Integrantes:

- Sofia Velasquez Marin - s.velasquezm2@uniandes.edu.co -
- Valeria Caro Ramirez - v.caro@uniandes.edu.co -

Documento de Análisis

En este documento estara conformado por el análisis de complejidad de cada uno de los requerimientos del Reto 1 y a su vez, las pruebas de tiempos de ejecución de los mismos.

1. Complejidad de los Algoritmos

n = Tamaño del Catalogo de Artistas

m = Tamaño del Catalogo de Obras

Req 1.

```
def getCronologicalArtist (sortedArtist_BDate, beginDate, endDate):
    """
    Req 1
    Retorna la lista de los artistas nacidos entre beginDate y endDate
    """
    BornInRange = lt.newList('SINGLE_LINKED')
    for artista in lt.iterator(sortedArtist_BDate):
        if beginDate <= artista['BeginDate'] and endDate >= artista['BeginDate']:
            lt.addLast(BornInRange, artista)
    return BornInRange
```

El algotimo depende de n. Se recorre en un for loop la lista de los artistas organizada previamente por Begin Date, por lo que la complejidad del requerimiento es O(n)

Complejidad: O(n)

Req 2.

```
def getCronologicalArtwork (sortedArtwork_Date, beginDate, endDate):
    """
    Req 2
    Retorna la lista de las obras que fueron adquiridas entre beginDate y endDate
    """
    AcquiredInRange = lt.newList('SINGLE_LINKED')
    for artwork in lt.iterator(sortedArtwork_Date):
        if beginDate <= artwork['Date Acquired'] and endDate >= artwork['Date Acquired']:
            lt.addLast(AcquiredInRange, artwork)

    return AcquiredInRange

def getArtworksPurchased (catalog):
    """
    Req 2
    Retorna el numero de obras que fueron adquiridas por compra ('purchase')
    """
    purchased = 0
    for item in lt.iterator(catalog):
        if "purchase" in item['CreditLine'].lower():
            purchased += 1
    return purchased
```

El algotimo depende de m. En la primera funcion, se recorre en un for loop la lista de las obras organizadas previamente por fecha de adquision, por lo que la complejidad de la funcion es O(m). En la segunda funcion se recorre la lista de obras dentro del rango ingresado por el usuario (z). Por lo que la complejidad de esta funcion es O(z). Al hacer la suma de las complejidades como (z) nunca va a ser mayor que (m), se concluye que la complejidad del requerimiento es O(m).

Complejidad: O(m)

Req 3. – Implementado por Sofia Velasquez

```
def getArtistInfo(catalog, artistName):
    """
    Req 3
    Retorna una el diccionario del artista a examinar
    Si no se encuentra el artista se retorna None
    """
    artist = catalog['Artist']
    info = None
    for i in lt.iterator(artist):
        if i['DisplayName'].lower() == artistName.lower():
            info = i
            break
    return info

def getArtistsArtwork(catalog, artistID):
    """
    Req 3
    Retorna una TAD lista con todas las obras del artista
    """
    artwork = catalog['Artwork']
    ArtistsArtwork = lt.newList('ARRAY_LIST')
    for i in lt.iterator(artwork):
        if artistID in i['ConstituentID']:
            lt.addLast(ArtistsArtwork, i)
    return ArtistsArtwork

def getArtistTechnique(catalog):
    """
    Req 3
    Retorna una tupla:
        1. Diccionario con todas las tecnicas de el artista
        como llaves y la cantidad respectivas de obras como valores
        2. La tecnica mas usada
    """
    Technique = {}
    top1 = 0
    topMedium = None
    for i in lt.iterator(catalog):
        medium = i["Medium"]

        veces = Technique.get(medium,0)

        Technique[medium] = veces +1
        if Technique[medium] > top1:
            top1 = Technique[medium]
            topMedium = medium

    return Technique, topMedium
```

