

### Observaciones:

- Utilizamos la librería de prettytable
- Selection.sortEdit es un ordenamiento con selection, pero con unas modificaciones para el reto. Lo que hace este algoritmo editado es ordenar las posiciones finales e iniciales que le digamos por parámetro. Como solamente necesitamos mostrar los 3 primeros y últimos artistas/obras en la mayoría de los requerimientos, solo queremos que estas posiciones quedan ordenadas. Por lo cual, al hacer este ordenamiento con selection significa que la lista se recorrerá 6 veces, es decir que la complejidad será  $O(6N) = O(N)$ . La finalidad de esta modificación era reducir la complejidad de todos los requerimientos del reto
- Las Pruebas de tiempos de ejecución del requerimiento 1,4 y 5, fueron elaboradas en:

Máquina 1	
Procesadores	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
Memoria RAM (GB)	8 GB
Nombre del SO	Windows 10 64-bits

En las pruebas de los requerimientos 2, 3, y 6 se utilizó la segunda maquina:

Máquina 2	
Procesadores	Intel(R) Core(TM) i7-1185G7 CPU @ 3GHz
Memoria RAM (GB)	16 GB
Nombre del SO	Windows 10 64-bits

- Las Pruebas de tiempos de ejecución no tienen en cuenta las cargas de los mapas, dado que estos son cargados al inicializar el catálogo. Sin embargo el tiempo promedio de la carga de datos en large utilizando la máquina 2 es de 6.35s.
- Los mapas que hicimos para el reto fueron:

Nombre Mapa	Tipo de mapa	N elementos	Factor de carga	Utilizado en el reqs.:	Key	Value
artists	Chaining	15000	4.0	Todos	Artists	Toda la información del artista
Artists_BeginDate	Chaining	331	4.0	Req 1	Años de nacimiento	Lista con los ConstituentID artistas nacidos en ese año
artworks_index_by_year	Chaining	1000	4.0	Req 2	Fechas de adquisición	Llave con una lista con los índices de los object id de las obras con esa fecha.
artists_index_name	Chaining	15000	4.0	Req 3 y 6	Nombre de un artista	ConstituentID del artista
Nationalities	Chaining	250	4.0	Req 4	Nacionalidad	Lista de obras con esa nacionalidad es. <sup>1</sup>
Department	Chaining	13	2.0	Req 5	Departamentos del museo	Lista de index de las obras del departamento

---

<sup>1</sup> Cada una de las nacionalidades tiene un diccionario con 4 llaves: "Nationality", "Artworks", "Total\_obras", "ObrasUnicas". Donde artworks contiene una lista con todos los index de las posiciones de las obras de esa nacionalidad

## Requerimiento 1

### Pruebas de tiempos de ejecución:

El rango de fechas para esta prueba fueron 1000-2000



Figura 1. Pruebas de tiempos de ejecución req1. Computador con 8gb de Ram

### Análisis de complejidad:

```
def listarArtistasCronologicamente(catalog, fechaInicial, fechaFinal):
    listaNac=lt.newList("ARRAY_LIST") #Se crea una nueva lista
    nacimientoKeys=mp.keySet(catalog["Artists_BeginDate"]) #Todos los keys del mapa de años de
nacimiento
    contador=0
    for fechaStr in lt.iterator(nacimientoKeys):
        fecha=int(fechaStr)
        if fecha>=fechaInicial and fecha<=fechaFinal:
            lt.addLast(listaNac, fecha)
            cantidadArtistas=mp.get(catalog["Artists_BeginDate"], fechaStr)["value"]["Artistas"]["siz
e"]

            contador+=cantidadArtistas
    selection.sortEdit(listaNac, cmpArtistDate, 3, ordenarInicio=True, ordenarFinal=True)
    respuestaLista=None
    respuestaLista=listasRespuesta(listaNac, catalog, "Artists", "req1")
    return listaNac, contador, respuestaLista
```

Figura 2. Código requerimiento 1

- La operación de `mp.keySet` tiene un tiempo  $O(N)$  debido a que recorre todo un map para obtener sus keys
- Clasificar en un rango de fechas tiene un tiempo de  $O(N)$ , dado que recorrerá una lista con las keys de nacimiento, y para cada una de esas keys hará operaciones que tienen tiempo  $O(1)$
- `Selection.sortEdit`<sup>2</sup>  $=O(N)$
- `listaRespuesta` tiene un tiempo de  $O(6)=O(K)$ . Creará una lista en donde se guardarán 6 elementos, cada uno de ellos representa a un artista. Por lo tanto, extraerá de las fechas de nacimiento los 3 primeros y últimos artistas.

