

            mapsToAdd = self.catalog.artistsIdAlbums.getArtistsMap([al.artist_id])
            for artist in mapsToAdd:
                artist['albumsType'].addAlbumByType(al)
```

```
            artist['albums_list'].addLast(al)

    def loadTracks(self, file_size):
        tracksfile = cf.data_dir + f'Spotify/spotify-tracks-utf8-{file_size}.csv'
        input_file = csv.DictReader(open(tracksfile, encoding='utf-8'))
        for track in input_file:
            tr = Model.createTrack(track)
            tr.albumInfo(self.catalog.albumsId.mapa)
            tr.artistsName(self.catalog.artistsId.mapa)
            self.catalog.tracksByPop.addTrackPop(tr)
            self.catalog.tracks.addLast(tr)
            mapsToAdd = self.catalog.artistsIdMarket.getArtistsMap(tr.artists_id)
            for artist in mapsToAdd:
                artist.addTrackByMarket(tr)

            self.catalog.albumTracks.addTrack(tr)

            # mapsToAdd = self.catalog.artistsIdAlbums.getArtistsMap(tr.artists_id)

    def loadTracksId(self, file_size):
        tracksfile = cf.data_dir + f'Spotify/spotify-tracks-utf8-{file_size}.csv'
        input_file = csv.DictReader(open(tracksfile, encoding='utf-8'))
        for track in input_file:
            tr = Model.createTrack(track)
            self.catalog.tracksId.addElement(tr.id, tr)
```

Requerimiento 1:

O(M log M) debido a que las funciones de consulta al mapa son O(1); pero, al obtener la lista de álbumes de un año se deben ordenar y esto tiene una complejidad de O(M log M) con el algoritmo merge. M es el tamaño de la lista de álbumes del año correspondiente. Varía según el año.

```
    def two(catalog, year):
        albumsInYear_count = mp.get(catalog.mapa, year)
        albumsInYear = me.getValue(albumsInYear_count)['list']
        albumsInYear.mergeSort()
        jan_count = me.getValue(albumsInYear_count)['jan_count']
        return albumsInYear, jan_count
```

Requirimiento 2:

O(M log M) debido a que las funciones de consulta al mapa son O(1); pero, al obtener la lista de artistas de una popularidad se deben ordenar y esto tiene una complejidad de O(M log M) con el algoritmo merge. M es el tamaño de la lista de artistas con la popularidad correspondiente. Varía según la popularidad.

```
def three(catalog, pop):
    artistsInPop = mp.get(catalog.mapa, (pop + ".0"))
    artistsInPop = me.getValue(artistsInPop)
    artistsInPop.mergeSort()
    return artistsInPop
```

Requerimiento 3:

O(M log M) debido a que las funciones de consulta al mapa son O(1); pero, al obtener la lista de canciones de una popularidad se deben ordenar y esto tiene una complejidad de O(M log M) con el algoritmo merge. M es el tamaño de la lista de canciones con la popularidad correspondiente. Varía según la popularidad.

```
def four(catalog, pop):
    tracksByPop = mp.get(catalog.mapa, pop)
    tracksByPop = me.getValue(tracksByPop)
    tracksByPop.mergeSort()
    return tracksByPop
```

Requerimiento 4:

O(M log M) debido a que las funciones de consulta al mapa son O(1); pero, al obtener la lista de canciones de un artista en un mercado especifico se deben ordenar y esto tiene una complejidad de O(M log M) con el algoritmo merge. M es el tamaño de la lista de canciones de un artista en un mercado especifico. Varía según el artista y el mercado.

```
def five(self, catalog, artist_name, market, artistsName):
    artist_id = mp.get(artistsName.mapa, artist_name)
    artist_id = me.getValue(artist_id).id
    market = self.formatMarket(market)
    marketsByArtist = mp.get(catalog.mapa, artist_id)
    marketsByArtist = me.getValue(marketsByArtist)
    artistMarket = mp.get(marketsByArtist.mapa, market)
    artistMarket = me.getValue(artistMarket)
    albumsArtistMarket = artistMarket['albums']
    tracksArtistMarket = artistMarket['tracks']

    tracksArtistMarket.mergeSort()

    return albumsArtistMarket, tracksArtistMarket
```

Requerimiento 5:

O(M log M) debido a que las funciones de consulta al mapa son O(1); pero, al obtener las listas de canciones y álbumes de un artista se deben ordenar para obtener la canción más popular y esto tiene una complejidad de O(M log M) con el algoritmo merge. M es el tamaño de la lista de canciones de un artista y de un album especifico. Varía según el artista y el albúm.

```
    def six(self, catalog, artist_name, artistsName, albumTracks):
        artist_id = mp.get(artistsName.mapa, artist_name)
        artist_id = me.getValue(artist_id).id
        artistAlbums = mp.get(catalog.mapa, artist_id)
        artistAlbums = me.getValue(artistAlbums)
        albumsArtistList = artistAlbums['albums_list']
        albumsArtistType= artistAlbums['albumsType']
        tracks = []
        for album in albumsArtistList.print():
            key = album.id
            if mp.contains(albumTracks.mapa, key):
                albumWithTracks = mp.get(albumTracks.mapa, key)
                albumWithTracks = me.getValue(albumWithTracks)
                if albumWithTracks.getSize() > 0:
                    albumWithTracks.mergeSort()
                    tracks.append(albumWithTracks.firstElement())

        albumsArtistTypeInfo= {'compilation': 0, 'single': 0, 'album': 0}
        for k in albumsArtistTypeInfo.keys():
            # print(mp.keySet(albumsArtistType.mapa))
            if mp.contains(albumsArtistType.mapa, k):
                size = mp.get(albumsArtistType.mapa, k)
                size = me.getValue(size)
                size = size['albums_list'].getSize()
                albumsArtistTypeInfo[k] = size