El algotimo depende de n y de m. En la primera funcion se recorre en un for loop la lista de los artistas, en el peor caso el artista a buscar se encuentra en la ultima posicion, por lo que la complejidad de la funcion es O(n). En la segunda funcion se recorre en un for loop la lista de obras, por lo que la complejidad de esta funcion es O(m).La tercera funcion depende del tamaño de la lista de obras del artista (a) la cual se recorre en un for loop por lo que la complejidad de esta funcion es O(a). Al hacer la suma de las complejidades como (a) nunca va a ser mayor que (m), y no se sabe si (n) es mayor que (m) o viceversa se concluye que la complejidad del requerimiento en el peor caso es O(n + m).

Complejidad: O(n + m)

Req 4. Implentado por Valeria Caro

```
def getArtworkNationality(catalog):

    """
    Req 4.
    Retorna:
        1. La lista de todas las nacionalidades, con su respectivo numero de obras
        2. Las obras de la nacionalidad que se encuentra en la primera posicion de la lista
    """
    obras = {'Unknown': []}
    artistas = {}
    for artist in lt.iterator(catalog['Artist']):
        artistas[str(artist['ConstituentID'])] = artist

    for artwork in lt.iterator(catalog['Artwork']):
        stringIDs = str(artwork['ConstituentID'])
        artistIDs = stringIDs[1:-1].replace(" ", "").split(",")
        if len(artistIDs) == 0:
            obras['Unknown'].append(artwork)

        for id in artistIDs:
            artist = artistas[id]
            nacionalidad = artist['Nationality']
            if nacionalidad == 'Nationality unknown':
                obras['Unknown'].append(artwork)
            elif nacionalidad in obras:
                obras[nacionalidad].append(artwork)
            else:
                obras[nacionalidad] = [artwork]

    sorted_list = lt.newList('ARRAY_LIST')
    for key in obras:
        lt.addLast(sorted_list, {'Longitud':len(obras[key]), 'Nacionalidad': key})

    mer.sort(sorted_list, cmpArtistbyNationality)
    lista = lt.subList(sorted_list, 1, 10)


    return lista, obras[lt.getElement(lista, 1)['Nacionalidad']]
```

El algotimo depende de m y n. Se recorre en un for loop la lista de obras por lo que la complejidad de la funcion es O(n) y luego, se realiza otro for loop por los artistas O(m). Posteriormente se organiza la lista, y por lo tanto, la complejidad es O(n + m).

Complejidad: O(n + m)

Req 5.

```
def getArworkByDepartment (catalog, department):
    """
    Req 5
    Retorna la lista de las obras que pertenecen al departamento dado
    """
    art = lt.newList('ARRAY_LIST')
    for i in lt.iterator(catalog):
        if i['Department'].lower() == department.lower():
            lt.addLast(art, i)
    return art

def getTransportationCost(catalog):
    """
    Req 5
    Se calcula el costo para transportar cada obra del catalogo dado y
    se añade a su respectivo diccionario el calculo mas costoso multiplicado
    por 72; si no hay informacion suficiente para el calculo se deduce que
    el costo es de 48 USD.

    Retorna una tupla:
        1. El catalogo actualizado .
        2. La suma estimada de los costos de transporte de las obra.
        3. La suma estimada del peso de las obras.
    """
    for i in lt.iterator(catalog):
        Weight = float(i['Weight'])
        Length = float(i['Length'])
        Width = float(i['Width'])
        Height = float(i['Height'])
        Radius = float(i['Diameter'] / 2) / 100

        m2 = (Height*Width)/10000
        m3 = (Height*Width*Length)/1000000
        m2_v2 = math.pi * (Radius**2)
        m3_v2 =  (4/3)*(math.pi)*(Radius**3)

        mayor = max(m2,m3,m2_v2,m3_v2,Weight)
        cost = 48
        if mayor != 0:
            cost = round(72*mayor, 3)

        i['TransCost'] = cost

    return catalog
```

```
def getArtworkTotal_CostWeight(catalog):
    """
    Req 5
    Retorna una tupla:
        1. La suma estimada de los costos de transporte de las obra.
        2. La suma estimada del peso de las obras.
    """
    Totalcost = 0
    TotalWeight = 0
    for i in lt.iterator(catalog):
        Totalcost += i['TransCost']
        TotalWeight += float(i['Weight'])
    return round(Totalcost,3) ,  round(TotalWeight,3)
```