Por lo tanto, la complejidad que tiene este algoritmo es:

$$O(N) + O(N) + O(N) + O(K)$$

$$O(N + N + N + K)$$

$$O(3N + K)$$

$$O(N)$$

## Requerimiento 2

### Pruebas de tiempos de ejecución:

Se utilizó el rango de fechas del ejemplo, 6 de junio de 1944 a 9 de noviembre de 1989.



Figura 3. Pruebas de tiempos de ejecución req1. Computador con 8gb de Ram

<sup>2</sup> Ver observaciones en primera pag

## Análisis de complejidad:

```
def listarAdquisicionesCronologicamente(catalog, fechaInicial, fechaFinal): # Requerimiento Grupal 2: Función Principal
    """
    contadorPurchase=0
    numeroArtistas=0

    yearInitial=int(fechaInicial.split("-")[0])
    yearFinal=int(fechaFinal.split("-")[0])
    year=yearInitial

    yaTengo3Iniciales=False
    yaTengo3Finales=False
    numeroAdquisiciones=0

    inicial=time.strptime(fechaInicial,"%Y-%m-%d")
    final=time.strptime(fechaFinal,"%Y-%m-%d")

    lista3Inicio=lt.newList('ARRAY_LIST')
    lista3Final=lt.newList('ARRAY_LIST')

    while year <= yearFinal:
        if mp.contains(catalog["artworks_index_by_year"],year):
            listaArtworkYearIni=mp.get(catalog["artworks_index_by_year"],year)["value"]
            for index_artwork in lt.iterator(listaArtworkYearIni):
                artw=lt.getElement(catalog["artworks"],index_artwork)
                fecha_obra=time.strptime(artw["DateAcquired"],"%Y-%m-%d")
                if inicial<=fecha_obra and final>=fecha_obra:
                    numeroAdquisiciones+=1
                    res=nombresArtistas(catalog,artw["ConstituentID"])
                    artw["ArtistsNames"]=res[0]
                    numeroArtistas+=res[1]
                    if "purchase" in artw["Creditline"].lower():
                        contadorPurchase+=1
                    if not yaTengo3Iniciales:
                        lt.addLast(lista3Inicio,artw)
            if not yaTengo3Iniciales:
                if lt.size(lista3Inicio)>3:
                    yaTengo3Iniciales=True
            year+=1

    yearContrario=yearFinal
    while (yearContrario >= yearInitial) and not yaTengo3Finales:
        if mp.contains(catalog["artworks_index_by_year"],yearContrario):
            listaArtworkYearIni=mp.get(catalog["artworks_index_by_year"],yearContrario)["value"]
            for index_artwork in lt.iterator(listaArtworkYearIni):
                artw=lt.getElement(catalog["artworks"],index_artwork)
                fecha_obra=time.strptime(artw["DateAcquired"],"%Y-%m-%d")
                if inicial<=fecha_obra and final>=fecha_obra:
                    if not yaTengo3Finales:
                        artw["ArtistsNames"]=nombresArtistas(catalog,artw["ConstituentID"])[0]
                        lt.addLast(lista3Final,artw)
            if not yaTengo3Finales:
                if lt.size(lista3Final)>3:
                    yaTengo3Finales=True
            yearContrario-=1

    lista3Final=sortList(lista3Final,cmpArtworkByDateAcquired)
    lista3Inicio=sortList(lista3Inicio,cmpArtworkByDateAcquired)

    for elementoFinal in lt.iterator(lista3Final):
        lt.addLast(lista3Inicio,elementoFinal)

    return lista3Inicio, contadorPurchase, numeroAdquisiciones, numeroArtistas
```

Figura 4. Código requerimiento 2

- Para reducir la complejidad se accede a un mapa en dónde las llaves son los años y los valores son listas en dónde están los índices (de la lista de artworks) de las obras de arte. Asegurando que la complejidad espacial no se exceda. Al recorrer las obras únicamente por año accedido la complejidad promedio se reduce. Sin embargo, el peor caso, cuando se selecciona todo el rango de fechas, será  $O(N)$ .
- Luego se hace un recorrido en sentido contrario, esto permite seleccionar un conjunto de obras candidatas a ser las primeras tres y un conjunto final de obras candidatas a

ser las primeras tres. Este recorrido usualmente no recorre más de 3 o 4 años por lo que no aumenta sustancialmente la complejidad temporal. Ahora bien, si tomamos el peor caso también sería  $O(N)$  en el caso de tener en un solo año todas las obras.