        return albumsArtistList, albumsArtistTypeInfo, tracks
```

Nota: Cómo el tamaño de M en la mayoría de los requerimientos es muy pequeño, no se aprecia una diferencia importante cuando se aumenta la cantidad de datos N.

Memoria:

Con respecto a la memoria, es constante en todos los requerimientos. Para la carga de datos, tiene un crecimiento de 0(N).

Ambientes de pruebas

	Máquina 1	Máquina 2
Procesadores	Intel Core i5 10400 6 núcleos 12 hilos, 2.9ghz	AMD Ryzen 5 5600x 6 núcleos 12 hilos 3.7ghz
Memoria RAM (GB)	32	16gb
Sistema Operativo	Windows 11	Windows 10

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

Maquina 1

Resultados

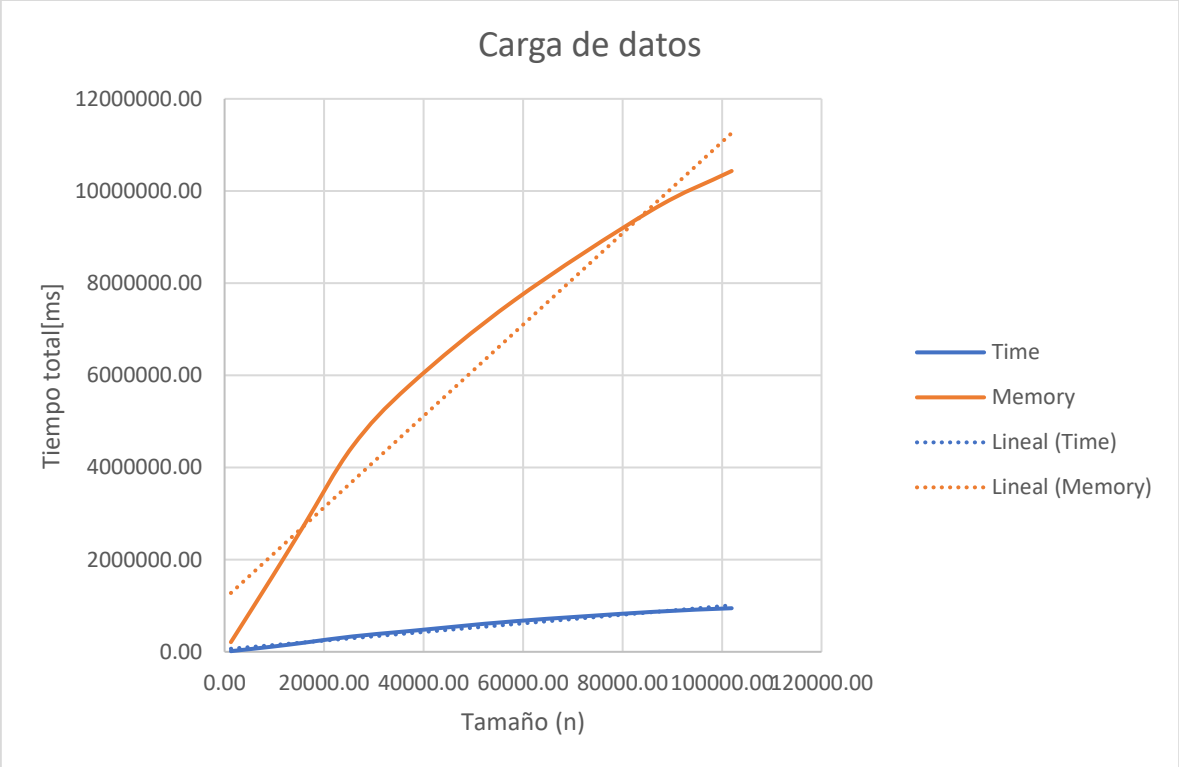
Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAY_LIST)	Carga de datos [ms]	Requerimiento 1 [ms]	Requerimiento 2 [ms]	Requerimiento 3 [ms]	Requerimiento 4[ms]	Requerimiento 5 [ms]
0.5%	1270.00	14010.86	0.44	0.23	0.04	111.55	0.32
5.0%	9625.00	112673.58	3.63	0.83	0.14	114.67	0.32
10.0%	16137.00	199967.64	7.27	0.74	0.26	113.93	0.31
20.0%	25718.00	333521.27	12.98	0.79	0.70	113.34	0.33
30.0%	37241.00	451251.27	19.61	0.81	0.69	113.35	0.32
50.0%	58216.00	662194.79	40.20	6.29	1.96	124.24	1.01
80.0%	85685.00	864097.11	52.30	1.34	1.94	116.37	0.75
100.0%	101939.00	945664.70	66.31	27.74	43.60	135.31	6.06

Tabla 2. Comparación de tiempos de ejecución para los requerimientos

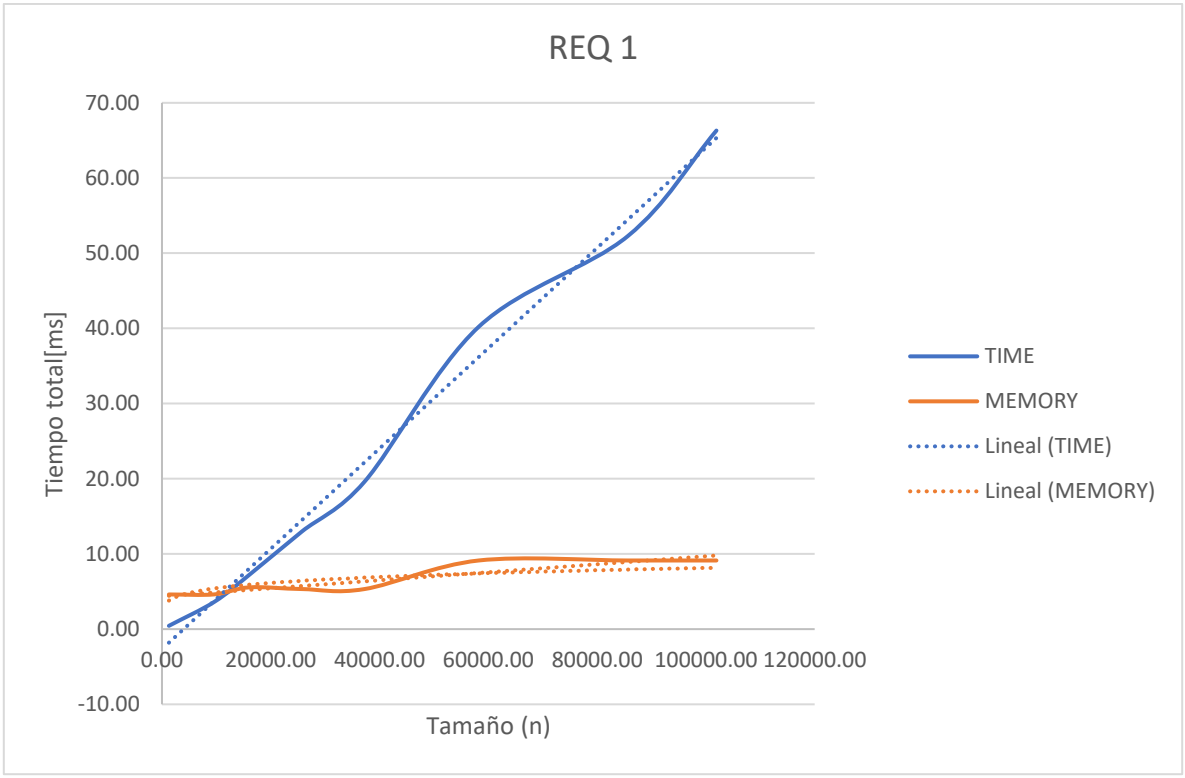
Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAY_LIST)	Carga de datos [kB]	Requerimiento 1 [kB]	Requerimiento 2 [kB]	Requerimiento 3 [kB]	Requerimiento 4[kB]	Requerimiento 5 [kB]