El algotimo depende de m. En la primera funcion, se recorre en un for loop la lista de las obras organizadas previamente por fecha, por lo que la complejidad de la funcion es O(m). En la segunda funcion se recorre la lista de obras dentro del departamento ingresado por el usuario (d). Por lo que la complejidad de esta funcion es O(d). En el view, se hace un merge sort de la lista de obras dentro del departamento, el cual tiene una complejidad de O(d*log(d)). Al hacer la suma de las complejidades como (d) nunca va a ser mayor que (m) pero no sabe si (d*log(d)) es mayor que m, se concluye que la complejidad del requerimiento es O(m + d*log(d)).

Complejidad: O(m + d*log(d)).

Req 6.

```
def createNewDisplay(catalog,beginYear, finalYear, area):
    Artwork = catalog['Artwork']
    display = lt.newList('ARRAY_LIST')

    totalArtworks = 0
    artArea = 0
    areaUsed = 0
    for i in lt.iterator(Artwork):
        if (beginYear <= i['Date'] and finalYear >= i['Date']) and (i['Area'] <= area):
            totalArtworks +=1
            if (i['Classification'] == 'Painting' or i['Classification'] == 'Photograph' \
                or i['Classification'] == 'Print' or i['Classification'] == 'Drawing') and (i['Area'] != 0):

                artArea += i['Area']
                if area < artArea:
                    continue
                else:
                    areaUsed = artArea
                    lt.addLast(display, i)

    return display, areaUsed, totalArtworks
```

El algotimo depende de m. Se recorre en un for loop la lista de obras por lo que la complejidad de la funcion es O(m). Se concluye que la complejidad del requerimiento es O(m).

Complejidad: O(m)

2. Tiempos de Ejecución

Req 1.

% de Muestra (ARRAYLIST)	**Tiempo(ms)**	**Sort_List(ms) (Merge Sort)**
0,50%	0.0	31.25
5%	0.0	93.75
10%	0.0	140.62
20%	15.62	171.87
30%	15.62	218.75
50%	15.62	312.5
80%	15.62	325.0
100%	31.25	343.75

Req 2.

% de Muestra (ARRAYLIST)	**Tiempo(ms)**	**Sort_List(ms) (Merge Sort)**
0,50%	0.0	15.62
5%	0.0	171.87
10%	46.87	328.12
20%	62.5	718.75
30%	62.5	1171.87
50%	156.25	1953.12
80%	218.75	3203.12
100%	298.87	3859.37

Req 3.

% de Muestra (ARRAYLIST)	Tiempo(ms)
0,50%	0.0
5%	31.25
10%	31.25
20%	46.87
30%	78.12
50%	93.75
80%	171.87
100%	203.12

Req 4.

% de Muestra (ARRAYLIST)	Tiempo(ms)
0,50%	15.62
5%	15.62
10%	31.25
20%	93.75
30%	109.375
50%	171.875
80%	328.12
100%	328.12

Req 5.

% de Muestra (ARRAYLIST)	Tiempo(ms)	Sort_List(ms) (Merge Sort)
0,50%	31.25	15.62
5%	93.75	156.25
10%	187.5	375.0
20%	390.62	671.87
30%	593.75	1140.62
50%	1062.5	1906.25
80%	1781.25	3046.87
100%	2234.37	3656.25

Req 6.