- Los ordenamientos tienen complejidad  $O(N)$  pues solo se hacen ordenamientos de grupos entre 1-10 obras, en el peor caso existe al seleccionar las obras la peor está al final.
- Las instrucciones que no están ligadas a ningún ciclo tienen complejidad  $O(K)$

Por lo tanto, la complejidad que tiene este algoritmo para su peor caso es:

$$O(N) + O(N) + O(N) + O(K)$$

$$O(N + N + N + K)$$

$$O(3N + K)$$

$$O(N)$$

Su mejor caso también será  $O(N)$

### Comparación con el reto 1:



Al compararlo con el primer reto se nota una mejora en los tiempos de ejecución de todas las muestras. Para el primer requerimiento se compara con Selection. Como se evidencia la combinación de un algoritmo eficiente y la utilización de mapas permiten unos resultados sorprendentes.

### Requerimiento 3 – Hecho por Daniel

#### Pruebas de tiempos de ejecución:

Se utilizó el artista de ejemplo para hacer las pruebas (Louise Bourgeois)



Figura 5. Pruebas de tiempos de ejecución req1. Computador con 8gb de Ram

## Análisis de complejidad:

```
def tecnicasObrasPorArtista(catalog,nombre): # Requerimiento Individual 3: Función Principal
    """
    Clasifica las obras de un artista por técnica dado un nombre
    Parámetros:
        catalog: estructura de datos con el catalogo de artistas y obras
        nombre: nombre del artista
        sortType: tipo de ordenamiento a utilizar
    Retorno:
        sortedList: lista de técnicas en donde cada elemento es una lista de obras de cada técnica
        totalObras: número total de obras del artista
    """
    constituentID = mp.get(catalog['artists_index_name'],nombre)["value"] # obtiene el constituentID
    del mapa
    indicesObras = mp.get(catalog['artists'], constituentID)["value"]["artwork_index_list"] # ob-
    tiene una lista con los indices de las obras
    obras=lt.newList()
    tecnicas=lt.newList()
    for indiceObra in lt.iterator(indicesObras): # obtiene las obras a partir de las lista de indi-
    ces
        lt.addLast(obras,lt.getElement(catalog["artworks"],indiceObra))
    for obraArtista in lt.iterator(obras): # se van añadiendo cada obra a una lista con la obras de
    cada tecnica alojada a su vez en una lista de tecnicas
        encontro=False
        for tecnica in lt.iterator(tecnicas): # busca si la tecnica existe en la lista de tecni-
    cas
            if obraArtista["Medium"] == lt.getElement(tecnica,0)["Medium"]:
                lt.addLast(tecnica,obraArtista) # si existe la añade la obra a la técnica
                encontro=True
            if not encontro: # si no existe
                lt.addLast(tecnicas,lt.newList()) # crea una técnica (lista de obras) en la lista de
    tecnicas
                lt.addLast(lt.lastElement(tecnicas),obraArtista) # añade la lista de obras de esa
    tecnica
        # sortedList=sortList(tecnicas,cmpFunctionTecnicasArtista,sortType) # utiliza la función de
    comparación con orden ascendente
        totalObras=lt.size(obras) # retorna el número total de obras
        tecnicas=sortList(tecnicas,cmpFunctionTecnicasArtista)
        return tecnicas,totalObras,lt.getElement(lt.getElement(tecnicas,0),0)["Medium"]
```

Figura 6. Código requerimiento 2

- El peor caso es cuando el artista tiene todas las obras  $O(N)$ , dado que acceder al artista es  $O(1)$  dado que hay un mapa relacionando nombres con lista de artistas.
- Organizar los medios como se hace por selección será  $O(N)$  en caso de tener mismos medios que obras
- Las instrucciones que no están ligadas a ningún ciclo tienen complejidad  $O(K)$

Por lo tanto, la complejidad que tiene este algoritmo para su peor caso es:

$$O(N) + O(N) + O(K)$$

$$O(N)$$

Su mejor caso también será  $O(N)$



### Comparación con el reto 1:



Al compararlo con el primer reto se nota una mejora en los tiempos de ejecución de todas las muestras. Para el primer requerimiento se compara con Selection. Como se evidencia la combinación de un algoritmo eficiente y la utilización de mapas permiten unos resultados sorprendentes.

### Requerimiento 4 – Hecho por Jenifer:

#### Pruebas de tiempos de ejecución:



Figura 7. Pruebas de tiempos de ejecución req4. Computador con 8gb de Ram

## Análisis de complejidad:

Código	Complejidad
<pre>nationalitiesQ=lt.newList("ARRAY_LIST") #Se crea una nueva lista nationalityKeys=mp.keySet(catalog["nationalities"]) #Todos los keys del mapa de nacionalidades for nationality in lt.iterator(nationalityKeys):     infoNationality=mp.get(catalog["nationalities"],nationality)["value"]     infoAdd={"Nacionalidad":nationality,             "Total_obras":infoNationality["Total_obras"]}     lt.addLast(nationalitiesQ,infoAdd)</pre>	<p>Se recorre todo el map para obtener los keys ( <math>O(N)</math>), seguido a eso se recorre la lista de keys() (<math>O(N)</math>) para guardar la información de cada nacionalidad en una nueva lista.</p> <p><math>O(N+N) = O(2N) = O(N)</math></p>
<pre>selection.sortEdit(nationalitiesQ,cmpNationalitiesSize,10,                     ordenarInicio=True,ordenarFinal=False) keyPrimerlugar=lt.getElement(nationalitiesQ,1)["Nacionalidad"] top10=lt.subList(nationalitiesQ,1,10)</pre>	<p>1. Ordenamiento editado de Sort <math>O(10N)=O(N)</math></p> <p>2. sublist() <math>O(10) = O(K)</math></p> <p><math>O(N+K)=O(N)</math></p>

```

#Respuesta con 6 obras
    listaObrasPrimerL=mp.get(catalog["nationalities"]
,keyPrimerlugar)["value"]
    obrasUnicas=listaObrasPrimerL["Artworks"]
    sizeObrasUnicas=listaObrasPrimerL["ObrasUnicas"]
    rtaNElementos=lt.newList("ARRAY_LIST")
    i=1
    n=0
    recorrer=True
    while recorrer:
        elemento=lt.getElement(obrasUnicas,i)
        obra=lt.getElement(catalog["artworks"],elemento)
        obra["NombresArtistas"]=nombresArtistas(catalog,obra["ConstituentID"])
        lt.addLast(rtaNElementos,obra)
        n+=1
        if n>6 or n>sizeObrasUnicas:
            recorrer=False
        if n==3:
            i=sizeObrasUnicas
        elif n>3:
            i-=1
        else:
            i+=1

```

- Distintas operaciones con tiempo  $O(K)$
- Ciclo while: Se hace el recorrido solamente 6 veces.
- Debido a que se utilizaron índices de las posiciones de cada obra de arte de la lista de artworks el tiempo es  $O(1)$  al momento de acceder a la información de cada obra

Es decir el tiempo es  $O(K)$

Por ende, la complejidad de este algoritmo es:

$$O(N) + O(N) + O(K)$$

$$O(2N+K)$$

$$O(N)$$

Aclaración:

- En la respuesta del view se imprimieron los 3 primeros y últimas obras. En las respuestas de las diapositivas las posiciones que muestran son [3,4,5,size-3,size-2,size-1], seguí solamente la instrucción de las diapositivas, no como el orden de la respuesta del view.

```

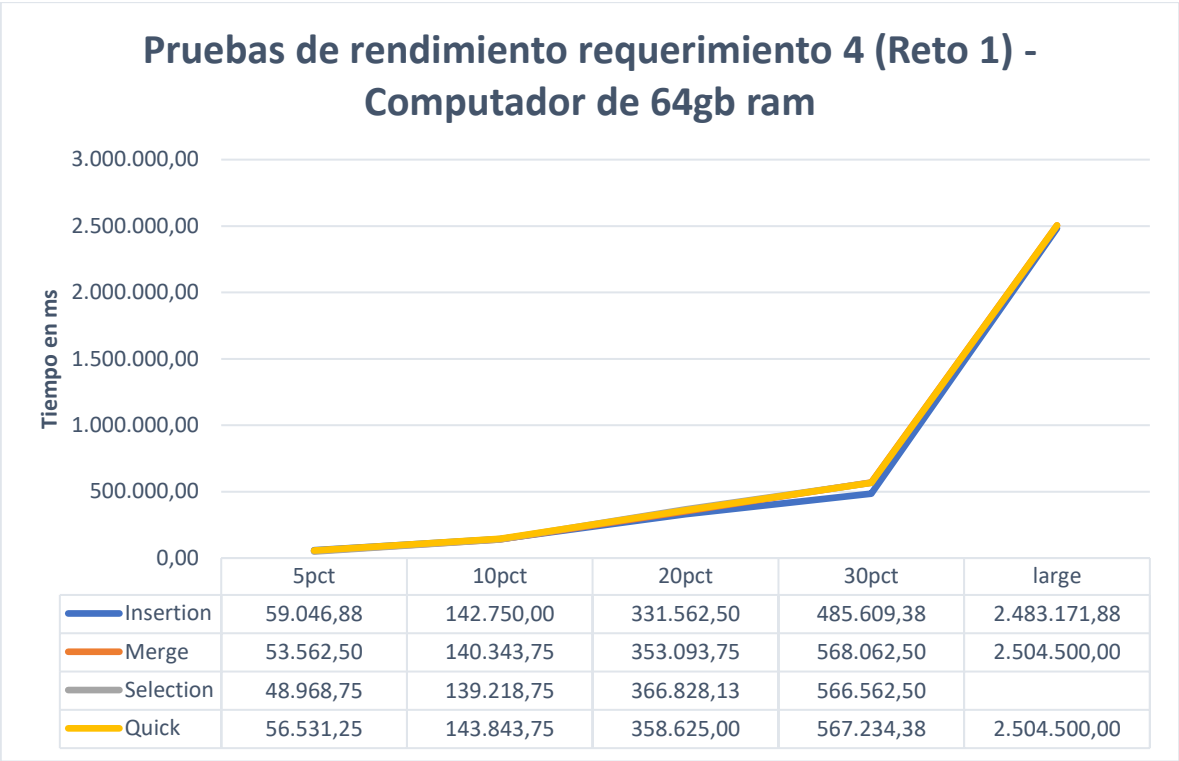
346 def add(lista, elemento):
347     lista.append(elemento)
348
349 def main():
350     lista = []
351     add(lista, 1)
352     add(lista, 2)
353     add(lista, 3)
354     add(lista, 4)
355     add(lista, 5)
356     add(lista, 6)
357     add(lista, 7)
358     add(lista, 8)
359     add(lista, 9)
360     add(lista, 10)
361     add(lista, 11)
362     add(lista, 12)
363     add(lista, 13)
364     add(lista, 14)
365     add(lista, 15)
366     add(lista, 16)
367     add(lista, 17)
368     add(lista, 18)
369     add(lista, 19)
370     add(lista, 20)
371     add(lista, 21)
372     add(lista, 22)
373     add(lista, 23)
374     add(lista, 24)
375     add(lista, 25)
376     add(lista, 26)
377     add(lista, 27)
378     add(lista, 28)
379     add(lista, 29)
380     add(lista, 30)
381     add(lista, 31)
382     add(lista, 32)
383     add(lista, 33)
384     add(lista, 34)
385     add(lista, 35)
386     add(lista, 36)
387     add(lista, 37)
388     add(lista, 38)
389     add(lista, 39)
390     add(lista, 40)
391     add(lista, 41)
392     add(lista, 42)
393     add(lista, 43)
394     add(lista, 44)
395     add(lista, 45)
396     add(lista, 46)
397     add(lista, 47)
398     add(lista, 48)
399     add(lista, 49)
400     add(lista, 50)
401     add(lista, 51)
402     add(lista, 52)
403     add(lista, 53)
404     add(lista, 54)
405     add(lista, 55)
406     add(lista, 56)
407     add(lista, 57)
408     add(lista, 58)
409     add(lista, 59)
410     add(lista, 60)
411     add(lista, 61)
412     add(lista, 62)
413     add(lista, 63)
414     add(lista, 64)
415     add(lista, 65)
416     add(lista, 66)
417     add(lista, 67)
418     add(lista, 68)
419     add(lista, 69)
420     add(lista, 70)
421     add(lista, 71)
422     add(lista, 72)
423     add(lista, 73)
424     add(lista, 74)
425     add(lista, 75)
426     add(lista, 76)
427     add(lista, 77)
428     add(lista, 78)
429     add(lista, 79)
430     add(lista, 80)
431     add(lista, 81)
432     add(lista, 82)
433     add(lista, 83)
434     add(lista, 84)
435     add(lista, 85)
436     add(lista, 86)
437     add(lista, 87)
438     add(lista, 88)
439     add(lista, 89)
440     add(lista, 90)
441     add(lista, 91)
442     add(lista, 92)
443     add(lista, 93)
444     add(lista, 94)
445     add(lista, 95)
446     add(lista, 96)
447     add(lista, 97)
448     add(lista, 98)
449     add(lista, 99)
450     add(lista, 100)
451     add(lista, 101)
452     add(lista, 102)
453     add(lista, 103)
454     add(lista, 104)
455     add(lista, 105)
456     add(lista, 106)
457     add(lista, 107)
458     add(lista, 108)
459     add(lista, 109)
460     add(lista, 110)
461     add(lista, 111)
462     add(lista, 112)
463     add(lista, 113)
464     add(lista, 114)
465     add(lista, 115)
466     add(lista, 116)
467     add(lista, 117)
468     add(lista, 118)
469     add(lista, 119)
470     add(lista, 120)
471     add(lista, 121)
472     add(lista, 122)
473     add(lista, 123)
474     add(lista, 124)
475     add(lista, 125)
476     add(lista, 126)
477     add(lista, 127)
478     add(lista, 128)
479     add(lista, 129)
480     add(lista, 130)
481     add(lista, 131)
482     add(lista, 132)
483     add(lista, 133)
484     add(lista, 134)
485     add(lista, 135)
486     add(lista, 136)
487     add(lista, 137)
488     add(lista, 138)
489     add(lista, 139)
490     add(lista, 140)
491     add(lista, 141)
492     add(lista, 142)
493     add(lista, 143)
494     add(lista, 144)
495     add(lista, 145)
496     add(lista, 146)
497     add(lista, 147)
498     add(lista, 148)
499     add(lista, 149)
500     add(lista, 150)
501     add(lista, 151)
502     add(lista, 152)
503     add(lista, 153)
504     add(lista, 154)
505     add(lista, 155)
506     add(lista, 156)
507     add(lista, 157)
508     add(lista, 158)
509     add(lista, 159)
510     add(lista, 160)
511     add(lista, 161)
512     add(lista, 162)
513     add(lista, 163)
514     add(lista, 164)
515     add(lista, 165)
516     add(lista, 166)
517     add(lista, 167)
518     add(lista, 168)
519     add(lista, 169)
520     add(lista, 170)
521     add(lista, 171)
522     
```

The screenshot displays the ISIS 1225 - Reto No 2.pdf application. On the left, there are two main sections: "Reto No. 1: Curando y Explorando el MoMA" and "Reto No. 2: Curando y Explorando el MoMA: RE". Below these is a "Datos" section. The central area shows a "PowerPoint Presentation" window with a slide titled "14 / 25" and a zoom level of "75%". The slide content is a table listing various objects with their details.

ObjectID	Title	Artist/Name	Medium	Date	Dimensions
54909	(Untitled)	George Krause	Gelatin silver print	1960	1 x 4 3/4 x 7" (12.1 x 11.7 cm)
83152	(Shack and stone-chimney)	Luke Swanck	Gelatin silver print	1930	7 15/16 x 5 7/8" (20.2 x 14.9 cm)
36708	.a: Untitled (page from Breton-Tanguy Notebook), .b: Untitled (page from Breton-Tanguy Notebook)	André Breton, Yves Tanguy	.a: Ink, pencil, and colored pencil on paper; .b: pencil on paper	1941	10 7/8 x 8 5/8" (27.8 x 22.1 cm)
90129	Wooden Pen with Eraser	Hyeon Lipman	Graphite, cedar wood, rubber, and metal ferrule	1950	7/4 x 1/4" (1.9 x 0.6 cm)
44346	World War II Rally, Lower East Side	Lisette Model	Gelatin silver print	1942	13 15/16 x 10 1/2" (35.4 x 27.2 cm)
3323	Wristwatch Face	Nathan George Burgess	White gold with diamonds	1947	diam. 1 5/16" (2.3 cm)

On the right side of the interface, there is a sidebar with a search bar and a list of object IDs and titles, including 54909 (Gang), 83152 (Shack and stone-chimney), 36708 (.a: Untitled (page from Breton-Tanguy Notebook)), 90129 (Wooden Pen with Eraser), 44346 (World War II Rally, Lower East Side), and 3323 (Wristwatch).

Comparación con el reto 1:



Se nota una mejora considerable en este requerimiento del reto 2 con respecto al del reto 1. Los tiempos del reto pasado eran mucho más altos, llegando a tomar hasta 2500000 ms en el archivo large en computador de 64 gb de ram, mientras que en este reto en el archivo large lo máximo que se demoró fue 175 ms. Lo anterior sucede por el uso de mapas con keys de nacionalidades, que aproximadamente son 250, solamente se tiene que recorrer una lista de 250 elementos, mientras que en el reto pasado se tenía que recorrer toda la lista de artworks para todas las obras. Además, se mejora el tiempo de búsqueda de la nacionalidad, debido a la implementación del mapa se pudo relacionar de una manera más fácil la nacionalidad del artista con la de cada obra, mientras que en el reto pasado se tenía que recorrer toda la lista de artistas para encontrar a los artistas de cada obra. Finalmente, para hacer el ordenamiento en este reto se utilizó selection editado, esto hace que ordenar el top 10 de nacionalidades sea más rápido

## Requerimiento 5

### Pruebas de tiempos de ejecución



Figura 8. Pruebas de tiempos de ejecución req5. Computador con 8gb de Ram

### Análisis de complejidad:

```
def transportarObrasDespartamento(catalog, departamento): # Requerimiento Grupal 5: Función Principal
    exisDepartamento=mp.contains(catalog["Department"], departamento)

    obrasArteDepto=lt.newList("ARRAY_LIST")
    if exisDepartamento:
        obrasDepartamento=mp.get(catalog["Department"], departamento)["value"]["Artworks"]
        for index in lt.iterator(obrasDepartamento):
            obra=lt.getElement(catalog["artworks"], index)
            obra["NombresArtistas"]=nombresArtistas(catalog, obra["ConstituentID"])
            lt.addLast(obrasArteDepto, obra)
            ##líneas con operaciones O(1)

        size=obrasArteDepto["size"]

    obrasDeptoCopy=lt.subList(obrasArteDepto, 0, size) #se copia la lista
    precioSorted=lt.subList((selection.sortEdit(obrasDeptoCopy, cmpArtworkBy Price, 5)), 1, 5)
    fechaSorted=lt.subList((selection.sortEdit(obrasArteDepto, cmpArtworkByDate, 5)), 1, 5)
    respuestaLPrecio=precioSorted
    respuestaLFecha=fechaSorted
    return precioTotalEnvio, pesoTotal, respuestaLFecha, respuestaLPrecio, size, obrasDeptoCopy
```

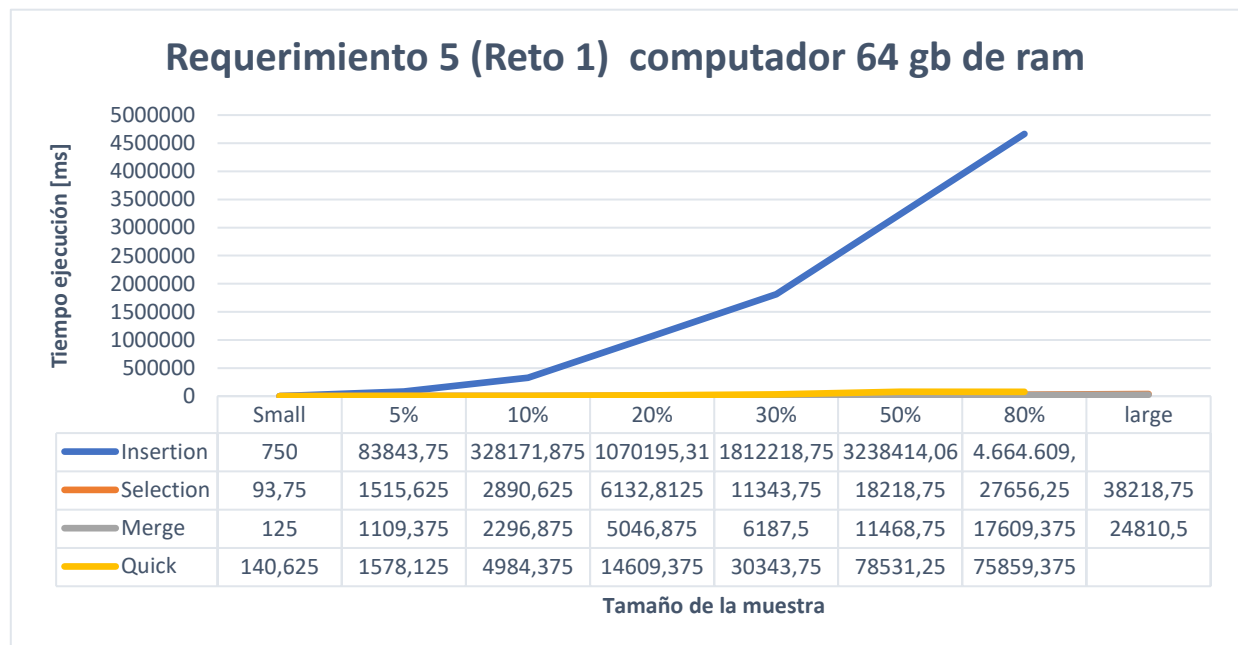
- Acceder a un departamento específico tiene un tiempo  $O(1)$ , dado que utilizamos un mapa. Seguido a esto recorreremos todos los índices (que representan una obra del departamento) tiene un tiempo  $O(N)$

- Para acceder a una obra de la lista de artworks  $O(1)$  en vista que poseemos el índice de la obra
- Para cada obra se hacen distintas operaciones que tienen un tiempo  $O(1)$  dado que solo es acceder a llaves de esta obra en específico
- Sacar un sublist() tiene tiempo  $O(N)$
- Hacer dos veces ordenamiento con selection.sortEdit() sería  $O(5N)*2 = O(10N) = O(N)$

Por ende, la complejidad de este algoritmo es:

$$O(N) + O(N) + O(N) = O(3N) = O(N)$$

### Comparación con el reto 1:



El requerimiento 5 también tiene una mejora en los tiempos de ejecución. Esto sucede porque ahora se puede acceder a todas las obras de un departamento en un tiempo  $O(1)$ , mientras que antes se tenía que recorrer todas las obras para hallar su departamento. Similarmente, debido al uso de índices como posiciones en los valores del mapa, se puede acceder a una obra directamente sin comprometer mucho espacio. Adicionalmente, dado que se utilizó el algoritmo de selection editado se pudo ordenar las listas de obras más antiguas y costosas en un menor tiempo.

### Requerimiento 6

## Pruebas de tiempos de ejecución

Parámetros: 7 artistas / 1914 y 1939

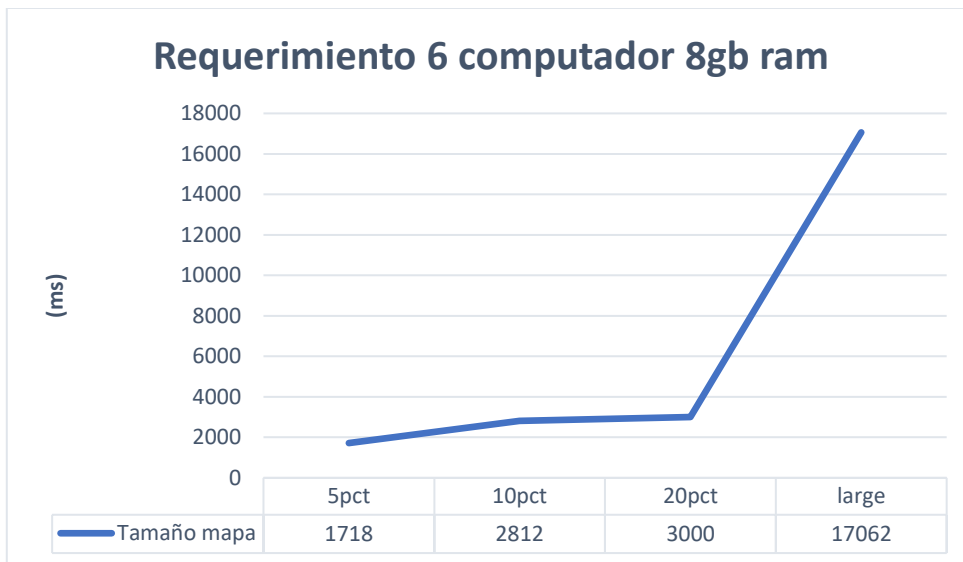


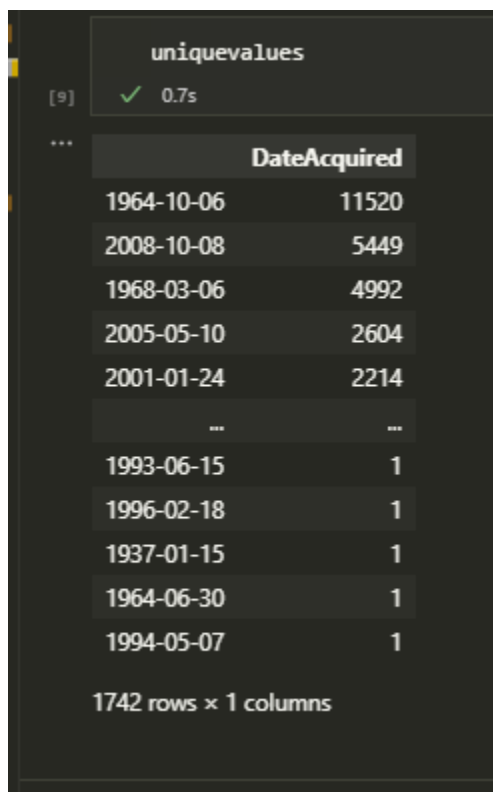
Figura 9 Requerimiento 6 tiempos de ejecución

## Análisis de complejidad

Caso promedio	$O(K*N)$ El caso promedio de este requerimiento en este requerimiento es $O(N)$ , dado que se tienen que buscar las obras de los respectivos artistas y recorrer hasta que se llegue al número de artistas deseados
Peor caso	$O(N^2)$ El peor caso se dará cuando todos los artistas tengan la misma cantidad de obras. De esta manera se tendrá que recorrer varias veces sus respectivas obras para encontrar cual de ellos es el mejor



## Extras



uniquevalues

[9] ✓ 0.7s

...

DateAcquired	
1964-10-06	11520
2008-10-08	5449
1968-03-06	4992
2005-05-10	2604
2001-01-24	2214
...	...
1993-06-15	1
1996-02-18	1
1937-01-15	1
1964-06-30	1
1994-05-07	1

1742 rows × 1 columns

*Figura 10. Columna DateAcquired*

- Tiene 1742 fechas únicas
- La fecha mínima es 1929-11-19
- La fecha máxima es 2020-10-26
- La fecha con mayor cantidad de compras hechas es 1964-10-06 con 11520 obras
- Chaining – Cantidad inicial 2k – factor de carga 4

uniquevaluesArtists	
[4]	✓ 0.5s
***	
BeginDate	
0	3674
1942	188
1938	180
1943	179
1937	170
...	...
1835	1
2012	1
1799	1
1787	1
1765	1
236 rows × 1 columns	

Figura 11 Valores únicos años de nacimiento