0.5%	1270.00	209523.62	4.61	0.17	0.17	1.40	2.77

5.0%	9625.00	1636428.29	4.62	0.17	0.17	0.87	2.77
10.0%	16137.00	2777162.98	5.56	0.17	0.17	0.87	2.77
20.0%	25718.00	4457436.74	5.33	0.17	0.17	0.87	2.77
30.0%	37241.00	5786595.14	5.33	0.17	0.17	0.87	2.77
50.0%	58216.00	7625362.33	9.13	0.17	0.17	1.93	2.77
80.0%	85685.00	9571805.37	9.13	0.17	0.17	1.40	2.77
100.0%	101939.00	10434704.92	9.13	0.17	0.17	1.40	2.77

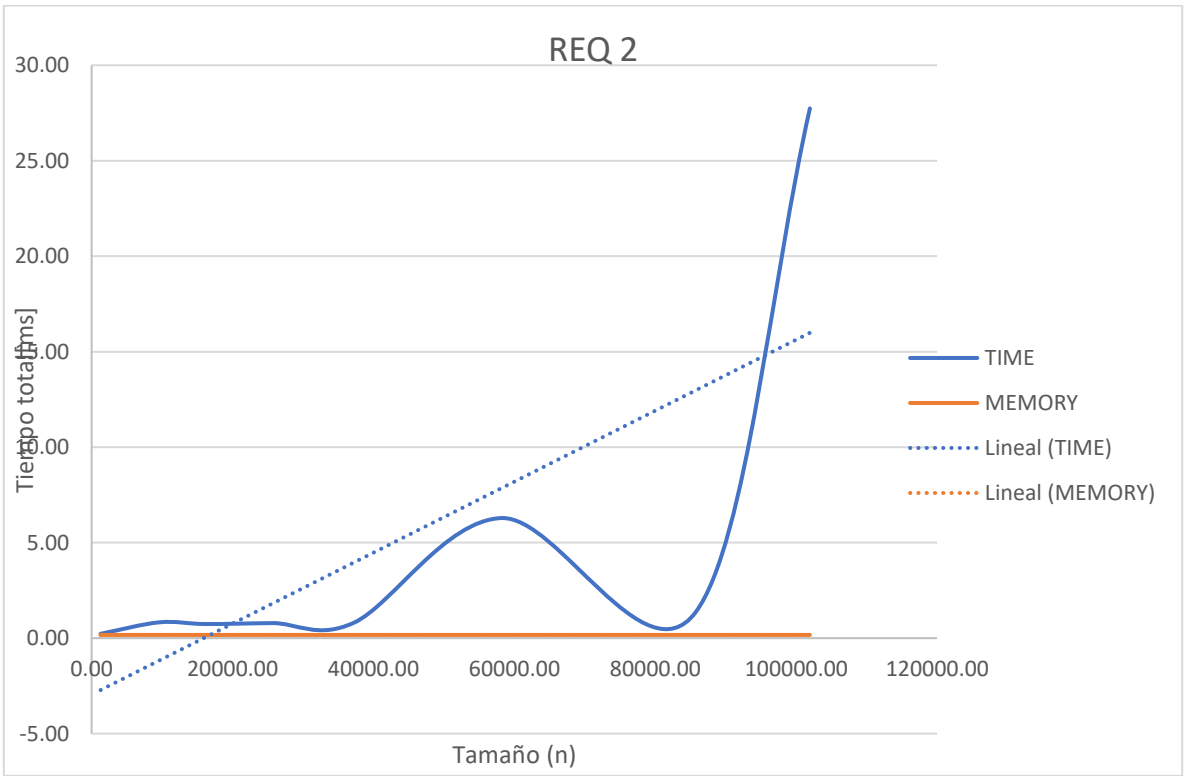
Tabla 3. Comparación de memoria para los requerimientos



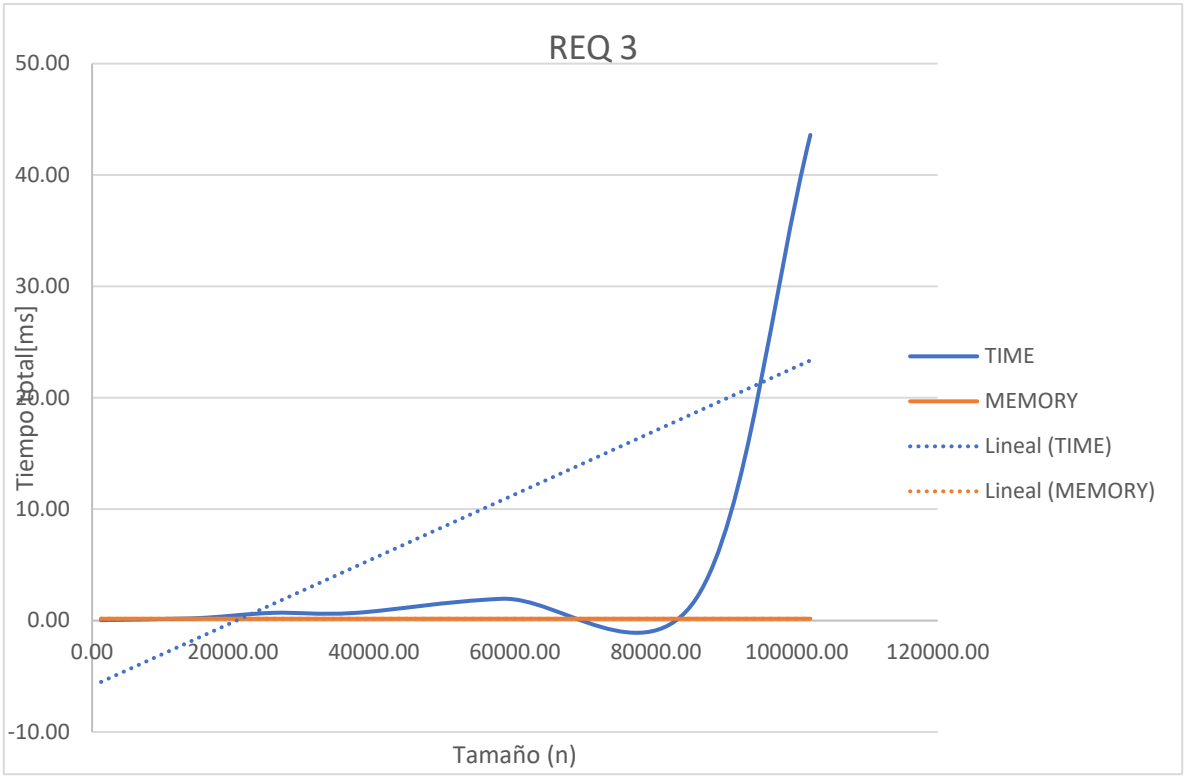
Grafica 1. Comparación de tiempos de ejecución y memoria para la carga de datos



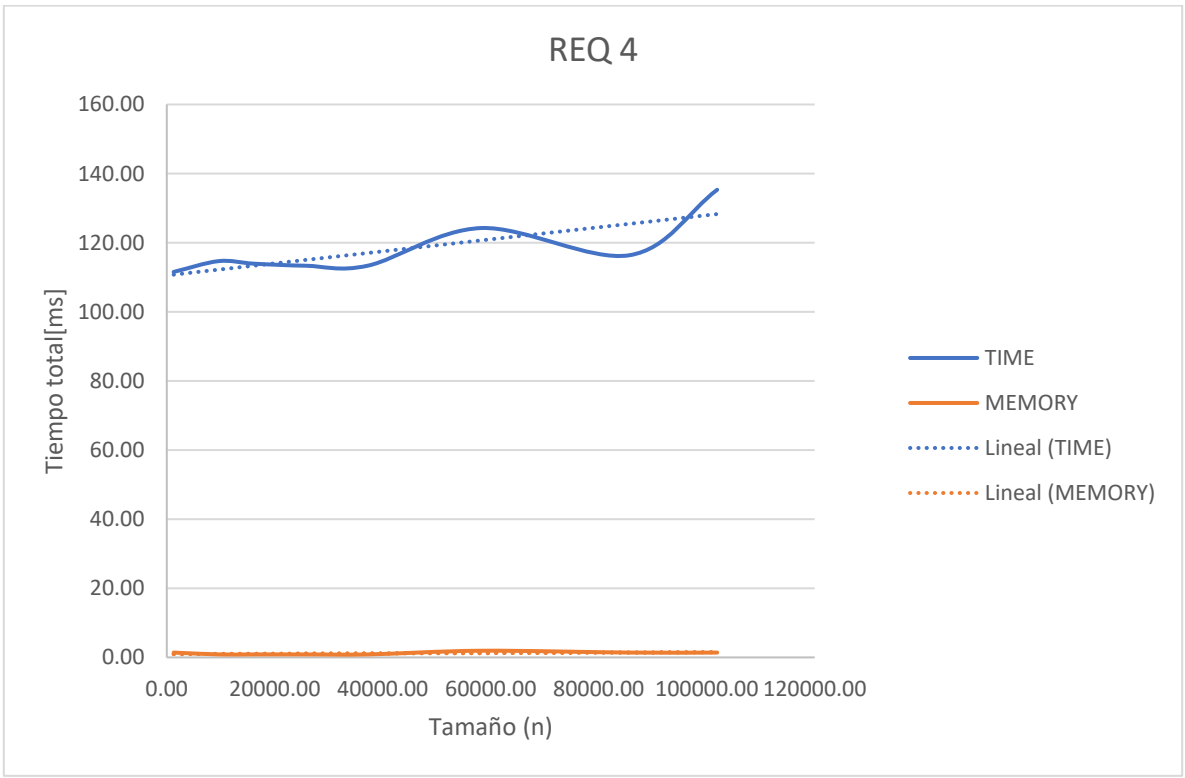
Grafica 2. Comparación de tiempos de ejecución y memoria para el requerimiento 1