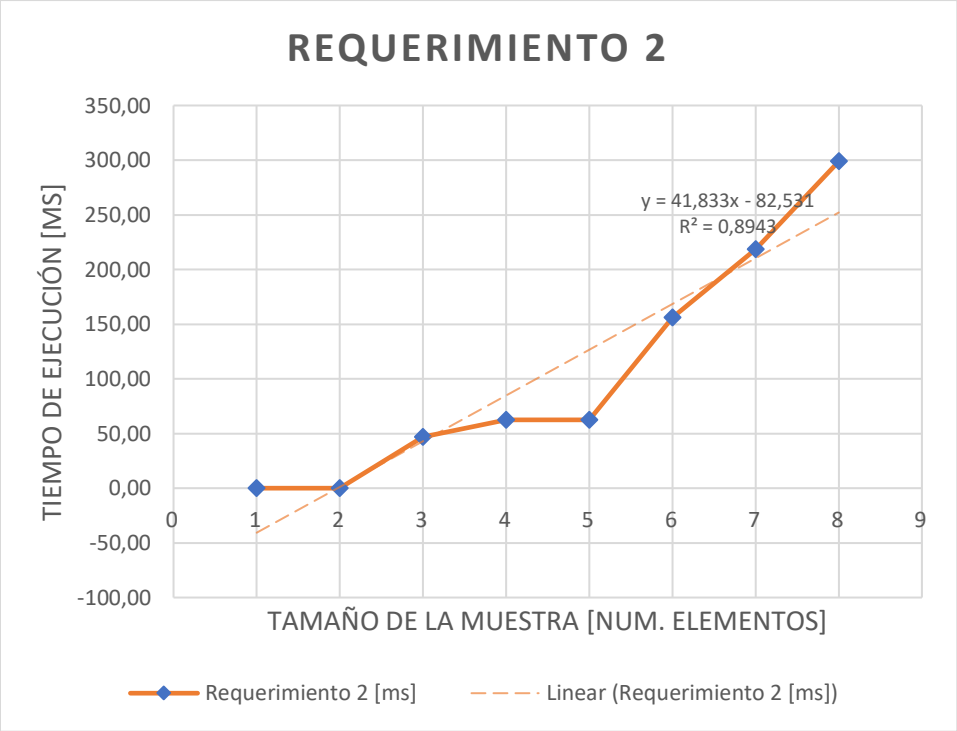
% de Muestra (ARRAYLIST)	Tiempo(ms)
0,50%	31.25
5%	31.25
10%	31.25
20%	78.12
30%	78.12
50%	140.62
80%	203.12
100%	250.0

A continuacion se presenta el Anexo de las graficas de cada unos de los requerimientos.

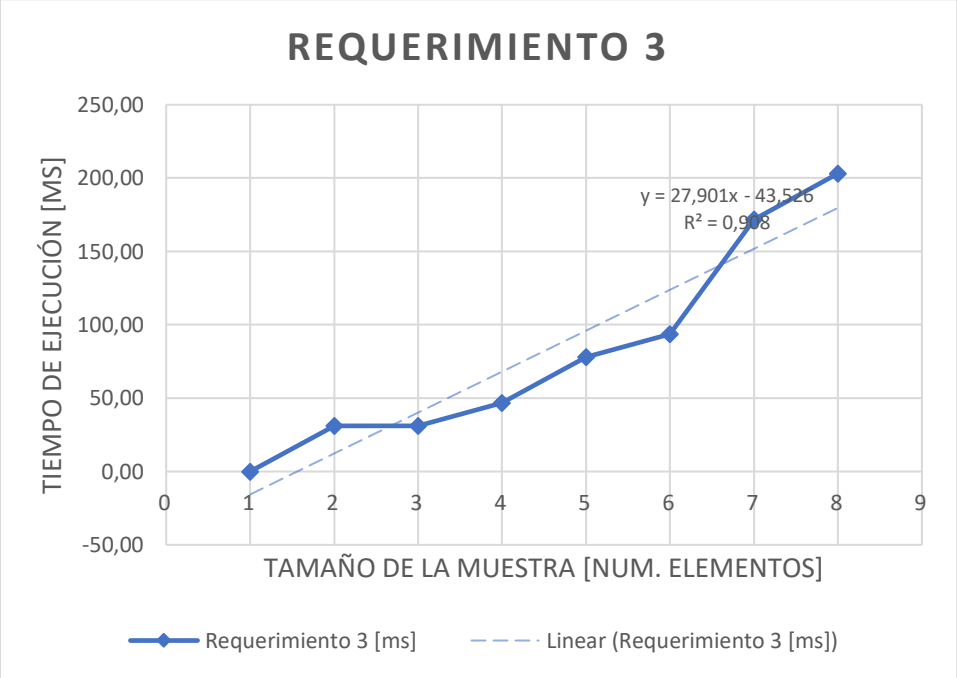
Anexo de Graficas:

Req 1.

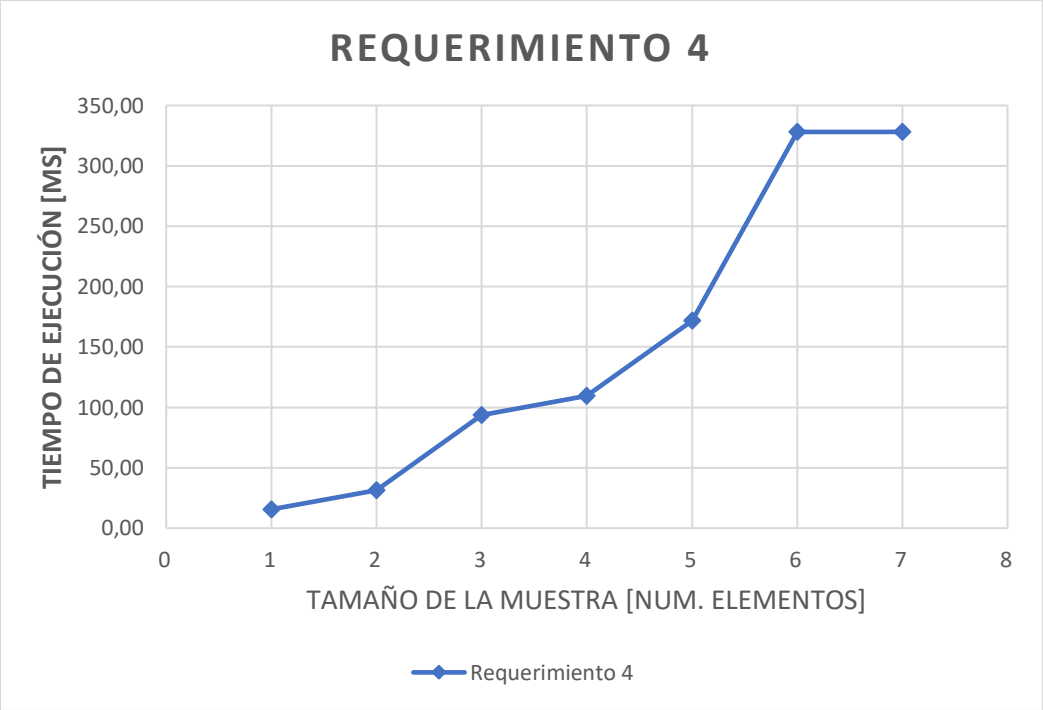
Req 2.


