

# ANÁLISIS DEL RETO

Lucas Valbuena León Cod 202311538

Juan José Cortés Villamil Cod 202325148

Alisson Moreno Cod 202120330

## Carga de datos

### Descripción

```
def add_data(data_structs, part, data, c):  
  
    """  
  
    Función para agregar nuevos elementos a la lista  
  
    """  
  
    #TODO: Crear la función para agregar elementos a una lista  
  
    if part == "employments_types":  
  
        key= f"{data['id']}"  
  
  
    elif part == "jobs":  
  
        lt.addLast(data_structs["jobs_cd"], data)  
  
        offer = data  
  
        offer['city'] = offer['city'].strip()  
  
        if offer['city'][0] != offer['city'][0].upper():  
  
            offer['city'] =  
                f"{offer['city'][0].upper()}{offer['city'][1:]}"  
  
        key = f"{data['id']}"
```

```

exist_country =
    mp.contains(data_structs['major_structure'], offer["country_code"])

if not exist_country:

    mp.put(data_structs["major_structure"], offer["country_code"],
    mp.newMap(numelements=1000, maptype="PROBING",
    loadfactor=0.5))

exist_city =
    mp.contains(me.getValue(mp.get(data_structs["major_structure"],
    offer["country_code"])), offer["city"])

if not exist_city:

    mp.put(me.getValue(mp.get(data_structs["major_structure"],
    offer["country_code"])), offer["city"],
    mp.newMap(numelements=1000, maptype="PROBING",
    loadfactor=0.5))

exist_business =
    mp.contains(me.getValue(mp.get(me.getValue(mp.get(data_structs
    ["major_structure"], offer["country_code"])), offer["city"])),
    offer["company_name"])

if not exist_business:

    mp.put(me.getValue(mp.get(me.getValue(mp.get(data_structs["maj
    or_structure"], offer["country_code"])), offer["city"])),
    offer["company_name"], mp.newMap(numelements=7,
    maptype="PROBING", loadfactor=0.5))

    "cmpfunction pendiente"

    mp.put(me.getValue(mp.get(me.getValue(mp.get(me.getValue(mp.ge
    t(data_structs["major_structure"],

```

```
offer["country_code"])), offer["city"])),
offer["company_name"])), "senior", om.newMap(omaptype='RBT',
cmpfunction=cmpFunctionFechaRBT))
```

```
mp.put(me.getValue(mp.get(me.getValue(mp.get(me.getValue(mp.ge
t(data_structs["major_structure"],
offer["country_code"])), offer["city"])),
offer["company_name"])), "mid",
om.newMap(omaptype='RBT', cmpfunction=cmpFunctionFechaRBT))
```

```
mp.put(me.getValue(mp.get(me.getValue(mp.get(me.getValue(mp.ge
t(data_structs["major_structure"],
offer["country_code"])), offer["city"])),
offer["company_name"])), "junior",
om.newMap(omaptype='RBT', cmpfunction=cmpFunctionFechaRBT))
```

```
divisa_oferta = None
```

```
salario_minimo = 0
```

```
salario_promedio_oferta= 0
```

```
for employment_type in
```

```
lt.iterator(me.getValue(mp.get(data_structs["employments_types
"], offer["id"]))):
```

```
divisa_oferta = employment_type["currency_salary"]
```

```
promedio_salarial =
```

```
convertir_divisas(employment_type["promedio_salarial"],
employment_type["currency_salary"], c)
```

```
salary_from =
```

```
convertir_divisas(employment_type["salary_from"],
employment_type["currency_salary"], c)
```

```
salario_promedio_oferta +=
```

```
promedio_salarial/lt.size(me.getValue(mp.get(data_structs["emp
loyments_types"], offer["id"])))
```

```

        if salary_from != "":
            if salario_minimo == 0:
                salario_minimo = salary_from
            if salary_from < salario_minimo:
                salario_minimo = salary_from

        #if employment_type["promedio_salarial"] < salario_minimo:
        #    salario_minimo = employment_type["promedio_salarial"]
data["salario_minimo"] = salario_minimo
data["salario_promedio"] = salario_promedio_oferta
habilidades_solicitadas = lt.newList("ARRAY_LIST")
promedio_habilidades = 0
for skill in
    lt.iterator(me.getValue(mp.get(data_structs["skills"],
offer["id"]))):
        lt.addLast(habilidades_solicitadas, skill["name"])
        promedio_habilidades =
            int(skill["level"])/lt.size(me.getValue(mp.get(data_structs["s
kills"], offer["id"])))
data["habilidades_solicitadas"] = habilidades_solicitadas
data["promedio_habilidad"] = promedio_habilidades

exist_salaryOffer = om.contains(data_structs["salaries_offers"],
data["salario_minimo"])
if not exist_salaryOffer:
    om.put(data_structs["salaries_offers"],
data["salario_minimo"], lt.newList("ARRAY_LIST"))

```

```

lt.addLast(me.getValue(om.get(data_structs["salaries_offers"],
    data["salario_minimo"])), offer)

data["salario_minimo"] = salario_minimo

data["salario_promedio"] = salario_promedio_oferta

data["divisa_para_revertir"] = divisa_oferta

"""

if divisa_oferta != None:

    data["salario_minimo"] = revertir_divisas(salario_minimo,
divisa_oferta, c)

    data["salario_promedio"] =
    revertir_divisas(salario_promedio_oferta, divisa_oferta, c)

"""

exist_fecha =
    om.contains(me.getValue(mp.get(me.getValue(mp.get(me.getValue(
mp.get(me.getValue(mp.get(data_structs["major_structure"], offer
r["country_code"])))

, offer["city"]))), offer["company_name"])), offer["experience_lev
el"])), offer["published_at"][:10])

if not exist_fecha:

    om.put(me.getValue(mp.get(me.getValue(mp.get(me.getValue(mp.ge
t(me.getValue(mp.get(data_structs["major_structure"], offer["co
untry_code"])))

, offer["city"]))), offer["company_name"])), offer["experience_lev

```

```

el"])), offer["published_at"][:10], om.newMap(omaptype="RBT",
cmpfunction=cmpFunctionSalarioRBT))

exist_salary =
    om.contains(me.getValue(om.get(me.getValue(mp.get(me.getValue(
mp.get(me.getValue(mp.get(me.getValue(mp.get(data_structs["maj
or_structure"],offer["country_code"])))

,offer["city"])),offer["company_name"])),offer["experience_lev
el"])), offer["published_at"][:10])),salario_minimo)

if not exist_salary:

    om.put(me.getValue(om.get(me.getValue(mp.get(me.getValue(mp.ge
t(me.getValue(mp.get(me.getValue(mp.get(data_structs["major_st
ructure"],offer["country_code"])))

,offer["city"])),offer["company_name"])),offer["experience_lev
el"])), offer["published_at"][:10])),salario_minimo,
lt.newList("ARRAY_LIST"))

lt.addLast(me.getValue(om.get(me.getValue(om.get(me.getValue(m
p.get(me.getValue(mp.get(me.getValue(mp.get(me.getValue(mp.get
(data_structs["major_structure"],offer["country_code"])))

,offer["city"])),offer["company_name"])),offer["experience_lev
el"])), offer["published_at"][:10])),salario_minimo)), offer)

exist_tamaño = om.contains(data_structs["size_empresas"],
data["company_size"])

```

```

if not exist_tamaño:

    om.put(data_structs["size_empresas"], data["company_size"],
    lt.newList("ARRAY_LIST"))

ya_esta =
    lt.isPresent(me.getValue(om.get(data_structs["size_empresas"],
    data["company_size"])), offer["company_name"])

if not ya_esta:

    lt.addLast(me.getValue(om.get(data_structs["size_empresas"],
    data["company_size"])), offer["company_name"])

exists_jobdate=
    om.contains(data_structs["jobs"], data["published_at"][:10])

if not exists_jobdate:

    om.put(data_structs["jobs"], data["published_at"][:10],
    lt.newList("ARRAY_LIST"))

    lt.addLast(me.getValue(om.get(data_structs["jobs"], data["publi
    shed_at"][:10])), offer)

elif part == "multilocations":

    key = f"{data['id']}"

else:

```

```

key = f"{data['id']}"

if part != "size" and part != "jobs":

    existkey = mp.contains(data_structs[part],key)

    if not existkey:

        mp.put(data_structs[part], key, lt.newList("ARRAY_LIST"))

    entry = mp.get(data_structs[part], key)

    lt.addLast(me.getValue(entry), data)

```

La carga de datos está orientada de la siguiente manera: la superficie es un mapa indexado por países, por cada llave país existe un mapa indexado por ciudades, por cada llave ciudad existe un mapa indexado por empresas, dentro de cada llave empresa hay un mapa y en la llave existen los niveles de experticia senior, mid y junior. Dentro de cada llave dentro de junior, mid o senior existe un árbol indexado por fechas y por cada nodo del árbol hay otro árbol indexado por salarios mínimos y finalmente cada nodo salario tiene un un mapa para el tipo de trabajo, y por cada llave de estas se encuentra un arreglo con las ofertas. Cabe aclarar, lo que se acaba de describir es la estructura principal de la carga de datos, sin embargo también hicimos un árbol indexado por fechas, uno por tamaños de empresas y otro por salarios. El primero y último tienen en cada nodo una lista de ofertas. El segundo tiene como valor de cada nodo una lista de nombres de empresas. Además de esto están los mapas de employments\_types, skills y multilocations indexados por id.

<b>Entrada</b>	Estructuras de datos del modelo, ID.
<b>Salidas</b>	El elemento con el ID dado, si no existe se retorna None
<b>Implementado (Sí/No)</b>	Sí se implementó y lo realizó Juan Jose Cortés.



## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Iterar por las líneas que lee el csv.reader()	$O(N)$
add_data()	
Cargar employment_types, skills o multilocations	
mp.contains(mapa, llave)	$O(N/M)$
mp.put()	$O(N/M)$
mp.get()	$O(N/M)$
lt.addLast()	$O(1)$
Cargar las ofertas de trabajo	
mp.contains() para verificar si ya existe el país en el mapa "major_structure".	$O(N)$
Si el país no existe mp.put(major_structure, país, mapa_ciudades)	$O(N/M)$
mp.contains() para verificar si ya existe la ciudad en el mapa del país	$O(M)$
Si la ciudad no existe mp.put(país, ciudad, mapa_empresas)	$O(N/M)$
mp.contains() para verificar si ya existe la empresa en el mapa de ciudad	$O(N)$
Si no existe la llave empresa en el mapa de ciudad, mp.put(ciudad, empresa, mapa_experticia)	$O(N/M)$
mp.put() para poner las llaves "mid", "junior" y "senior" en cada mapa empresa	$O(3)$ en el peor caso, entonces $O(1)$
Iterar por los employments_types para ponerle a cada oferta su salario mínimo y su salario_promedio	$O(N)$
lt.newList() de las habilidades solicitadas de la oferta	$O(1)$
Verificar si existe el índice del salario mínimo de la oferta en el árbol RBT "salaries_offers" (diferente a major_structure)	$O(\log N)$
Si no existe se pone la llave del salario y una lista de ofertas om.put(..., lt.newList())	$O(2\log N)$
lt.addLast() para añadir la oferta a la lista del índice	$O(1)$
Verificar si existe la llave fecha de la oferta en el árbol que hay en cada llave "mid", "junior"... (major_structure)	$O(\log N)$
Si no existe se pone om.put()	$O(2\log N)$
Los últimos dos pasos se repiten pero en el árbol que hay en cada nodo fecha, verificando si en el árbol esta llave salario	$O(\log N)$
lt.addLast() Añadir la oferta a la lista de ofertas a cada nodo salario	$O(1)$

mp.contains() para verificar si existe la llave tamaño empresa en el árbol "size_empresas"	$O(\log N)$
Si no lo contiene om.put() para poner la lista en el nodo tamaño empresa	$O(2\log N)$
lt.isPresent() para no tener una lista en la que se repitan nombres de empresas	$O(N)$
Verificar si existe el índice fecha en el árbol "jobs"	$O(\log N)$
Si no existe om.put(árbol, fecha, ARRAY_LIST)	$O(2\log N)$
lt.addLast()	$O(1)$
<b>TOTAL</b>	<b><math>O(N^2)</math></b>

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones.