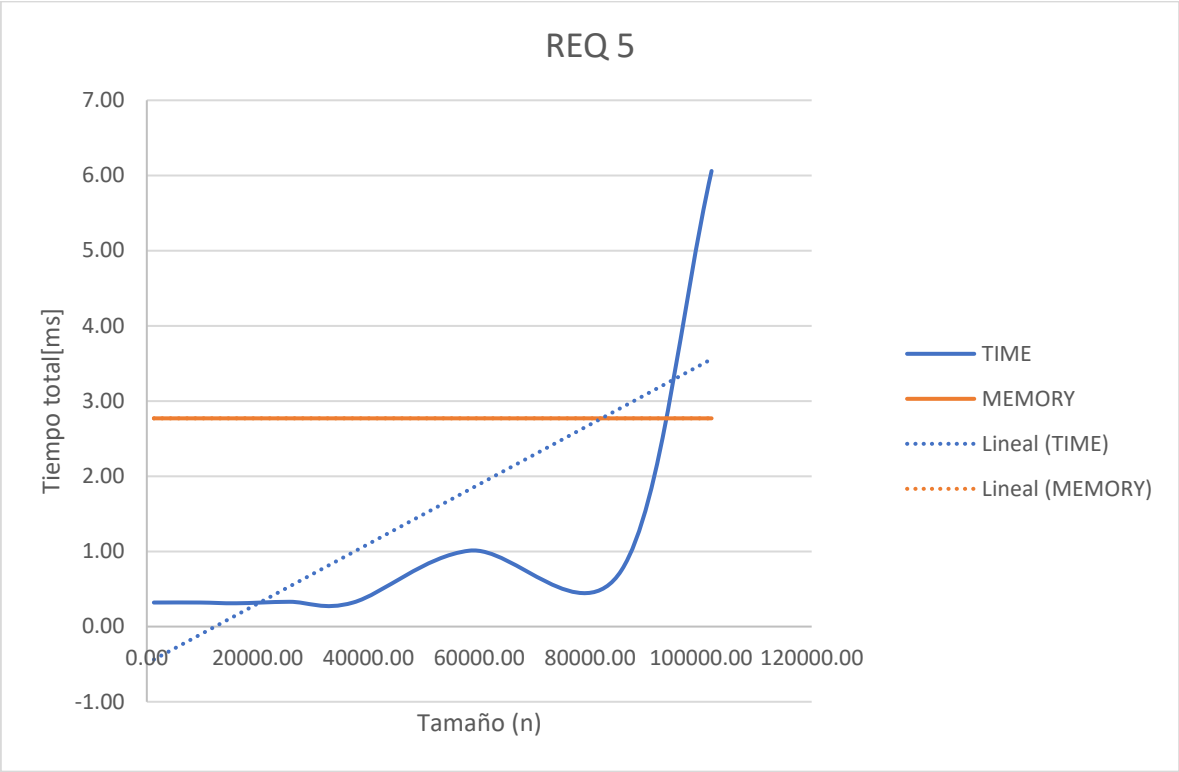
Grafica 3. Comparación de tiempos de ejecución y memoria para el requerimiento 2



Grafica 4. Comparación de tiempos de ejecución y memoria para el requerimiento 3



Grafica 5. Comparación de tiempos de ejecución y memoria para el requerimiento 4



Grafica 6. Comparación de tiempos de ejecución y memoria para el requerimiento 5

Maquina 2

Resultados

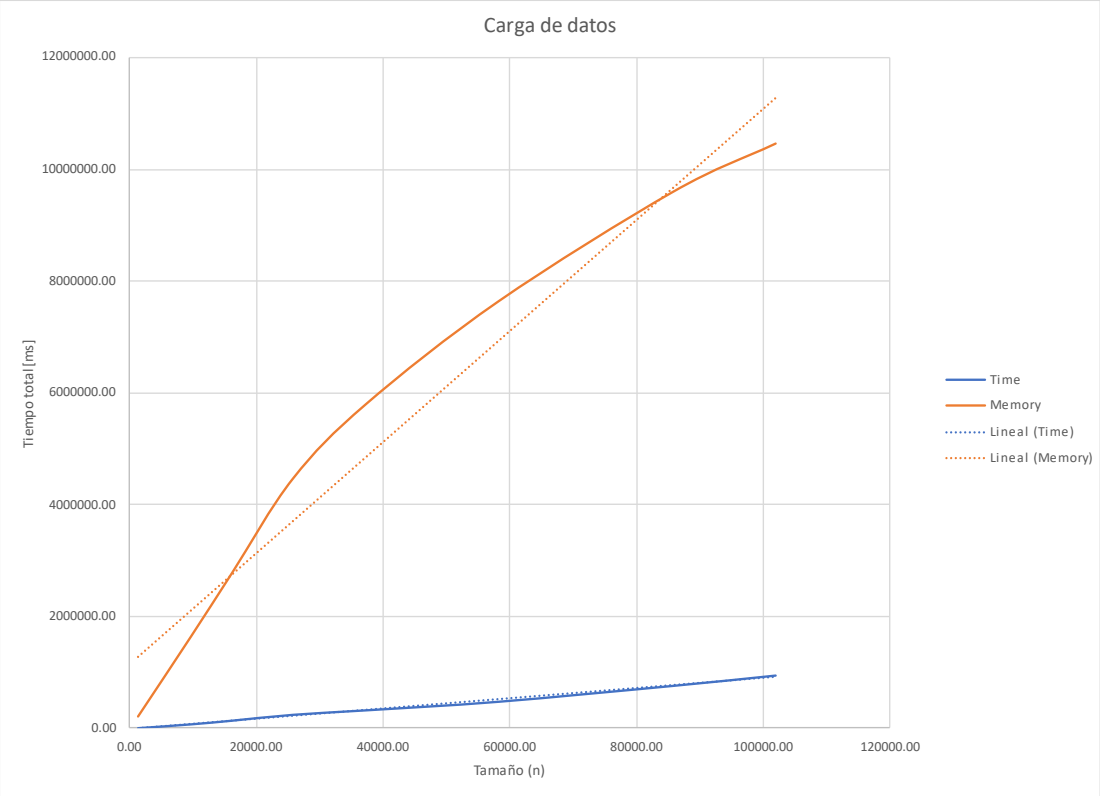
Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAY_LIST)	Carga de datos [ms]	Requerimiento 1 [ms]	Requerimiento 2 [ms]	Requerimiento 3 [ms]	Requerimiento 4[ms]	Requerimiento 5 [ms]
0.5%	1270.00	9289.43	0.43	0.16	0.09	146.39	0.38
5.0%	9625.00	77147.00	2.50	0.48	0.14	154.95	0.27
10.0%	16137.00	142820.34	4.94	0.55	0.24	149.30	0.33