REQUERIMIENTO 2
350,00
300,00
250,00
200,00
150,00
100,00
50,00
0,00
-50,00
-100,00
TIEMPO DE EJECUCIÓN [MS]
0
1
2
3
4
5
6
7
8
9
y = 41,833x - 82,531
R² = 0,8943
TAMAÑO DE LA MUESTRA [NUM. ELEMENTOS]
Requerimiento 2 [ms]
Linear (Requerimiento 2 [ms])

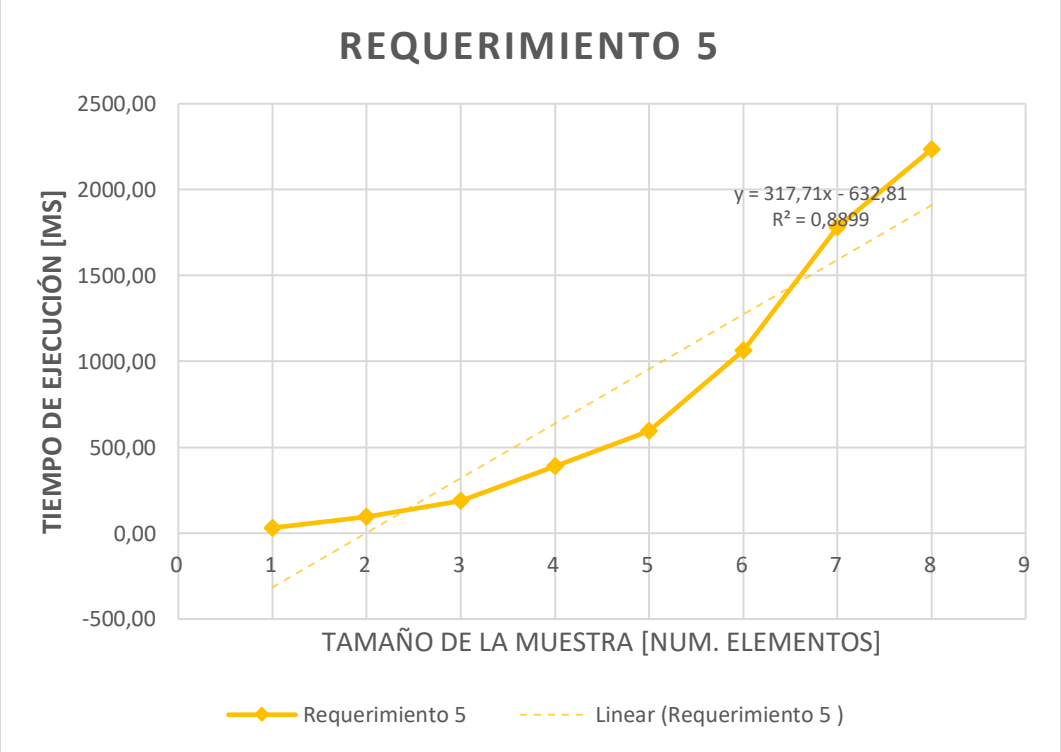
Req 3.


REQUERIMIENTO 3
250,00
200,00
150,00
100,00
50,00
0,00
-50,00
TIEMPO DE EJECUCIÓN [MS]
0
1
2
3
4
5
6
7
8
9
y = 27,901x - 43,526
R² = 0,908
TAMAÑO DE LA MUESTRA [NUM. ELEMENTOS]
Requerimiento 3 [ms]
Linear (Requerimiento 3 [ms])