Procesadores	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz
Memoria RAM	16,0 GB
Sistema Operativo	Windows 11 Pro

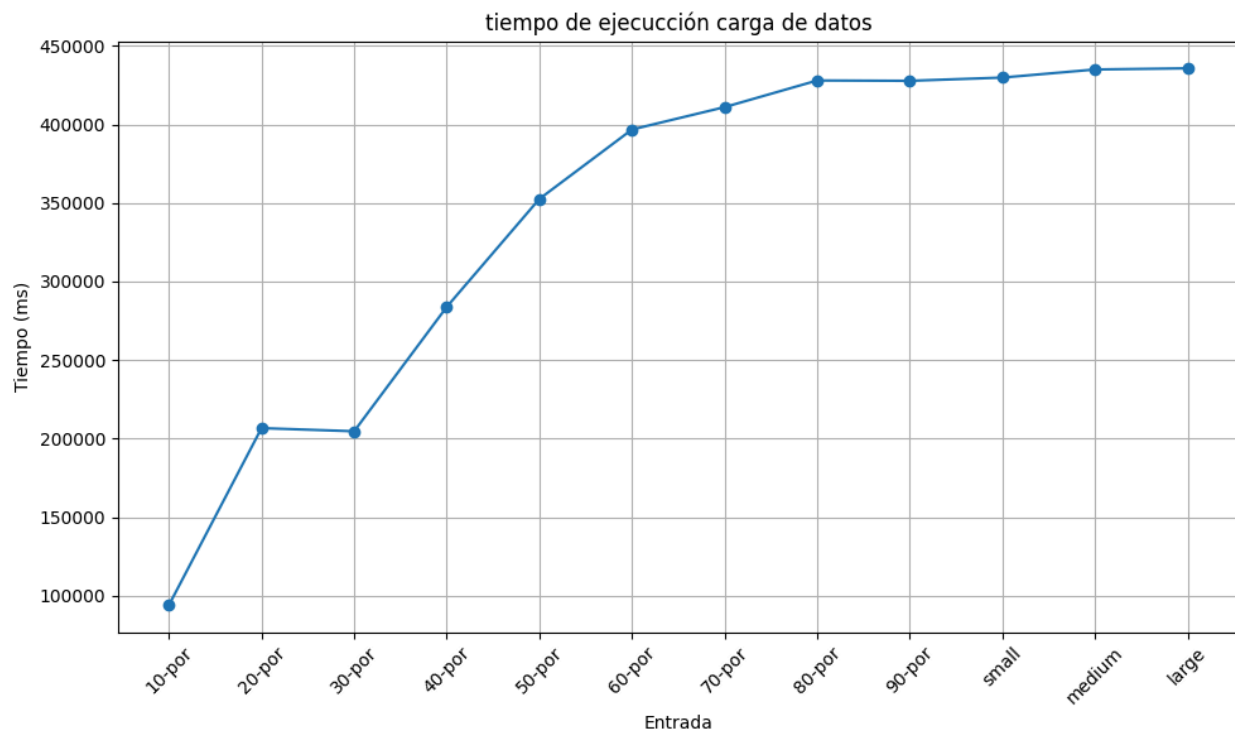
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	93905.379
20-por	206670.706
30-por	204656.663
40-por	283874.241
50-por	352636.904
60-por	396850.813
70-por	411109.237
80-por	428079.877
90-por	427866.341
small	429928.422
medium	435081.681
large	435836.424

## Gráficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

Por la gráfica vemos un comportamiento logarítmico, esto porque en ningún momento se llega al peor caso de los `mp.put()` o `om.put()` muy probablemente gracias al tamaño de cada mapa, ya que al no saber la cantidad de tuplas, en algunos casos, el tamaño pudo haber sido muy grande haciendo realmente difícil que se dieran colisiones. En un principio íbamos a hacer la carga de datos completamente invertida, es decir los árboles en las 'capas' exteriores pero nos dimos cuenta de que significaban muchos más ciclos y una complejidad quizás mucho mayor. Igual el hecho de que se comience a estabilizar el tiempo de ejecución en los últimos tamaños de muestra también se debe a que ya la diferencia de datos no era tan significativa como en un principio. Sin más que decir, la carga de datos demostró una gran rapidez a la hora de aplicar los requerimientos y mucha facilidad para filtrar por cosas como país, ciudad o empresa.

## Requerimiento <<1>>

### Descripción

```
def req_1(data_structs, fecha0, fecha1):
```

```
    """
```

```

Función que soluciona el requerimiento 1

"""

# TODO: Realizar el requerimiento 1

#fecha0 = datetime.strptime(fecha0,"%Y-%m-%d")

#fecha1 = datetime.strptime(fecha1,"%Y-%m-%d")

cantidad_ofertas = 0


total_ofertas = lt.newList("ARRAY_LIST")

listas_ofertas = om.values(data_structs, fecha0, fecha1)

for lista_oferta in lt.iterator(listas_ofertas):

    cantidad_ofertas += lt.size(lista_oferta)

    for offer in lt.iterator(lista_oferta):

        lt.addLast(total_ofertas, offer)


cinco = False

if lt.size(total_ofertas) > 10:

    cinco =True


merg.sort(total_ofertas, sort_crit_reciente_a_antiguo)

return cantidad_ofertas, total_ofertas, cinco

```

Este requerimiento no supuso realmente mayor complejidad, el mayor reto fue empezar a entender cómo usar la estructura de datos que creamos en la carga. Sin embargo, hacer este requerimiento permitió los demás requerimientos al desarrollar nuestro entendimiento sobre cómo usar la estructura de los árboles en general de la carga de datos. Ahora, básicamente este requerimiento recibe como parámetros de entrada un umbral mínimo y uno máximo de fechas en formato %Y-%M-%D y busca devolver las ofertas dentro de ese rango de fechas. Para facilitar el requerimiento hicimos un árbol indexado por fechas donde cada valor era una lista de ofertas según correspondiere, sin discriminar las

ofertas por algún otro factor. De este modo, usamos la función .values(map, keylo, keyhi) de DISClib para obtener las listas de las ofertas que nos interesan de acuerdo al rango establecido por el usuario. Finalmente se itera sobre esas listas para unificar todo en un solo ARRAY\_LIST y organizarlas desde el más reciente hasta el más antiguo usando el siguiente criterio de ordenamiento:

```
def sort_crit_reciente_a_antiguo(oferta1, oferta2):

    fecha1 = datetime.strptime(oferta1["published_at"],
"%Y-%m-%dT%H:%M:%S.%fZ")

    fecha2 = datetime.strptime(oferta2["published_at"],
"%Y-%m-%dT%H:%M:%S.%fZ")

    if fecha1 > fecha2:

        return True

    elif fecha1 < fecha2:

        return False

    else:

        if oferta1["salario_minimo"] > oferta2["salario_minimo"]:

            return True

        elif oferta1["salario_minimo"] < oferta2["salario_minimo"]:

            return False

        else:

            return True
```

<b>Entrada</b>	Fecha inicial de periodo de consulta, fecha final de periodo de consulta.
<b>Salidas</b>	Cantidad de ofertas totales, total_ofertas (Lista de las ofertas organizadas por fecha), cinco: bool.
<b>Implementado (Sí/No)</b>	Sí. Implementado por Juan José Cortés Villamil

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
-------	-------------

Crear lista vacía (lt.newList("ARRAY_LIST"))	$O(1)$
Obtener listas de ofertas en el rango de fechas consultado (om.values(map, keylo, keyhi))	$O(2\log_2 N)$
Ciclo para iterar por las listas de ofertas	$O(N)$ peor caso
Ciclo para iterar por las ofertas de las listas de ofertas	$O(N)$
lt.addLast()	$O(1)$
merg.sort()	$O(N \log N)$
<b>TOTAL</b>	<b><math>O(N^2)</math></b>

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron 2022-04-12 y 2023-12-24.

Procesadores	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10

## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

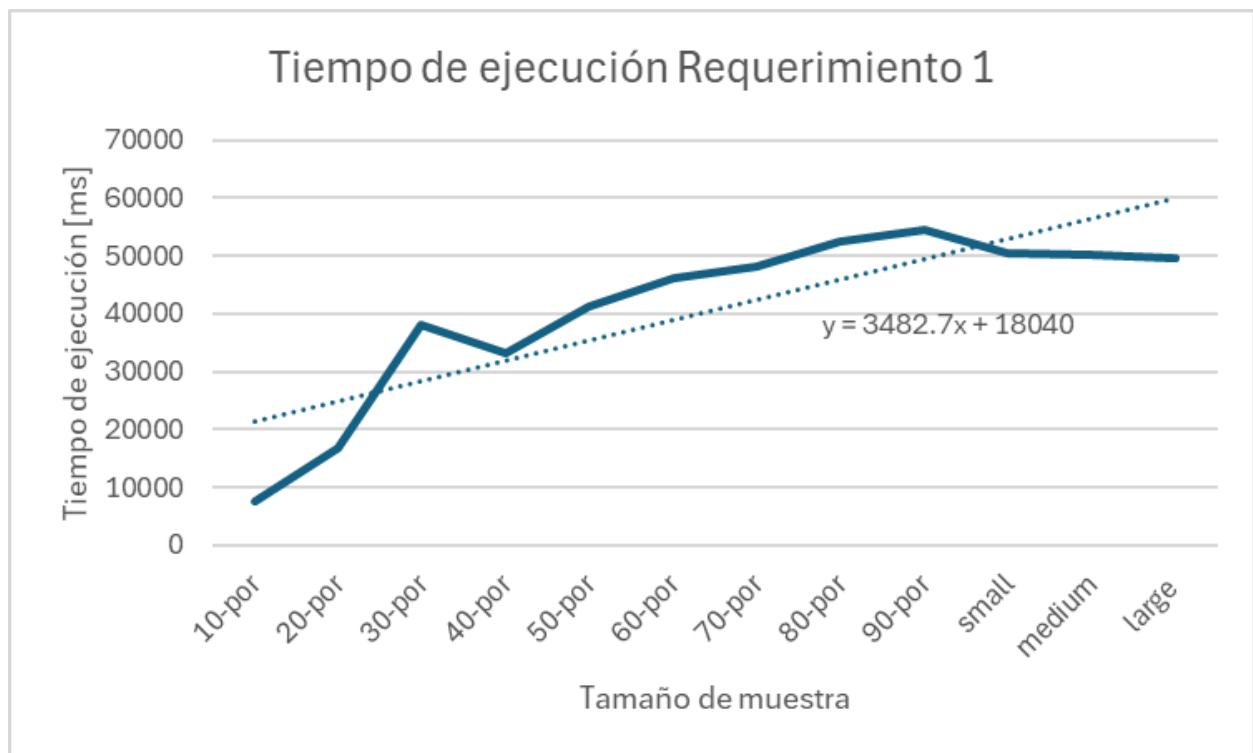
Entrada	Tiempo (ms)
10-por	7493.292
20-por	16727.5668
30-por	37980.2812
40-por	33062.9439
50-por	41242.5086
60-por	46179.228
70-por	48055.5424
80-por	52528.6104
90-por	54441.5091
small	50412.6539
medium	50310.0012

large

49688.5049

## Gráficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

A partir de la gráfica observamos un comportamiento más bien logarítmico en contraposición con el comportamiento cuadrático que se había planteado previamente. De todos modos, es difícil asegurar que tiene un comportamiento logarítmico. Incluso me atrevería a decir que tiene un comportamiento lineal, que se observa en los primeros tamaños de muestra. Digo esto porque como todas las pruebas las hice con el mismo rango de fechas 2022-04-12, hasta 2023-12-24 es posible que a partir de 80-por más o menos, el tiempo de ejecución se estabiliza alrededor de los 50000 ms, mostrando que el tiempo de ejecución para una misma entrada tiende a ser constante pero si pusiera un rango de fechas que abarcara todos los datos muy probablemente la gráfica tendría una tendencia lineal.

## Requerimiento <<2>>

```
def req_2(data_structs, salario_minimo, salario_maximo):  
  
    """  
  
    Función que soluciona el requerimiento 2  
  
    """  
  
    # TODO: Realizar el requerimiento 2  
  
  
    cantidad_ofertas = 0  
  
  
    total_ofertas = lt.newList("ARRAY_LIST")  
    final = lt.newList("ARRAY_LIST")  
  
    listas_ofertas = om.values(data_structs, salario_minimo,  
salario_maximo)  
  
  
    for lista_oferta in lt.iterator(listas_ofertas):  
        cantidad_ofertas += lt.size(lista_oferta)  
  
        for offer in lt.iterator(lista_oferta):  
            lt.addLast(total_ofertas, offer)  
  
  
  
    if total_ofertas !=None:  
        merg.sort(total_ofertas, sort_crit_reciente_a_antiguo)  
  
  
    if lt.size(total_ofertas) >= 10:  
        sb1 = lt.subList(total_ofertas, 1, 5)  
  
        for zc in lt.iterator(sb1):
```



```

lt.addLast(final, zc)

sb2 = lt.subList(total_ofertas, (lt.size(total_ofertas)-5), 5)

for zt in lt.iterator(sb2):

    lt.addLast(final, zt )

elif lt.size(total_ofertas)<10 and lt.size(total_ofertas)>0:

    final = total_ofertas

elif lt.size(total_ofertas)==0:

    final = None

return cantidad_ofertas, final

```

## Descripción

Este requerimiento consistía en dar las ofertas laborales entre un rango de salarios. Primero destinamos dos arreglos para tener las ofertas finales, uno para poner el total de ofertas y otro será el arreglo que se pasa a controller para imprimirlas ya que si son más de diez ofertas solo se imprimen las primeras cinco y las últimas cinco. Después vamos a buscar con la función values todos los valores del árbol “salaries\_offers” que estén dentro del rango de salarios, se itera con un for anidado para poner esas ofertas en uno de los arreglos y al mismo tiempo contarlas. Al final antes de pasar las ofertas a otro arreglo para imprimirlo utilizamos la función merge sort y una función de comparación para organizar cronológicamente.

<b>Entrada</b>	El data_structurs, el salario minimo y el salario maximo
<b>Salidas</b>	Las ofertas que cumplan el rango de salarios dado.
<b>Implementado (Sí/No)</b>	Si se implementó y lo hizo Lucas Valbuena.

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1: Creación nuevos arreglos	$O(1)$
Paso 2 : función values	$O(2 \log N)$
Paso 3: Iterar sobre la lista de ofertas	$O(2 \log N)$

Paso 4: AddLast	$O(1)$
Paso 5: Sacar sublista	$O(1)$
<b>TOTAL</b>	$O(2 \log N)$

## 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).

Input:

salario mínimo: 5000

salario máximo: 6500

Procesadores	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz
Memoria RAM	16,0 GB
Sistema Operativo	Windows 11 Pro

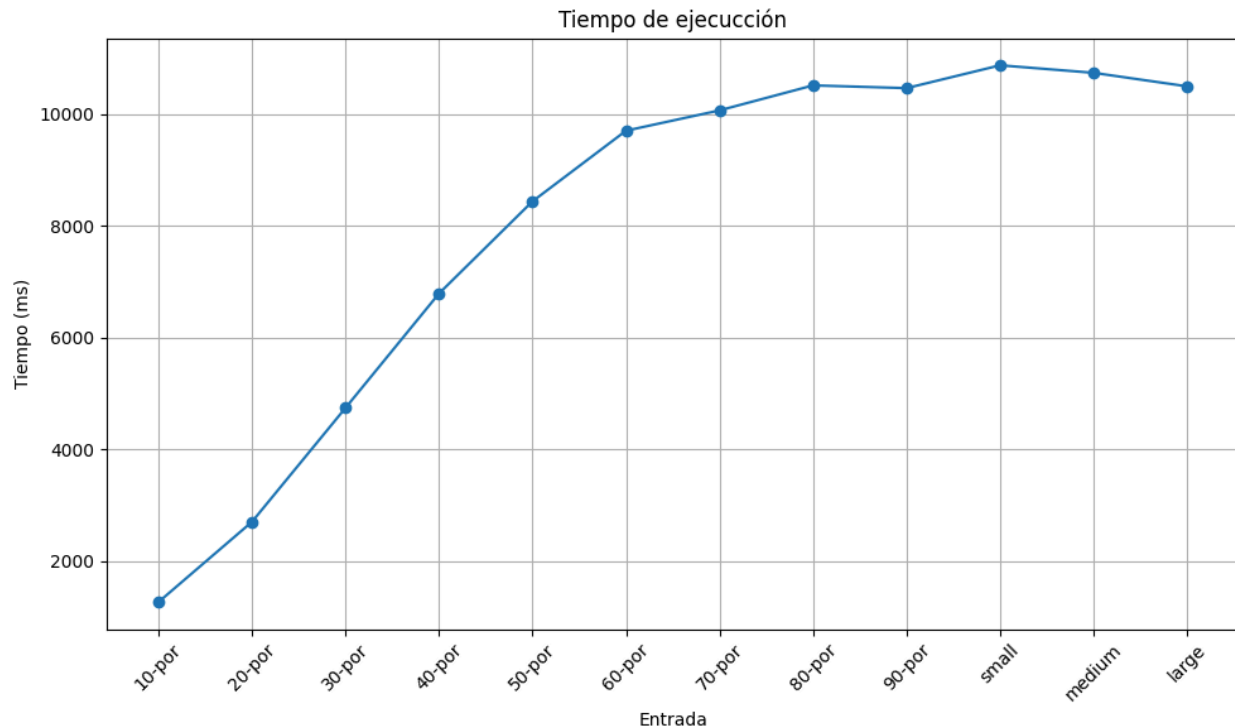
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	1267.943
20-por	2704.546
30-por	4743.820
40-por	6791.829
50-por	8442.085
60-por	9706.868
70-por	10070.774
80-por	10518.006
90-por	10468.882
small	10877.611
medium	10742.972
large	10500.939

## Graficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

Después de ver la distribución de los puntos en la gráfica podemos apreciar de manera clara como la función tiene una semejanza a una función logarítmica y esto se debe a que entre mayor es la entrada en el programa el tiempo de ejecución se vuelve cada vez más parecido. A pesar de que el requerimiento no abarca una complejidad tan grande, el recorrido de la estructura principal conforme llegan más datos se vuelve más demorada para todos los requerimientos. Hablando sobre la complejidad espacial, es uno de los requerimientos menos costosos ya que maneja muy pocas estructuras auxiliares y por lo tanto la memoria no se ve afectada en gran medida.

## Requerimiento <<3>>

```
def req_3(data_structs,pais, experticia,N ):
    """
    Función que soluciona el requerimiento 3
    """
    catalogo=data_structs["major_structure"]
    lista_arboles=lt.newList("ARRAY_LIST")
    ofertas=lt.newList("ARRAY_LIST")
```

```

mapaciudades= me.getValue(mp.get(catalogo,pais))

mapas_empresas=mp.valueSet(mapaciudades)

for mapa in lt.iterator(mapas_empresas):

    mapas_experticia= mp.valueSet(mapa)

    for mapa_e in lt.iterator(mapas_experticia):

        if experticia != "indiferente":

            mapa_fechas= me.getValue(mp.get(mapa_e,experticia))

            lt.addLast(lista_arboles,mapa_fechas)

        else:

            valores=mp.valueSet(mapa_e)

            for element in lt.iterator(valores):

                lt.addLast(lista_arboles,element)

ofertas= get_ofertas(lista_arboles,N)

ordenada= merg.sort(ofertas,sort_crit_req3)

return ordenada,lt.size(ordenada)

def get_ofertas(lista_arboles,N):

    respuesta=lt.newList("ARRAY_LIST")

    while lt.size(respuesta) < N:

        mas_reciente= "0000-00-00"

        d_interes= None

        i=0

        indice=0

        for arbol in lt.iterator(lista_arboles):

            if not om.isEmpty(arbol):

```

```

        reciente= om.maxKey(arbol)

        if reciente > mas_reciente:

            indice=i

            mas_reciente=reciente

            d_interes=arbol

        i+=1

    arbol_salarios= me.getValue(om.get(d_interes,mas_reciente))

    lt.deleteElement(lista_arboles,indice)

    arbol_nuevo=om.deleteMax(d_interes)

    lt.addLast(lista_arboles,arbol_nuevo)

    for key in lt.iterator(om.keySet(arbol_salarios)):

        lista_ofertas=me.getValue(mp.get(arbol_salarios,key))

        for element in lt.iterator(lista_ofertas):

            lt.addLast(respuesta,element)

    if lt.size(respuesta)>= N:

        return lt.subList(respuesta,0,N)

```

## Descripción

La idea general de este requerimiento es retornar un número determinado de ofertas laborales, teniendo como parámetro un país y un nivel de experticia específico. En primer lugar, se obtiene el valor del mapa de países correspondiente a la llave “país” que entra como parámetro, y se itera por cada mapa de empresas para obtener el value set de la llave experticia, en el mapa experticia. Se itera sobre este value set, y cada árbol se añade a una lista de tipo ARRAY LIST. Luego, se hace uso de una función auxiliar, “get\_ofertas” a la que entra por parámetro esta lista de árboles y un número N, aquí, por cada árbol ordenado se comprueba que no esté vacío y se obtiene la llave más grande (fecha) del mismo, hasta llegar al árbol de salarios, y finalmente añadir las ofertas a una lista.

<b>Entrada</b>	Datastructs (el model), N, pais, y experience level
<b>Salidas</b>	Cantidad de ofertas totales de la consulta, Las N ofertas ordenadas por fecha y salario

Implementado (Sí/No)	Sí. Implementado por Alisson Moreno
----------------------	-------------------------------------

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
<code>me.getValue(mp.get(catalogo,pais))</code>	O(1)
<code>mp.valueSet()</code>	O(N)
iterar por el value set del mapa ciudades	O(M) siendo M el tamaño del mapa
iterar por el value set del mapa empresas	O(M) siendo M el tamaño del mapa
iterar por el value set del mapa experticia	O(3) 1 para cada nivel de experticia
<code>addLast()</code>	O(1)
<code>while lt.size(respuesta) &lt; N:</code>	O(N)
<code>for arbol in lt.iterator(lista_arboles):</code>	O(N)
<code>om.maxkey()</code>	O(1)
<code>om.isEmpty()</code>	O(1)
For anidado llaves y ofertas del mapa salarios	O(N*M) N^2 en el peor caso
<code>merge.sort()</code>	O(N log N)
<b>TOTAL</b>	<b><math>O(N^4)</math></b>

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron N = 5, PL y mid.

Procesadores	AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10

## Tablas de datos

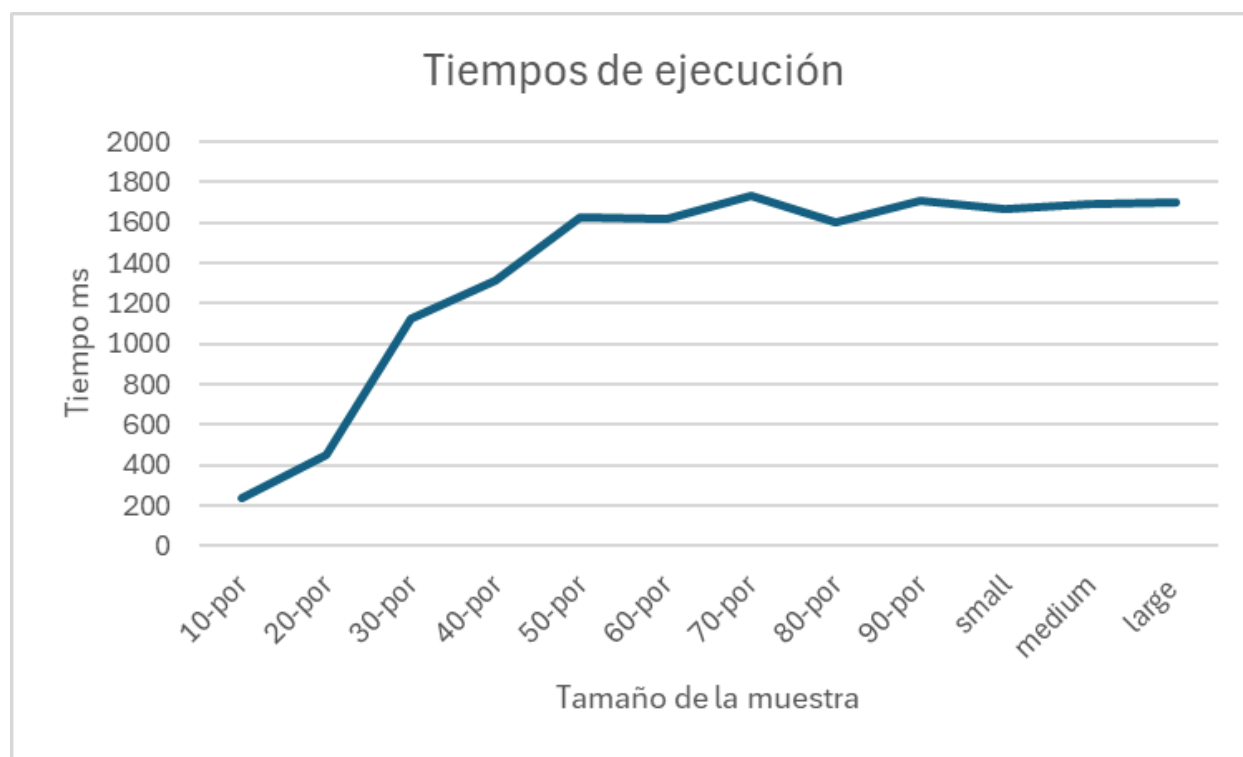
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	239.25
20-por	452.03

30-por	1124.32
40-por	1315.95
50-por	1625.55
60-por	1618.78
70-por	1736.69
80-por	1605.59
90-por	1712.36
small	1669.85
medium	1695.41
large	1703.33

## Gráficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

Para este requerimiento, se puede afirmar que el segmento en el que se hacen las iteraciones sobre cada uno de los mapas correspondientes a la estructura de datos que realizamos para la carga de datos, es donde más complejidad hay a lo largo del código, ya que es necesario recorrer cada uno de los mapas para poder obtener los valores asociados a las llaves de país y experticia que entran por parámetro a la función. Debido a esto, es consistente que a medida que sean más los datos y ofertas cargadas, se demore más, pues necesita iterar sobre más elementos para poder encontrar las N ofertas publicadas por un país y cierto nivel de experticia. Se podría decir que se trató de mitigar este orden de crecimiento utilizando un while que se rompe justo cuando llega a las N ofertas, ya que, de no tenerlo, se tendrían que iterar sobre todas las ofertas y al final hacer una sublista.

## Requerimiento <<4>>

### Descripción

```
def req_4(data_structs, N, nombre_ciudad, ubicacion):  
  
    """  
    Función que soluciona el requerimiento 4  
    """  
  
    ciudad = None  
  
    lista_llavesp = mp.keySet(data_structs["major_structure"])  
  
    for llave_pais in lt.iterator(lista_llavesp):  
  
        exists_city =  
mp.contains(me.getValue(mp.get(data_structs["major_structure"], llave_pais)  
, nombre_ciudad)  
  
        if exists_city and llave_pais != "":  
  
            print(f"|{llave_pais}|")
```



```

print(mp.keySet(me.getValue(mp.get(data_structs["major_structure"], llave_pais))))

        ciudad =
me.getValue(mp.get(me.getValue(mp.get(data_structs["major_structure"], llave_pais)), nombre_ciudad))

        break

    if ciudad == None:

        return False, False, False

empresas = mp.keySet(ciudad)

ofertasSN = lt.newList("ARRAY_LIST")

ofertas_MD = lt.newList("ARRAY_LIST")

ofertas_JU =lt.newList("ARRAY_LIST")

ofertas_totales = lt.newList("ARRAY_LIST")

for llave_empresa in lt.iterator(empresas):

    ofertas_senior = me.getValue(mp.get(me.getValue(mp.get(ciudad, llave_empresa)), "senior"))

    ofertas_mid = me.getValue(mp.get(me.getValue(mp.get(ciudad, llave_empresa)), "mid"))

    ofertas_junior = me.getValue(mp.get(me.getValue(mp.get(ciudad, llave_empresa)), "junior"))

    lista_provisional1 = lt.newList("ARRAY_LIST")

    lista_provisional2 = lt.newList("ARRAY_LIST")

    lista_provisional3 = lt.newList("ARRAY_LIST")

    if not om.isEmpty(ofertas_senior):

```

```

        #senior = om.values(ofertas_senior,
ofertas_senior["root"]["key"], "2025-12-24")

        senior = om.valueSet(ofertas_senior)

        for nivel in lt.iterator(senior):

            lt.addLast(lista_provisional1, om.valueSet(nivel))

        for lista in lt.iterator(lista_provisional1):

            lt.addLast(ofertasSN, lista)


    if not om.isEmpty(ofertas_mid):

        #mid = om.values(ofertas_mid, ofertas_mid["root"]["key"],
"2025-12-24")

        mid = om.valueSet(ofertas_mid)

        for nivel in lt.iterator(mid):

            lt.addLast(lista_provisional2, om.valueSet(nivel))

        for lista in lt.iterator(lista_provisional2):

            lt.addLast(ofertas_MD, lista)


    if not om.isEmpty(ofertas_junior):

        #junior = om.values(ofertas_junior,
ofertas_junior["root"]["key"], "2025-12-24")

        junior = om.valueSet(ofertas_junior)

        for nivel in lt.iterator(junior):

            lt.addLast(lista_provisional3, om.valueSet(nivel))

        for lista in lt.iterator(lista_provisional3):

            lt.addLast(ofertas_JU, lista)

```

```

conteo = 0

for valueset in lt.iterator(ofertasSN):
    for list in lt.iterator(valueset):
        for offer in lt.iterator(list):
            conteo += 1

            if offer["workplace_type"] == ubicacion:
                lt.addLast(ofertas_totales, offer)

for valueset in lt.iterator(ofertas_MD):
    for list in lt.iterator(valueset):
        for offer in lt.iterator(list):
            conteo += 1

            if offer["workplace_type"] == ubicacion:
                lt.addLast(ofertas_totales, offer)

for valueset in lt.iterator(ofertas_JU):
    for list in lt.iterator(valueset):
        for offer in lt.iterator(list):
            conteo +=1

            if offer["workplace_type"] == ubicacion:
                lt.addLast(ofertas_totales, offer)

print(F"CONTEOOOOOO: {conteo} ACÁÁÁÁÁÁÁÁ")

cantidad_ofertas = lt.size(ofertas_totales)

```

```

merg.sort(ofertas_totales, sort_crit_reciente_a_antiguo)

if N == None:

    return cantidad_ofertas , ofertas_totales, False

if lt.size(ofertas_totales) >= N:

    ofertas_totales = lt.subList(ofertas_totales, 1, N)

cinco = False

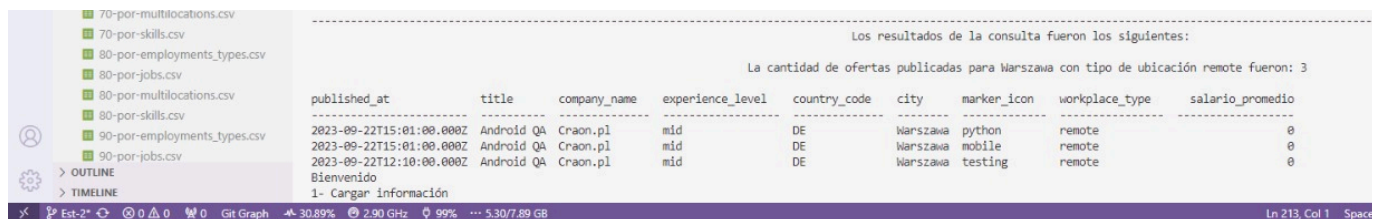
if lt.size(ofertas_totales)>10:

    cinco = True

return cantidad_ofertas , ofertas_totales, cinco

```

Esta función contribuyó al entendimiento de la estructura que hicimos y que quedó plasmada de forma gráfica al final de este documento. Entonces, este requerimiento básicamente lo que hace es que consulta las N ofertas laborales más recientes según una ciudad y tipo de ubicación. Primero consigo el keyset de la primera “capa” de la estructura ‘major\_structure’ y itero por las llaves hasta encontrar que el mapa de país en la llave de la iteración contiene la ciudad de interés, una vez la encuentra esta ciudad, la guarda en una variable y se sale de ese primer loop. Acá pasa algo interesante y es que por como está planteada ‘major\_structure’, detectó inconsistencias en los datos del csv como por ejemplo entradas del csv cuyo ‘country\_code’ es un string vacío, o por ejemplo entradas en las que las ciudades no concuerdan con su country\_code. A continuación un ejemplo:



Los resultados de la consulta fueron los siguientes:

La cantidad de ofertas publicadas para Warszawa con tipo de ubicación remote fueron: 3

published_at	title	company_name	experience_level	country_code	city	marker_icon	workplace_type	salario_promedio
2023-09-22T15:01:00.000Z	Android QA	Craon.pl	mid	DE	Warszawa	python	remote	0
2023-09-22T15:01:00.000Z	Android QA	Craon.pl	mid	DE	Warszawa	mobile	remote	0
2023-09-22T12:10:00.000Z	Android QA	Craon.pl	mid	DE	Warszawa	testing	remote	0

Bienvenido  
1- Cargar información

Luego, si no encontró nada, retorna False. Después, consigo el keyset de la ciudad que van a las los nombres de las empresas presentes en la ciudad e inicialice unas listas que van a contener las ofertas

senior, junior y mid y el total de ofertas. Luego, itere por las llaves empresas y accedí a las llaves “junior”, “mid” y “senior” que tiene el mapa contenido dentro de cada empresa. Como en este caso no tenía que filtrar por fecha como tal, consigo el valueset del árbol contenido en las llaves de experticia. A partir de acá llegar más profundo en la estructura se volvió complicado en un principio pues había que iterar por muchos valuesets, entonces para no contribuir a una complejidad mayor a la cuadrática, se iban sacando los elementos de un ciclo afuera del mismo para volver a iterar en un nuevo ciclo. Volviendo al nivel de experticia, ahí quedaba preguntar que el árbol de salarios no estuviese vacío pues eso significaba que no habían ofertas en esa categoría. Si no estaba vacío se conseguía el valuset del árbol que ya eran las listas de ofertas per se, y se añadían a una lista aparte del ciclo. Actos seguido se acaba el ciclo y se comienzan a separar las ofertas para meterlas en una lista y de este modo queda no una lista de listas sino una lista final de ofertas. Una vez se tienen las ofertas, se hace mergesort con un criterio de ordenamiento de fecha más reciente a más antigua y se devuelve la sublista de N ofertas.

<b>Entrada</b>	Datastructs (el model), N, nombre_ciudad, y tipo de ubicación
<b>Salidas</b>	Cantidad de ofertas totales de la consulta, N ofertas, un parámetro cinco:bool que dice si en el view se imprimen las cinco primeras y últimas ofertas o no.
<b>Implementado (Sí/No)</b>	Sí. Implementado por Juan José Cortés Villamil

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
mp.keySet()	O(N) en el peor caso, O(1) en el mejor caso
iterar por el keyset de datastructus["major_structure"]	O(N)
<code>mp.contains(me.getValue(mp.get(data_structs["major_structure"], llave_pais)), nombre_ciudad)</code>	O(N/M)
<code>me.getValue(mp.get(me.getValue(mp.get(data_structs["major_structure"], llave_pais)), nombre_ciudad))</code>	O(N/M) peor caso
<code>empresas = mp.keySet(ciudad)</code>	O(N)
<code>ofertasSN = lt.newList("ARRAY_LIST")</code> <code>ofertas_MD =</code> <code>lt.newList("ARRAY_LIST")</code> <code>ofertas_JU</code> <code>=lt.newList("ARRAY_LIST")</code>	O(1)

<code>ofertas_totales = lt.newList("ARRAY_LIST")</code>	
Iterar por las llaves de cada empresa en la ciudad	O(N)
<code>ofertas_senior = me.getValue(mp.get(me.getValue(mp.get( ciudad, llave_empresa)), "senior")) ofertas_mid = me.getValue(mp.get(me.getValue(mp.get( ciudad, llave_empresa)), "mid")) ofertas_junior = me.getValue(mp.get(me.getValue(mp.get( ciudad, llave_empresa)), "junior"))</code>	O(3) en el peor caso si es que llegaron a quedar en el hash juntas cosa que es realmente poco probable. $\approx O(1)$
lt.newList("ARRAY_LIST") (creación de listas provisionales)	O(1)
om.isEmpty(ofertas senior V mid V junior)	O(1)
om.valueSet(ofertas senior V mid V junior)	O(N)
Iterar por los niveles que corresponde a los nodos del primer árbol de fechas	O(N)
<code>lt.addLast(lista_provisional 1 V 2 V 3, om.valueSet(nivel))</code>	$O(1) + O(N) \approx O(N)$
Iterar por los valueSets en la listaprovisional 1 V 2 V 3	O(N)
<code>lt.addLast(ofertas SN V JU V MD, lista)</code>	O(1)
Iterar por los valueSets en ofertasSN V ofertasMD V ofertasJN	O(N)
iterar por las listas de cada valueSet	Depende, si el anterior paso fue O(N) significa todas las ofertas fueron de alguna experticia específica, eso también significa entonces que en cada lista solo puede haber una oferta, porque si no habrían más de N ofertas, habrían $N^2$ ofertas, cosa que no es cierta, osea sería O(N)
iterar por las ofertas en cada lista	Como se mencionó anteriormente, en el peor caso, solo podría haber una una oferta en cada lista, entonces O(1)
Merge.sort()	O(NlogN)
lt.subList()	O(1)
<b>TOTAL</b>	<b><math>O(N^2)</math> después del análisis de los tres ciclos anidados.</b>

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron N = 20, Poznan y partly\_remote.

Procesadores	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10

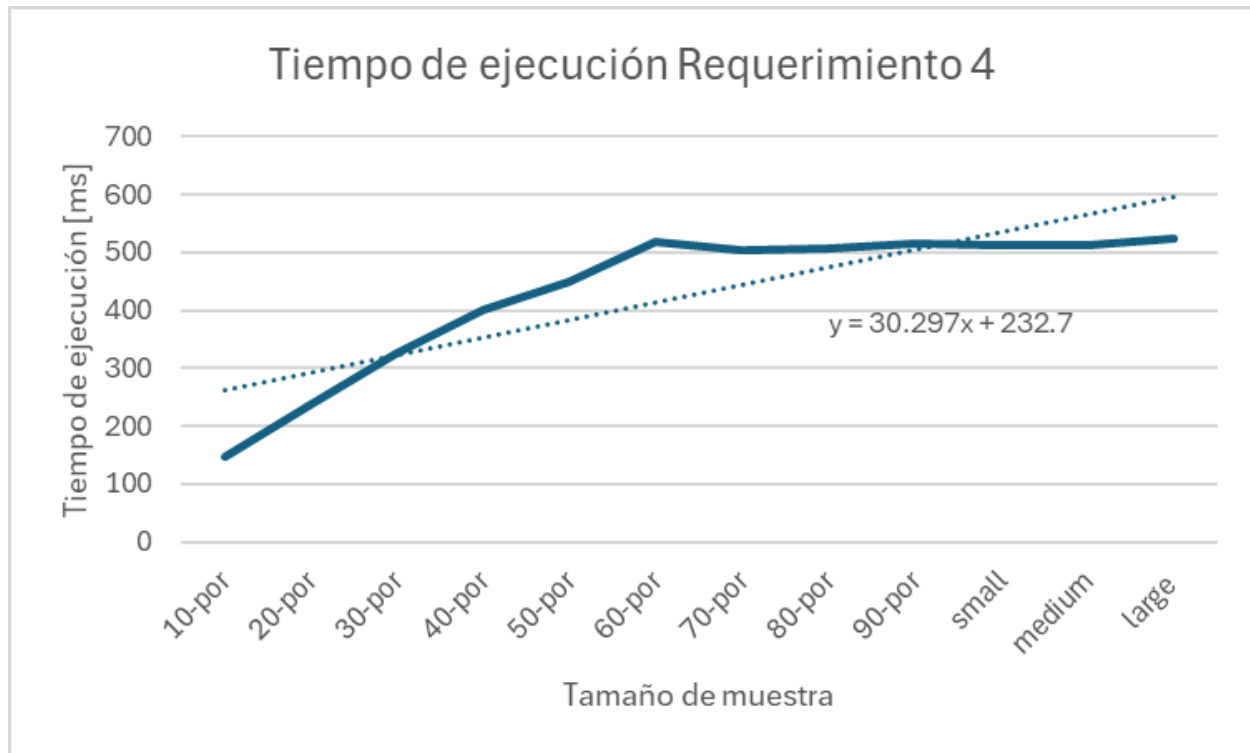
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	7493.292
20-por	16727.5668
30-por	37980.2812
40-por	33062.9439
50-por	41242.5086
60-por	46179.228
70-por	48055.5424
80-por	52528.6104
90-por	54441.5091
small	50412.6539
medium	50310.0012
large	49688.5049

## Gráficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

A partir de la gráfica observamos que hay un comportamiento semejante al logarítmico y no cuadrático o incluso cúbico como erróneamente se hubiese llegado a concebir. Este requerimiento a diferencia del uno, sí demuestra un comportamiento logarítmico porque usando siempre las mismas entradas,  $N = 20$ , Poznan y 'partly\_remote' aumenta el tiempo de ejecución pero mínimamente. Es un comportamiento logarítmico porque la cantidad de ofertas de Poznan aumentan en las diferentes muestras. También es cierto que si las ofertas de poznan fuesen las  $N$  ofertas quizás estaríamos ante un tiempo de ejecución mucho mayor y en general un comportamiento cercano al lineal. Pero de la teoría obtenemos que el peor caso de este algoritmo tiene una complejidad temporal cuadrática.

## Requerimiento <<5>>

```
def req_5(data_structs, numero_ofertas, tamano_minimo_compania,
          tamano_maximo_compania, skill, limite_inferior_skill,
          limite_superior_skill):
    """
    Función que soluciona el requerimiento 5
    """
```



```

# TODO: Realizar el requerimiento 5

ofertas_totales = lt.newList("ARRAY_LIST")

final = lt.newList("ARRAY_LIST")

cantidad_ofertas = 0

#Compañías que estan entre el rango de tamaños

lista_nombres_companias =
om.values(data_structs["size_empresas"],tamano_minimo_compania,tamano_maxi
mo_compania)

#Lista de ciudades

lista_ciudades = mp.valueSet(data_structs["major_structure"])

lista_empresas_filtradas = lt.newList("ARRAY_LIST")

lista_nueva =lt.newList("ARRAY_LIST")

ofertasSN = lt.newList("ARRAY_LIST")

ofertas_MD = lt.newList("ARRAY_LIST")

ofertas_JU =lt.newList("ARRAY_LIST")

#####

#####

for ciudad in lt.iterator(lista_ciudades):

    lista_ciudades_por_valueset = mp.valueSet(ciudad)

    lt.addLast(lista_nueva, lista_ciudades_por_valueset)

```

```
lista_hi = lt.newList("ARRAY_LIST")
```

```
for df in lt.iterator(lista_nueva):
```

```
    for hi in lt.iterator(df):
```

```
        lt.addLast(lista_hi, hi)
```

```
#####  
#####
```

```
lista_empresas_para_usar = lt.newList("ARRAY_LIST")
```

```
#print(data_structs["size_empresas"])
```

```
for companies in lt.iterator(lista_nombres_companias):
```

```
    for companies2 in lt.iterator(companies):
```

```
        lt.addLast(lista_empresas_para_usar, companies2)
```

```
#####  
#####
```

```
for ciudades_c in lt.iterator(lista_hi):
```

```
    #print(ciudades_c)
```

```

for ofertas_que_son in lt.iterator(lista_empresas_para_usar):

    xt = mp.get(ciudades_c, ofertas_que_son)

    if xt !=None:

        off = me.getValue(mp.get(ciudades_c, ofertas_que_son))

        ofertas_senior = me.getValue(mp.get(off,"senior"))

        ofertas_mid = me.getValue(mp.get(off,"mid"))

        ofertas_junior = me.getValue(mp.get(off,"junior"))


        lista_provisional1 = lt.newList("ARRAY_LIST")

        lista_provisional2 = lt.newList("ARRAY_LIST")

        lista_provisional3 = lt.newList("ARRAY_LIST")

        if not om.isEmpty(ofertas_senior):

            #senior = om.values(ofertas_senior,
om.maxKey(ofertas_senior), ofertas_senior["root"]["key"])

            senior = om.valueSet(ofertas_senior)


            for nivel in lt.iterator(senior):

                lt.addLast(lista_provisional1, om.valueSet(nivel))

                for lista in lt.iterator(lista_provisional1):

                    lt.addLast(ofertasSN, lista)


            """

            salaries = om.values(nivel, om.maxKey(nivel),
nivel["root"]["key"] )

```

```

        ubicaciones = om.values(salaries,
mp.valueSet(salaries), salaries["root"]["key"])

        offers = me.getValue(mp.get(ubicaciones))

        for oferta in lt.iterator(offers):

            if skill in oferta["habilidades solicitadas"]:

                if tamano_minimo_compania <=
oferta["promedio_habilidad"] and oferta["habilidades solicitadas"] <=
tamano_maximo_compania:

                    lt.addLast(ofertas_totales, oferta)

                    cantidad_ofertas +=1

        """

    if not om.isEmpty(ofertas_mid):

        mid = om.valueSet(ofertas_mid)

        for nivel in lt.iterator(mid):

            lt.addLast(lista_provisional2, om.valueSet(nivel))

            for lista in lt.iterator(lista_provisional2):

                lt.addLast(ofertas_MD, lista)

        """

        mid = om.values(ofertas_senior,
om.maxKey(ofertas_mid), ofertas_mid["root"]["key"])

        for nivell in lt.iterator(senior):

```

```

        salaries1 = om.values(nivel1, om.maxKey(nivel1),
nivel1["root"]["key"] )

        ubicaciones1 = om.values(salaries1,
mp.valueSet(salaries1), salaries1["root"]["key"])

        offers1 = me.getValue(mp.get(ubicaciones1))

        for ofertal in lt.iterator(offers1):

            if skill in ofertal["habilidades
solicitudes"]):

                if tamano_minimo_compania <=
ofertal["promedio_habilidad"] and ofertal["habilidades solicitudes"] <=
tamano_maximo_compania:

                    lt.addLast(ofertas_totales, ofertal)

                    cantidad_ofertas +=1

        """

    if not om.isEmpty(ofertas_junior):

        junior = om.valueSet(ofertas_junior)

        for nivel in lt.iterator(junior):

            lt.addLast(lista_provisional3, om.valueSet(nivel))

            for lista in lt.iterator(lista_provisional3):

                lt.addLast(ofertas_JU, lista)

        """

        junior = om.values(ofertas_senior,
om.maxKey(ofertas_mid), ofertas_junior["root"]["key"])

```

```

        for nivel2 in lt.iterator(senior):

            salaries2 = om.values(nivel2, om.maxKey(nivel2),
nivel2["root"]["key"] )

            ubicaciones2 = om.values(salaries2,
mp.valueSet(salaries2), salaries2["root"]["key"])

            offers2 = me.getValue(mp.get(ubicaciones2))

            for oferta2 in lt.iterator(offers2):

                if skill in oferta2["habilidades
solicitudes"]]:

                    if tamano_minimo_compania <=
oferta2["promedio_habilidad"] and oferta2["habilidades solicitudes"] <=
tamano_maximo_compania:

                        lt.addLast(ofertas_totales, oferta2)

                        cantidad_ofertas +=1

            """

for valueset in lt.iterator(ofertasSN):

    for list in lt.iterator(valueset):

        for offer in lt.iterator(list):

            if skill in offer["habilidades_solicitudes"]["elements"]:

                if limite_inferior_skill <=
offer["promedio_habilidad"] and offer["promedio_habilidad"]<=
limite_superior_skill:

                    lt.addLast(ofertas_totales, offer)

for valueset1 in lt.iterator(ofertas_MD):

```

```
for list1 in lt.iterator(valueset1):  
    for offer1 in lt.iterator(list1):  
  
        if skill in offer1["habilidades_solicitadas"]["elements"]:  
  
            if limite_inferior_skill <=  
offer1["promedio_habilidad"] and offer1["promedio_habilidad"] <=  
limite_superior_skill:  
  
                lt.addLast(ofertas_totales, offer)  
  
for valueset2 in lt.iterator(ofertas_JU):  
    for list2 in lt.iterator(valueset2):  
        for offer2 in lt.iterator(list2):  
  
            if skill in offer2["habilidades_solicitadas"]["elements"]:  
  
                if limite_inferior_skill <=  
offer2["promedio_habilidad"] and offer2["promedio_habilidad"] <=  
limite_superior_skill:  
  
                    lt.addLast(ofertas_totales, offer)
```

```
cantidad_ofertas = lt.size(ofertas_totales)

if ofertas_totales !=None:

    merg.sort(ofertas_totales, sort_crit_reciente_a_antiguo)

if lt.size(ofertas_totales) >= 10:

    sb1 = lt.subList(ofertas_totales, 1, 5)

    for zc in lt.iterator(sb1):

        lt.addLast(final, zc)

    sb2 = lt.subList(ofertas_totales, (lt.size(ofertas_totales)-5), 5)

    for zt in lt.iterator(sb2):

        lt.addLast(final, zt )

elif lt.size(ofertas_totales)<10 and lt.size(ofertas_totales)>0:

    if lt.size(ofertas_totales)>= int(ofertas_totales):

        final = lt.subList(ofertas_totales, 1, int(numero_ofertas))

    else:

        final = ofertas_totales

elif lt.size(ofertas_totales)==0:

    final = None
```



```
return cantidad_ofertas, final
```

## Descripción

Este requerimiento consiste en mostrar N ofertas laborales dado un rango de tamaños de compañías, una skill y un rango de nivel de la skill. Dentro del requerimiento empiezo inicializando varios arreglos, una lista con los nombres de las compañías que cumplen con el parámetro de tamaños de compañías, ya que tenemos una estructura que es destinada para eso, tambien empiezo sacando el valueset de nuestra estructura grande, el cual son los valores de los paises, osea las ciudades. después empiezo a iterar por las “ciudades” para acceder hasta las ofertas por nivel de experticia senior, mid y junior, metiéndolas en otro arreglo e iterando nuevamente. Después organizo bien los valores de las empresas que cumplen con el requisito de tamaño. Empiezo a iterar por las ofertas y las compañías y saco las ofertas por nivel de experticia, las guardo en un arreglo y después las paso a otro arreglo llamado lista provisional y finalmente a otro arreglo que está por nivel de experticia. Finalmente hago los filtros ya que pudimos llegar a la oferta neta en sí, en donde puedo acceder a sus llaves y preguntar por las habilidades solicitadas que tiene cada oferta y el promedio de la habilidad, si cumple con los filtros pasa a el arreglo “ofertas totales”. Al final sacó la sublista dependiendo del número de ofertas que se hayan guardado al igual que el requerimiento 2.

<b>Entrada</b>	Estructuras de datos del modelo, el número de ofertas a imprimir, el tamaño mínimo de la compañía, el tamaño máximo de la compañía, la habilidad solicitada, el límite inferior del nivel de la habilidad, el límite superior del nivel de la habilidad.
<b>Salidas</b>	Las ofertas que cumplan con esos filtros.
<b>Implementado (Sí/No)</b>	Si. Implementado por Lucas Valbuena Leon

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1: Creación nuevos arreglos	$O(1)$
Paso 2 : función values	$O(2 \log N)$
Paso 3:funcion valueset	$O(2 \log N)$
Paso 4: iterar por el valueset(las ciudades)	$O(N)$
Paso 5: iterar por las compañías	$O(M)$ M siendo el número de compañías que pasaron el filtro de tamaño
Paso 6: iterar por ciudades y compañías anidadas	$O(M*N)$
Paso 7: iterar por las listas que armamos en el ciclo anterior	$O(M)$

Paso 8: Guardar ofertas que pasan por los filtros	O(M)
Paso 9: Sacar sublista	O(1)
<b>TOTAL</b>	O(N)

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Input:

Número de ofertas a imprimir: 8

tamaño mínimo compañía: 20

tamaño máximo compañía: 120

habilidad solicitada : HTML

nivel mínimo habilidad: 0

nivel máximo habilidad: 5

Procesadores	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz
Memoria RAM	16,0 GB
Sistema Operativo	Windows 11 Pro

## Tablas de datos

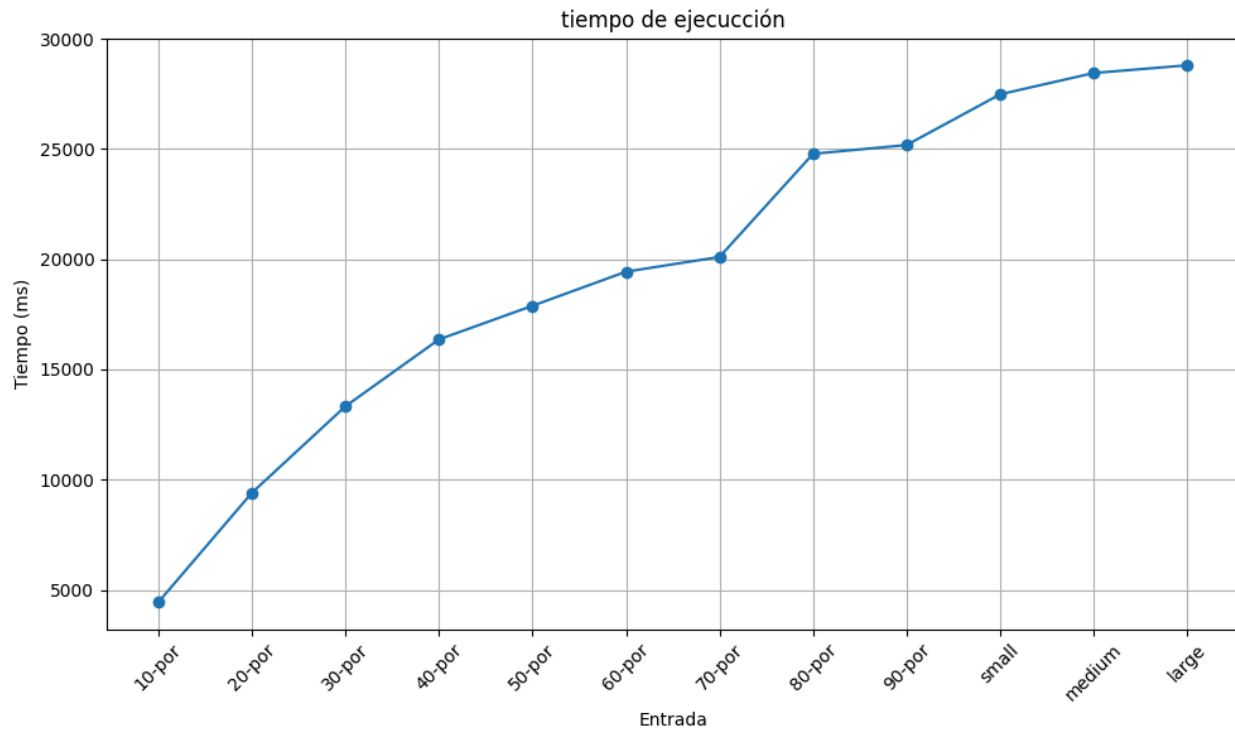
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	4443.417
20-por	9404.170
30-por	13329.059
40-por	16363.515
50-por	17889.320
60-por	19436.529
70-por	20105.423
80-por	24783.598
90-por	25183.083
small	27483.763
medium	28449.002

large	28799.788
-------	-----------

## Gráficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

Podemos percibir una tendencia tanto lineal como logarítmica en los tiempos de ejecución de este requerimiento, lo que nos lleva a pensar que las entradas que se utilizaron para hacer estas pruebas pudieron ser un caso promedio de este requerimiento o un caso un poco mejor que el promedio. De la entrada "70-por" a 80-por" podemos ver un salto en la ejecución, pudiendo ser una diferencia sustancial en cuanto a los resultados, como por ejemplo que se encuentren muchas mas ofertas que cmlpan con los requisitos, y lo que implicaría mayores recorridos a la estructura principal y estructuras auxiliares.

## Requerimiento <<6>>

```
def req_6(data_structs,N,fecha1,fecha2,salario1,salario2):  
  
    """  
  
    Función que soluciona el requerimiento 6  
  
    """  
  
    # TODO: Realizar el requerimiento 6  
  
    lista_fechas=lt.newList("ARRAY_LIST")  
  
    mapa_salarios=mp.newMap(numelements=203560,maptype="CHAINING",  
loadfactor= 4)  
  
    lista_total= lt.newList("ARRAY_LIST")  
  
    jobs= data_structs["jobs"]  
  
    salarios=data_structs["salaries_offers"]  
  
    listas_ofertas_fechas=om.values(jobs,fecha1,fecha2)  
  
    listas_ofertas_salarios=om.values(salarios,salario1,salario2)  
  
    for lista_offer in lt.iterator(listas_ofertas_fechas):  
        for offer in lt.iterator(lista_offer):  
            lt.addLast(lista_fechas,offer)  
  
  
    for lista_ofertas in lt.iterator(listas_ofertas_salarios):  
        for oferta in lt.iterator(lista_ofertas):  
            salario= oferta["salario_minimo"]  
  
            if mp.contains(mapa_salarios,salario) == False:
```

```

        valor=lt.newList("ARRAY_LIST")

        lt.addLast(valor,oferta)

        mp.put(mapa_salarios,salario,valor)

    else:

        pareja=mp.get(mapa_salarios,salario)

        valorC=me.getValue(pareja)

        lt.addLast(valorC,oferta)

for element1 in lt.iterator(lista_fechas):

    salario_of= element1["salario_minimo"]

    if mp.contains(mapa_salarios,salario_of):

        lt.addLast(lista_total,element1)

tamaño_total= lt.size(lista_total)

ciudad,tamaño,lista=mas_ciudades(lista_total,N)

lista_ofertas_ciudad=lt.newList("ARRAY_LIST")

for oferta in lt.iterator(lista_total):

    if oferta["city"]== ciudad:

        lt.addLast(lista_ofertas_ciudad,oferta)

return

tamaño,lista,merg.sort(lista_ofertas_ciudad,sort_crit_req3),tamaño_total

def mas_ciudades(lista,N):

    lista_ciudades=[]

    for oferta in lt.iterator(lista):

        lista_ciudades.append(oferta["city"])

```

```

contador_ciudades=Counter(lista_ciudades)

ciudades_mas_ofertas = contador_ciudades.most_common(N)

ciudad_mas_ofertas = contador_ciudades.most_common(1)[0][0]

size=len(contador_ciudades)

return ciudad_mas_ofertas,size,ciudades_mas_ofertas

```

## Descripción

El requerimiento filtra ofertas publicadas en un rango de fechas especificadas por el usuario, así como por un rango de salarios determinados. Primero, se toman los valores correspondientes a los rangos de fechas y salarios para los mapas “jobs” y “salarios” respectivamente. Una vez hecho esto, se itera únicamente para los valores de jobs, y se agrega cada oferta a una lista de fechas. Se hace algo similar con los valores del mapa salarios, pero se agregan a un mapa de salarios cuya llave es “salario mínimo” de cada oferta. Luego, se itera sobre cada oferta en lista\_fechas y si el salario mínimo de esa oferta se encuentra en el mapa de salarios, se agregara a una lista que contendrá las ofertas que nos interesan. Finalmente, se llama la función mas\_ciudades() donde con las funciones counter() y most\_common() se obtienen las N ciudades con más ofertas publicadas y la cantidad de ciudades que publicaron ofertas.

<b>Entrada</b>	data structs, fecha1, fecha2, salario 1, salario 2
<b>Salidas</b>	N ciudades con más ofertas publicadas, ofertas de la ciudad con mayor cantidad de ofertas, número total de ofertas publicadas que cumplan con la especificaciones
<b>Implementado (Sí/No)</b>	Si. Implementado por Alisson Moreno

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
<code>om.values(jobs, fecha1, fecha2)</code>	$O(2 \log N)$
<code>om.values(salarios, salario1, salario2)</code>	$O(2 \log N)$
iterar sobre los valores de cada mapa y agregar ofertas a la lista	$O(N*M)$ donde M es la cantidad de llaves en el mapa

<code>mp.contains(mapa_salarios, salario)</code>	O(M) donde M es la cantidad de llaves en el mapa
funcion mas_ciudades( )	O(N)
merg.sort( )	O( N Log N)
<b>TOTAL</b>	O(N^2)

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron N = 5, 2023-01-01, 2023-01-02, 1000, 1500

Procesadores	AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10

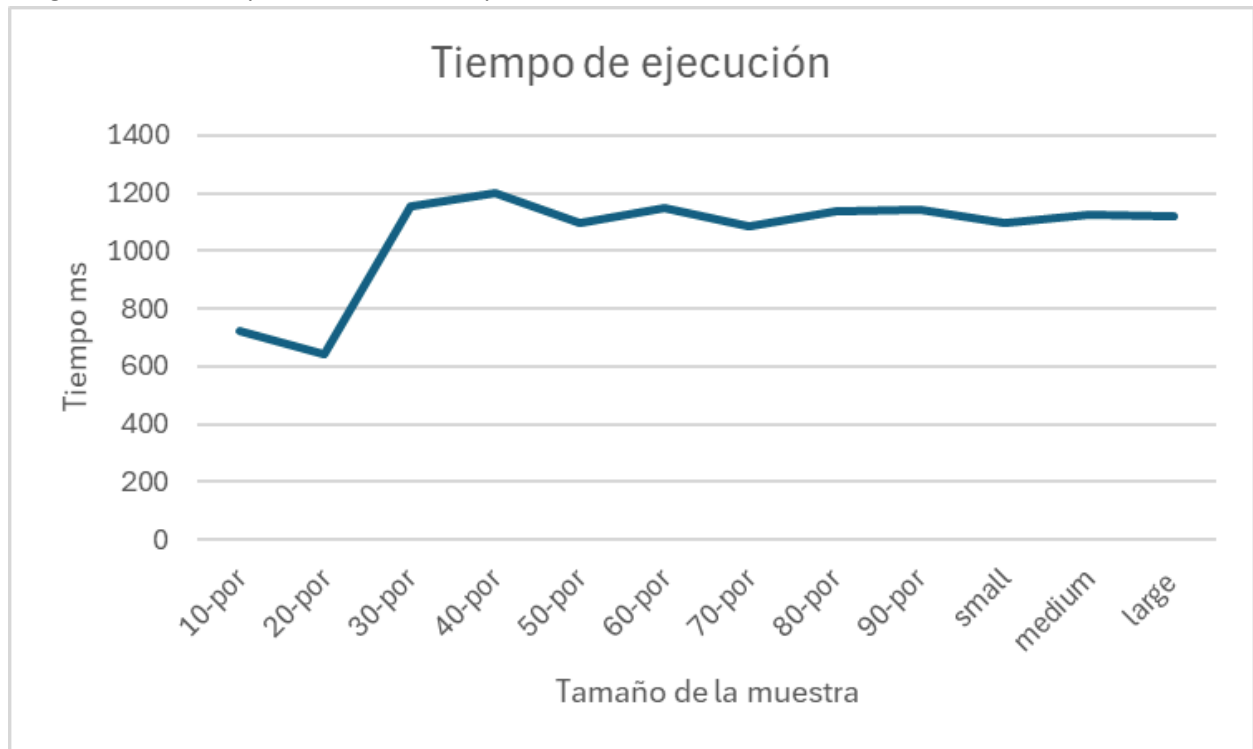
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	724.40
20-por	642.44
30-por	1157.33
40-por	1199.09
50-por	1094.65
60-por	1150.11
70-por	1085.59
80-por	1137.64
90-por	1141.45
small	1099.87
medium	1125.36
large	1119.40

## Gráficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

Al observar el comportamiento en términos de tiempo de la función del requerimiento 6 frente a la cantidad de datos cargados, se puede apreciar que tal como se esperaba a medida que se aumenta la cantidad de datos, le toma más tiempo poder cumplir con su función. Sin embargo, al solo tener que listar N ciudades, más no N ofertas, se reduce en un alto grado el orden de crecimiento temporal, ya que el número de ciudades es mucho menor que el número de ofertas publicadas. Además, el uso de mapas ordenados para fechas y salarios, facilitó el desarrollo del código y poder llevar a cabo con éxito el requerimiento solicitado, permitiendo obtener los valores asociados a un rango de llaves en una sola operación. En la gráfica, se puede observar que en un momento la complejidad se estabiliza, y esto puede deberse a que los datos con los que se probó, son rangos de fechas y salarios muy cortos entre sí, dejando un margen muy pequeño entre las ofertas de una muestra de datos y otra.

## Requerimiento <<7>>

```
def req_7(data_structs, año, codigo_pais, propiedad_conteo, bins):
```



```

"""
Función que soluciona el requerimiento 7
"""

# TODO: Realizar el requerimiento 7

pais = me.getValue(mp.get(data_structs["major_structure"],
codigo_pais))

ciudades = lt.newList("ARRAY_LIST")

for llave_ciudad in lt.iterator(mp.keySet(pais)):

    lt.addLast(ciudades, me.getValue(mp.get(pais, llave_ciudad)))

empresas = lt.newList("ARRAY_LIST")

for ciudad in lt.iterator(ciudades):

    for empresa in lt.iterator(mp.valueSet(ciudad)):

        lt.addLast(empresas, empresa)

        # ofertas = om.values(ofertas_senior,
ofertas_senior["root"]["key"], "2023-12-24")

valueSetsJN = lt.newList("ARRAY_LIST")

valueSetsMI = lt.newList("ARRAY_LIST")

valueSetsSN = lt.newList("ARRAY_LIST")

for empresa in lt.iterator(empresas):

    ofertas_senior = om.values(me.getValue(mp.get(empresa, "senior")),
f"{año}-01-1", f"{año}-12-31")

```

```
ofertas_mid = om.values(me.getValue(mp.get(empresa, "mid")),
f"{año}-01-1", f"{año}-12-31")

ofertas_junior = om.values(me.getValue(mp.get(empresa, "junior")),
f"{año}-01-1", f"{año}-12-31")

lista_provisional1 = lt.newList("ARRAY_LIST")
lista_provisional2 = lt.newList("ARRAY_LIST")
lista_provisional3 = lt.newList("ARRAY_LIST")
if not om.isEmpty(ofertas_senior):

    for nivel in lt.iterator(ofertas_senior):
        lt.addLast(lista_provisional1, om.valueSet(nivel))
    for lista in lt.iterator(lista_provisional1):
        lt.addLast(valueSetsSN, lista)

if not om.isEmpty(ofertas_mid):

    for nivel in lt.iterator(ofertas_mid):
        lt.addLast(lista_provisional2, om.valueSet(nivel))
    for lista in lt.iterator(lista_provisional2):
        lt.addLast(valueSetsMI, lista)

if not om.isEmpty(ofertas_junior):

    for nivel in lt.iterator(ofertas_junior):
```

```

        lt.addLast(lista_provisional3, om.valueSet(nivel))

    for lista in lt.iterator(lista_provisional3):

        lt.addLast(valueSetsJN, lista)

ofertas_totales = lt.newList("ARRAY_LIST")
data_experticia = []
data_ubicacion = []
data_habilidad = []

listsSN = lt.newList("ARRAY_LIST")
listsMD = lt.newList("ARRAY_LIST")
listsJN = lt.newList("ARRAY_LIST")

overview = mp.newMap(numelements=13, maptype="PROBING",
loadfactor=0.5)

mp.put(overview, "senior", 0)
mp.put(overview, "mid", 0)
mp.put(overview, "junior", 0)
mp.put(overview, "office", 0)
mp.put(overview, "remote", 0)
mp.put(overview, "partly_remote", 0)

overview_skills = mp.newMap(numelements=100, maptype="PROBING",
loadfactor=0.5)

for valueset in lt.iterator(valueSetsSN):

```

```
for list in lt.iterator(valueset):

    lt.addLast(listsSN, list)

for list in lt.iterator(listsSN):

    for offer in lt.iterator(list):

        lt.addLast(ofertas_totales, offer)

        data_experticia.append("senior")

        data_ubicacion.append(offer["workplace_type"])

        mp.put(overview, "senior", me.getValue(mp.get(overview,
"senior"))+1)

        mp.put(overview, offer["workplace_type"],
me.getValue(mp.get(overview, offer["workplace_type"]))+1)

for valueset in lt.iterator(valueSetsMI):

    for list in lt.iterator(valueset):

        lt.addLast(listsMD, list)

for list in lt.iterator(listsMD):

    for offer in lt.iterator(list):

        lt.addLast(ofertas_totales, offer)

        data_experticia.append("mid")

        data_ubicacion.append(offer["workplace_type"])

        mp.put(overview, "mid", me.getValue(mp.get(overview,
"mid"))+1)

        mp.put(overview, offer["workplace_type"],
me.getValue(mp.get(overview, offer["workplace_type"]))+1)
```

```

for valueset in lt.iterator(valueSetsJN):
    for list in lt.iterator(valueset):
        lt.addLast(listsJN, list)

for list in lt.iterator(listsJN):
    for offer in lt.iterator(list):
        lt.addLast(ofertas_totales, offer)

        data_experticia.append("junior")

        data_ubicacion.append(offer["workplace_type"])

        mp.put(overview, "junior", me.getValue(mp.get(overview,
"junior"))+1)

        mp.put(overview, offer["workplace_type"],
me.getValue(mp.get(overview, offer["workplace_type"]))+1)

for offer in lt.iterator(ofertas_totales):
    for skill in lt.iterator(offer["habilidades_solicitadas"]):
        if not mp.contains(overview_skills, skill):
            mp.put(overview_skills, skill,-1)

            mp.put(overview_skills, skill,
me.getValue(mp.get(overview_skills, skill))+1)

            data_habilidad.append(skill)

minimo_experticia = me.getValue(mp.get(overview, "junior"))
minimo_ubicacion = me.getValue(mp.get(overview, "remote"))
minimo_skill = None
maximo_experticia = 0
maximo_ubicacion = 0
maximo_skill = 0

```

```

for key_skill in lt.iterator(mp.keySet(overview_skills)):

    if me.getValue(mp.get(overview_skills, key_skill)) > maximo_skill:

        maximo_skill = me.getValue(mp.get(overview_skills, key_skill))

    if minimo_skill == None:

        minimo_skill = me.getValue(mp.get(overview_skills, key_skill))

    else:

        if me.getValue(mp.get(overview_skills, key_skill)) <
minimo_skill:

            minimo_skill = me.getValue(mp.get(overview_skills,
key_skill))

for key in lt.iterator(mp.keySet(overview)):

    if (key == "senior") or (key == "junior") or (key == "mid"):

        if me.getValue(mp.get(overview, key)) > maximo_experticia:

            maximo_experticia = me.getValue(mp.get(overview, key))

        elif me.getValue(mp.get(overview, key)) < minimo_experticia:

            minimo_experticia = me.getValue(mp.get(overview, key))

    if (key == "office") or (key == "remote") or (key ==
"partly_remote"):

        if me.getValue(mp.get(overview, key)) > maximo_ubicacion:

            maximo_ubicacion = me.getValue(mp.get(overview, key))

        elif me.getValue(mp.get(overview, key)) < minimo_ubicacion:

            minimo_ubicacion = me.getValue(mp.get(overview, key))

if propiedad_conteo == "Experticia":

```

```

        return data_experticia, data_ubicacion, data_habilidad,
ofertas_totales, maximo_experticia, minimo_experticia

    elif propiedad_conteo == "Ubicacion":

        return data_experticia, data_ubicacion, data_habilidad,
ofertas_totales, maximo_ubicacion, minimo_ubicacion

    else:

        return data_experticia, data_ubicacion, data_habilidad,
ofertas_totales, maximo_skill, minimo_skill

```

## Descripción

El requerimiento solo se va a filtrar exclusivamente por año y país al principio ya que la propiedad de conteo puede cambiar. Vamos a tomar esas ofertas por país en el año pedido por el usuario y ya con esa ofertas se iteran metiendolas a varias listas según nivel de experticia y después a listas nativas para que puedan ser leídas por matplotlib. se retornan diferentes cosas para que matplotlib lo lea dependiendo de la propiedad de conteo dada por el usuario. Siempre va a devolver el “máximo” y el “mínimo” de la propiedad de conteo para generar el histograma.

<b>Entrada</b>	data structs, año de consulta, código de país, propiedad de conteo, número de bins.
<b>Salidas</b>	El histograma sobre la propiedad de conteo
<b>Implementado (Sí/No)</b>	Si. Implementado por Juan José Cortés

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1: sacar ciudades dado un codigo de pais	$O(2 \log N)$
Paso 2 : iterar por las ciudades dado un codigo de pais	$O(2 \log N)$
Paso 3: función get y get value	$O(N/M)$ peor caso
Paso 4: iterar por las empresas	$O(2 \log N)$
Paso 5: Crear nuevos arreglos auxiliares	$O(1)$
Paso 5: iterar por el niveles de experticia con función om.values	$O(2 \log N)$
Paso 7: Crear nuevos arreglos auxiliares	$O(1)$
Paso 8: Iterar sobre los arreglos auxiliares por nivel de experticia	$O(2 \log N)$

Paso 9: hacer maximos y minimos de las tres propiedades de conteo que se pueden elegir (funciones KeySet, GetValue y Get)	O(N) considerando que todas las ofertas hayan pasado hasta aquí y el keyset arroje O(N)
<b>TOTAL</b>	O(N)

### Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Input:

año de consulta: 2023

codigo de pais de consulta: PL

propiedad de conteo: ubicación

número de bins para dividir el histograma: 12

Procesadores	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz
Memoria RAM	16,0 GB
Sistema Operativo	Windows 11 Pro

### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	3991.156
20-por	8582.154
30-por	13188.176
40-por	17970.919
50-por	18742.778
60-por	20381.266
70-por	21338.228
80-por	22337.726
90-por	22820.424
small	24787.421
medium	26797.435
large	28563.544



## Graficas

Las gráficas con la representación de las pruebas realizadas.



## Análisis

En la gráfica de cantidad de datos cargados vs el tiempo de ejecución del requerimiento es posible observar una relación lineal entre ambas variables. Los datos que sirvieron como parámetros de entrada podrían ser la aproximación hacia el peor caso de la complejidad espacial, que sería  $O(N)$ . En cuanto a complejidad espacial, se tendría que calcular el promedio de datos divididos en todas las estructuras auxiliares manejadas para después sumarlos, pero sí sería menor a un caso  $O(N)$  a menos de que todas las ofertas cumplan con los requisitos, entonces tendríamos varias estructuras de tamaño  $O(N)$ , lo cual en la notación que estamos manejando queda como  $O(N)$ . La complejidad espacial podría ser un tanto menos costosa si optamos por reescribir algunas listas de tipo "ARRAY\_LIST" pero eso equivaldría a tener que realizar más operaciones y afectar nuestra complejidad temporal.

## Requerimiento <<8>>

## Descripción

Poner el código de todo el requerimiento 8 sería muy extenso, por esta misma razón solo voy a poner una parte del requerimiento ocho ya que las demás partes siguen la misma lógica.

```
def print_req_8(control):  
    """  
        Función que imprime la solución del Requerimiento 8 en consola  
    """  
  
    # TODO: Imprimir el resultado del requerimiento 8  
  
    print("POR FAVOR ESCOJA QUE REQUERIMIENTO DESEA VISUALIZAR EN EL MAPA  
INTERACTIVO... ")  
  
    req = int(input("Requerimiento... "))  
  
    if req == 1:  
  
        print("\n POR FAVOR DIGITE LOS SIGUIENTES DATOS PARA HACER  
EFECTIVA SU CONSULTA\n")  
  
        print("\nConsultar las ofertas que se publicaron durante un  
periodo de tiempo\n")  
  
        fecha0 = input("Digite el límite inferior del rango de fechas  
(Y-M-D): ")  
  
        fecha1 = input("Digite el límite superior del rango de fechas  
(Y-M-D): ")  
  
        correct0 =  
re.search(pattern=r"^[0-9]{4}-[0]{1}[0-9]{1}-[0-9]{2}$|^[0-9]{4}-[1]{1}[0-  
2]{1}-[0-9]{2}$", string= fecha0)  
  
        correct1 =  
re.search(pattern=r"^[0-9]{4}-[0]{1}[0-9]{1}-[0-9]{2}$|^[0-9]{4}-[1]{1}[0-  
2]{1}-[0-9]{2}$", string= fecha1)  
  
        if not correct0 or not correct1:
```

```

        print("-"*220)

        print(";Digito mal el formato de fecha, por favor intentelo
nuevamente!".center(220))

        print("-"*220)

        print_req_4(control)

        return None

    cantidad_ofertas, total_ofertas, cinco, prueba, nombre_prueba =
controller.req_1(control, fecha0, fecha1)

    ofertal = lt.getElement(total_ofertas, 1)

    m = folium.Map(location=(float(ofertal["latitude"]),
float(ofertal["longitude"])))

    for offer in lt.iterator(total_ofertas):

        folium.Marker(

            location=[float(offer["latitude"]),
float(offer["longitude"])],

            tooltip="Click me!",

            popup=f"Título: {offer['title']}\n compañía:
{offer['company_name']}, \n Nivel de experiencia:
{offer['experience_level']}",

            icon=folium.Icon(color="green"),

        ).add_to(m)

    m.show_in_browser()

```

En lo concerniente a esta implementación del requerimiento ocho como tal no tiene mayor misterio. El caso de la imagen de arriba es el requerimiento 1 en el requerimiento 8 es decir, hago un condicional para cada requerimiento porque desplegar un mapa interactivo de todos los requerimientos a la vez sería muy demandante para un computador portátil sin ayuda de una máquina virtual. Entonces el usuario digita qué requerimiento quiere visualizar en el mapa interactivo. En la imagen de arriba está el requerimiento uno, entonces simplemente se le pide al usuario que provea los datos necesarios para que el requerimiento corra. De este modo se llama la función en controller y como toda función

devuelve un listado de ofertas se procede a iterar por las ofertas y añadir un marcador de Folium por cada oferta en el mapa. Cabe resaltar que algunas ofertas tienen coordenadas prácticamente iguales por lo que se tapan entre sí. Por ejemplo en el requerimiento 7 como hay una condición de consulta, los marcadores se pintan en el caso de habilidades, de un color aleatorio, en el caso de experticia y ubicación, cada una de las tres posibilidades tiene un color específico.

<b>Entrada</b>	Como tal el requerimiento 8 no tiene ninguna entrada per se, pues cada requerimiento tiene sus entradas aparte.
<b>Salidas</b>	Un mapa interactivo que se despliega en el browser mostrando las ofertas de consulta de algún requerimiento específico.
<b>Implementado (Sí/No)</b>	Sí. Implementado por Juan José Cortés Villamil

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Requerimiento x	$O(2\log N)$ o $O(N^2)$ depende del requerimiento
lt.getElement() para obtener la primera oferta y inicializar el mapa desde esta	$O(1)$
Folium.map()	$O(1)$
Ciclo para iterar por las ofertas resultantes del requerimiento x	$O(N)$
folium.Marker().add_to(mapa)	$O(1)$
mapa.show_in_browser()	$O(1)$ En complejidad temporal es rápido, pero cuando son muchas ofertas pide muchas exigencias del equipo en el que se esté probando.
<b>TOTAL</b>	<b><math>O(N^2)</math></b>

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron 4000 y 5000 como rango salarial en el marco del requerimiento 2.

<b>Procesadores</b>	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
<b>Memoria RAM</b>	8.00 GB (7.88 GB usable)
<b>Sistema Operativo</b>	Windows 10

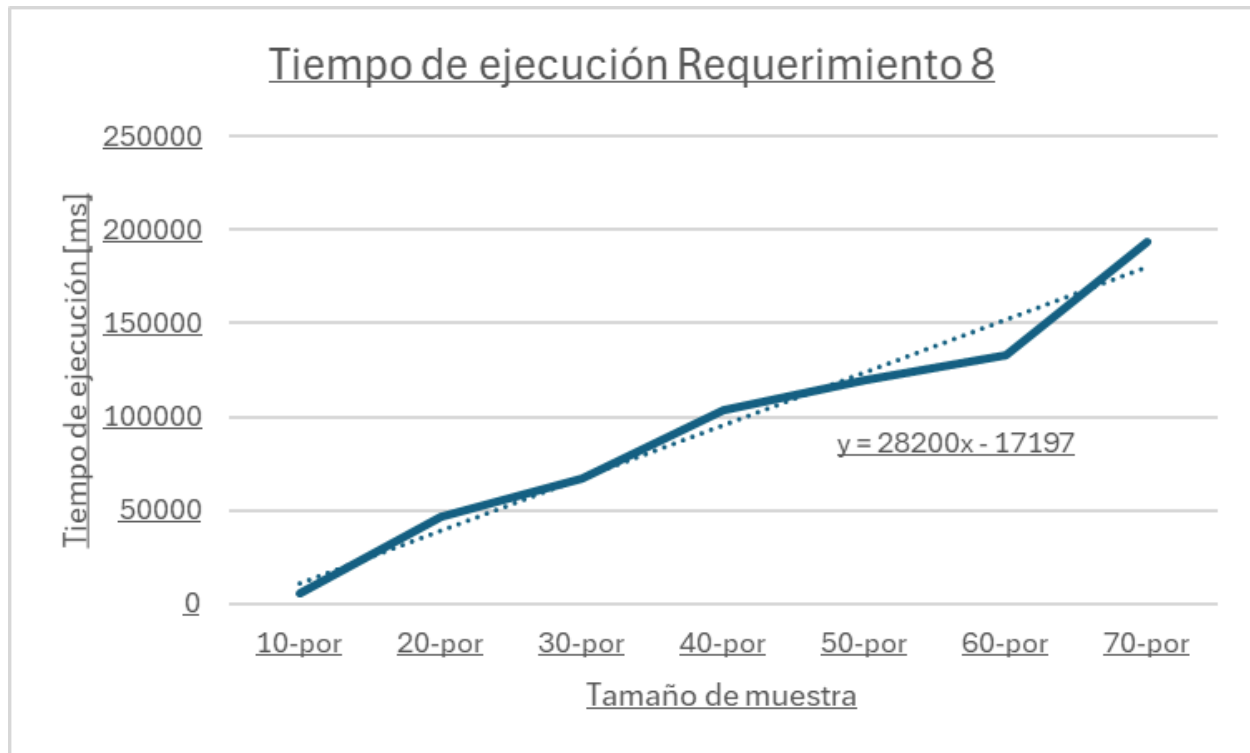
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	5426.7661
20-por	46947.5045
30-por	66805.0541
40-por	103641.9084
50-por	119902.0592
60-por	132816.5057
70-por	193681.1586

## Gráficas

Las gráficas con la representación de las pruebas realizadas.

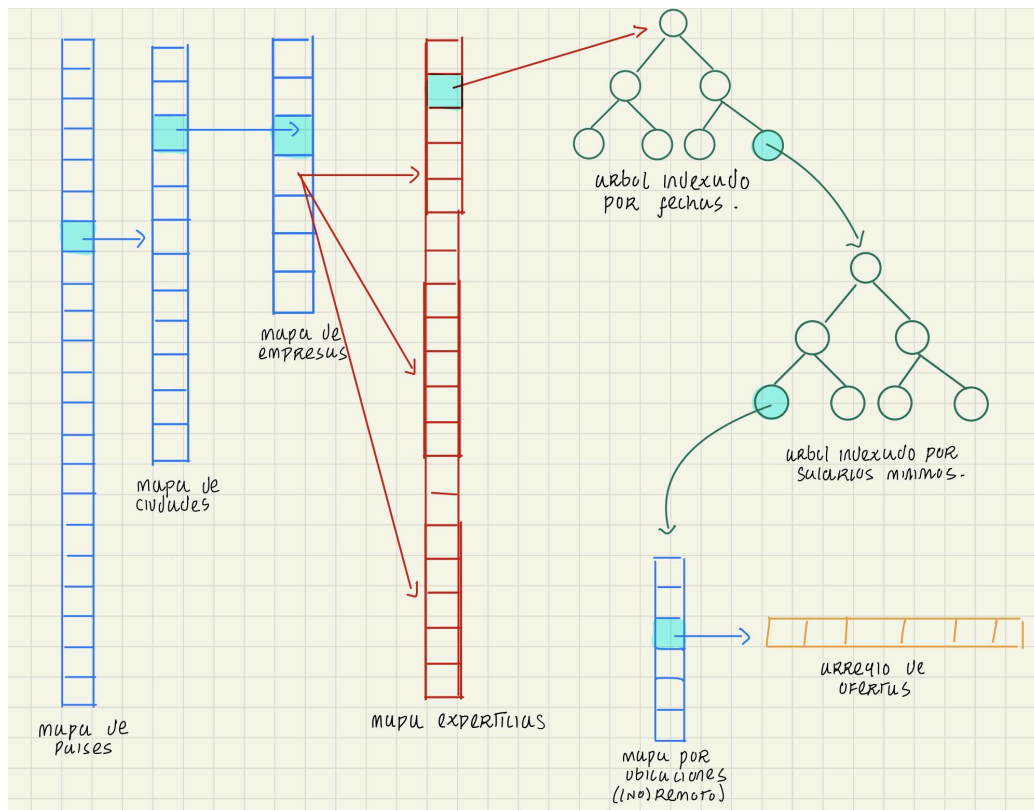


## Análisis

Los datos del req 8 los tomamos principalmente con base al requerimiento 2, es decir probamos el mapa interactivo del requerimiento 2 con diferentes muestras. Lo que encontré fue que a partir de 80-por, cargar el mapa de Folium exigía mucha memoria, esto derivó en que se me crasheara visual code en repetidas ocasiones al intentar cargar el mapa con 80-por, principalmente porque justo use un rango relativamente amplio que cada vez comprendía más ofertas. Usé el rango de salarios de 4000 a 5000. Ahora, como se ve tiene una tendencia lineal, pero me atrevería a decir que dependiendo del requerimiento que se escoja, el tiempo de ejecución va tener comportamientos diferentes, pues es la complejidad del requerimiento sumada a un ciclo es decir, más  $O(N)$ .

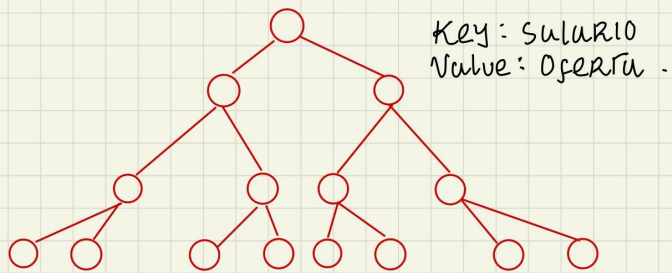
## Diagramas estructuras de datos

Estructura *major structure*:



Estructuras adicionales:

SALARIES Offers



SIZE empresas.