20.0%	25718.00	246874.72	73.99	0.94	0.33	178.34	0.28
30.0%	37241.00	326517.31	143.55	10.65	11.85	168.90	4.07
50.0%	58216.00	479632.01	158.81	13.72	31.85	190.35	6.90
80.0%	85685.00	761425.15	282.23	16.05	55.83	197.36	8.05

100.0%	101939.00	945664.70	330.77	17.24	66.81	203.52	10.26

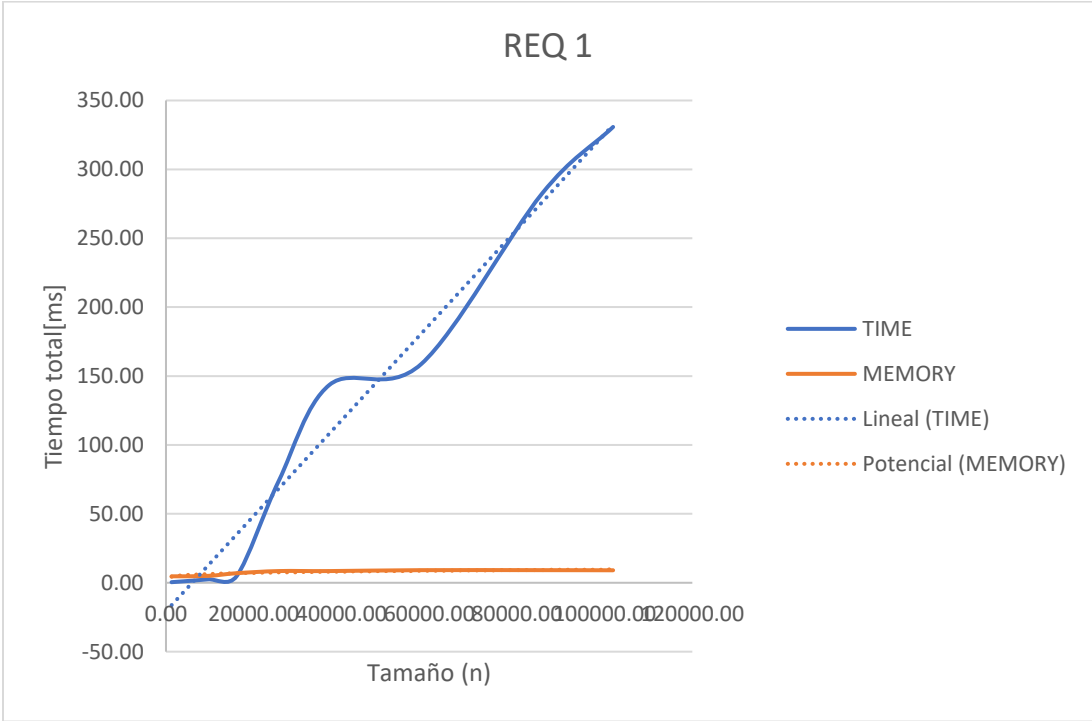
Tabla 4. Comparación de tiempos de ejecución para los requerimientos

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAY_LIST)	Carga de datos [kB]	Requerimiento 1 [kB]	Requerimiento 2 [kB]	Requerimiento 3 [kB]	Requerimiento 4[kB]	Requerimiento 5 [kB]
0.5%	1270.00	209664.30	4.68	1.67	0.59	4526.36	7.60
5.0%	9625.00	1639304.37	5.02	1.13	0.59	4526.32	4.24
10.0%	16137.00	2781302.08	6.96	0.71	1.12	4526.54	6.27
20.0%	25718.00	4462919.62	8.50	0.57	1.27	4525.93	5.80
30.0%	37241.00	5794562.87	8.50	0.17	2.20	4526.14	5.86
50.0%	58216.00	7637617.09	9.16	0.17	2.20	4525.62	5.38
80.0%	85685.00	9589849.11	9.16	0.17	2.20	4526.16	4.74
100.0%	101939.00	10456170.14	9.06	0.17	2.74	4526.32	3.29

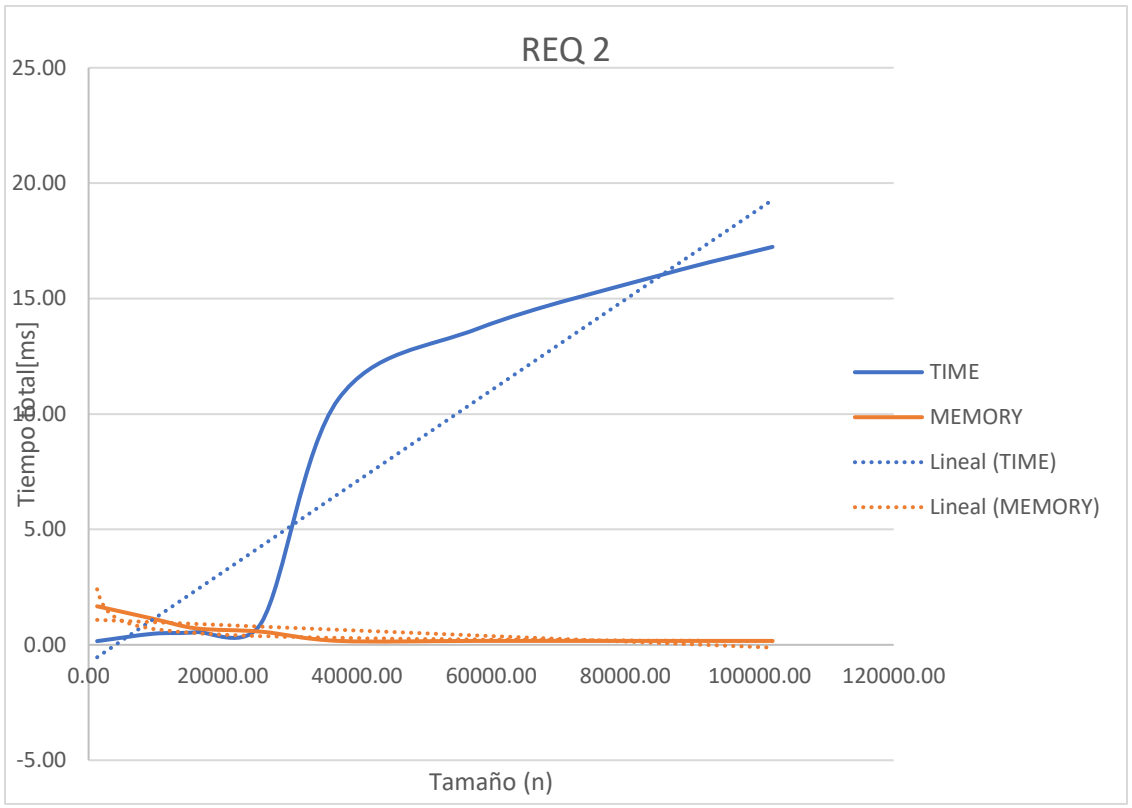
Tabla 5. Comparación de memoria para los requerimientos



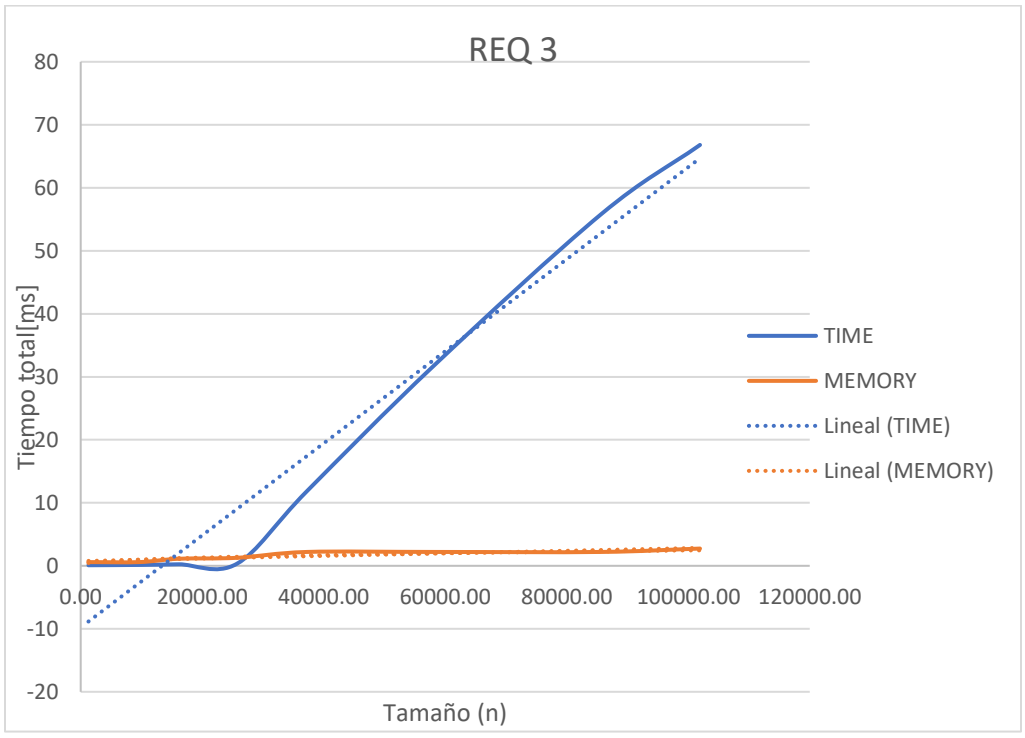
Grafica 7. Comparación de tiempos de ejecución y memoria para la carga de datos



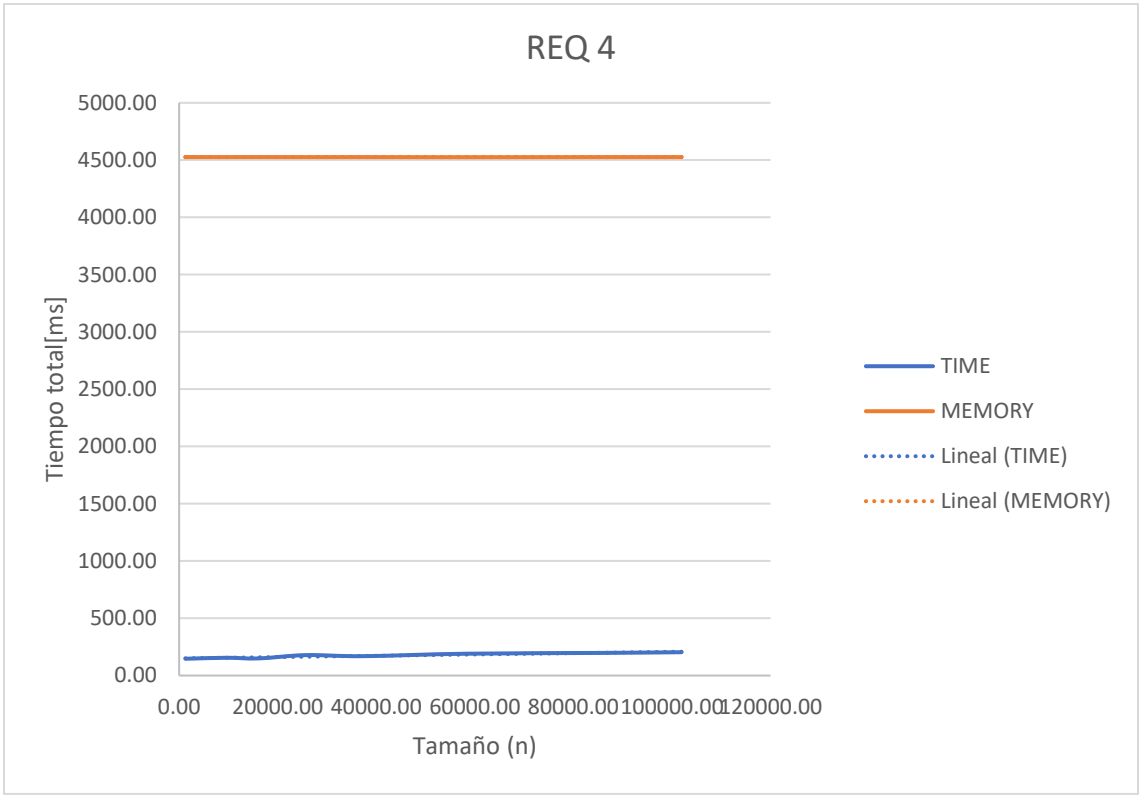
Grafica 8. Comparación de tiempos de ejecución y memoria para el requerimiento 1



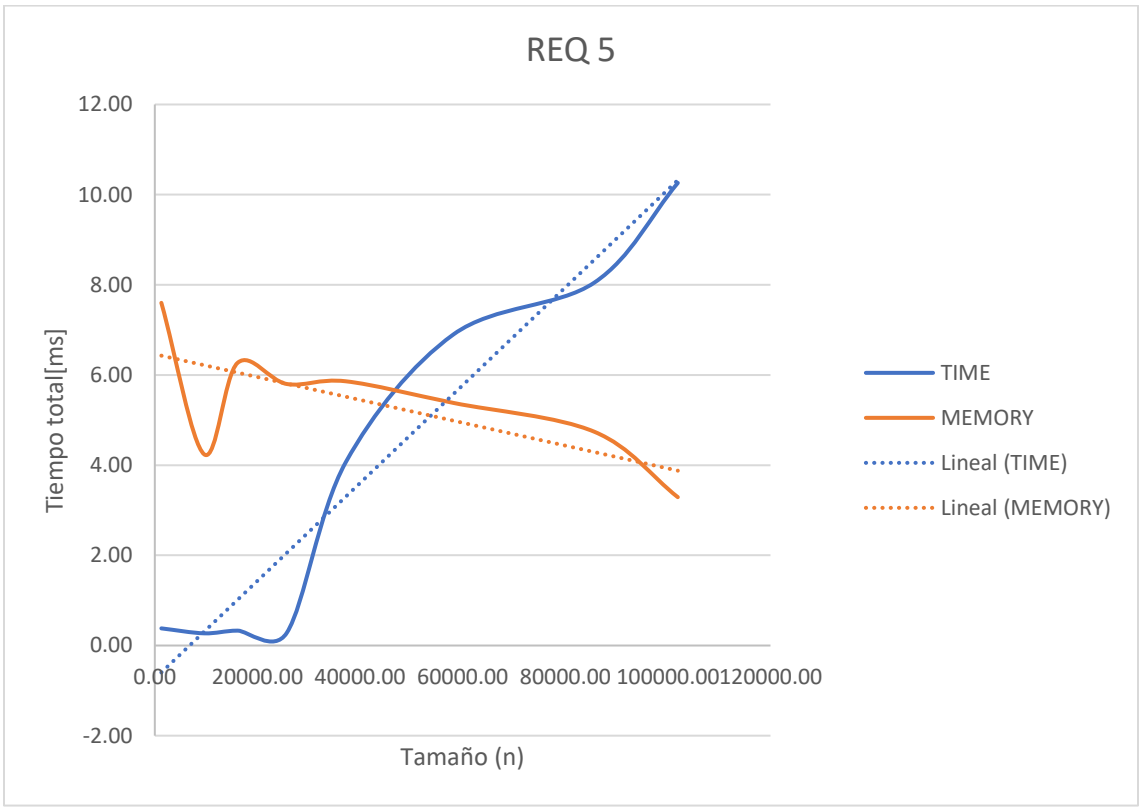
Grafica 9. Comparación de tiempos de ejecución y memoria para el requerimiento 2



Grafica 10. Comparación de tiempos de ejecución y memoria para el requerimiento 3



Grafica 11. Comparación de tiempos de ejecución y memoria para el requerimiento 4



Grafica 12. Comparación de tiempos de ejecución y memoria para el requerimiento 5

Comparación frente a Reto 1:

Como podemos apreciar, los tiempos de ejecución del reto 2 con respecto al reto 1 son mucho más rápidos y la complejidad es menor. Esto debido a que cuando cargamos los datos, lo hacemos usando las estructuras de datos de mapas, que son muy útiles y nos permiten obtener la información de manera más eficiente, sin necesidad de recorrer TODOS los datos para encontrar algo en específico. No obstante, se identifica que durante la carga de datos el tiempo es mayor debido a que las estructuras mapas deben realizar re hash y otros procedimientos para poder mantener tiempos de ejecución rápidos.