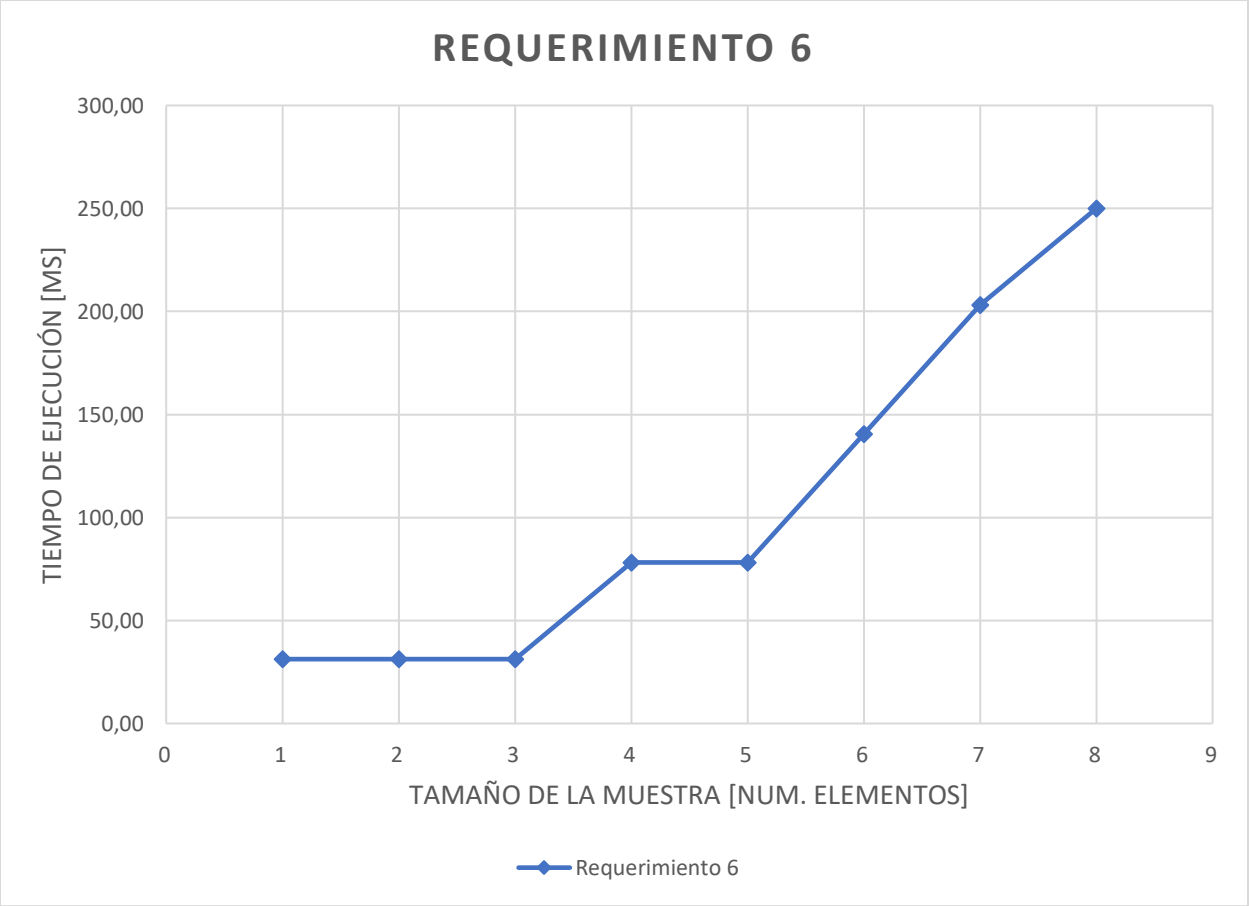
Req 4.


REQUERIMIENTO 4
TIEMPO DE EJECUCIÓN [MS]
350,00
300,00
250,00
200,00
150,00
100,00
50,00
0,00
0
1
2
3
4
5
6
7
8
TAMAÑO DE LA MUESTRA [NUM. ELEMENTOS]
Requerimiento 4

Req 5.


REQUERIMIENTO 5
TIEMPO DE EJECUCIÓN [MS]
2500,00
2000,00
1500,00
1000,00
500,00
0,00
-500,00
0
1
2
3
4
5
6
7
8
9
y = 317,71x - 632,81
R² = 0,8999
TAMAÑO DE LA MUESTRA [NUM. ELEMENTOS]
Requerimiento 5
Linear (Requerimiento 5)

Req 6.


REQUERIMIENTO 6
300,00
250,00
200,00
150,00
100,00
50,00
0,00
TIEMPO DE EJECUCIÓN [MS]
0
1
2
3
4
5
6
7
8
9
TAMAÑO DE LA MUESTRA [NUM. ELEMENTOS]
Requerimiento 6

Grafica de Comparacion de todos los requerimientos: