ANÁLISIS DEL RETO

Nelson Felipe Celis – 202320636 – nf.celis@uniandes.edu.co
Felipe Boada – 202311982 – f.boada@uniandes.edu.co
Maria Alejandra Carrillo – 202321854 – ma.carrillo2@uniandes.edu.co

Requerimiento 1

Descripción

```
def req_1(data_structs, n, pais, exp):
    """
    Función que soluciona el requerimiento 1
    """
    paises=mp.get(data_structs['paises'], pais)
    lista_ofertas=me.getValue(paises)
    tamanio_pais=lt.size(lista_ofertas)
    filtrados=lt.newList('ARRAY_LIST')
    for oferta in lt.iterator(lista_ofertas):
        if oferta['experience_level']==exp:
            lt.addLast(filtrados, oferta)
    tamanio_exp=lt.size(filtrados)
    filtrados=sa.sort(filtrados, fecha)
    if lt.size(filtrados)>n:
        return lt.subList(filtrados, 1, n), tamanio_pais, tamanio_exp
    else:
        return filtrados, tamanio_pais, tamanio_exp
```

El requerimiento 1 se encarga de presentar un número de ofertas de trabajo de acuerdo con el número de ofertas que desee ver el usuario, el código del país, y el nivel de experticia. Si existen dichos datos, se retorna la lista de las ofertas que cumplan con los parámetros de entrada. De lo contrario se retorna None y en consola aparecerá un mensaje.

Entrada	Número de ofertas, país y nivel de experticia.
Salidas	Total de ofertas de trabajo para país y condición, lista de las ofertas.
Implementado (Sí/No)	Sí - Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener la pareja llave-valor para el país en un mapa (mp.get())	O(1)
Obtener el valor de la pareja (me.getValue())	O(1)
Obtener el tamaño de la lista de los países (lt.size())	O(1)

Crear una nueva lista *(lt.newList())*	O(1)
Iterar sobre la lista de ofertas para agregarlas a la lista nueva si cumplen con las condiciones *(lt.iterator())*	O(n)
Ordenar los datos filtrados de la lista (*sa.sort())*	O(n^1.5)
TOTAL	***O(n^1.5)***

Pruebas Realizadas

Procesadores	**Intel ® Core ™ i7 – 7500u CPU @ 2.70 GHz 2.90 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Pro

Datos constantes de entrada:

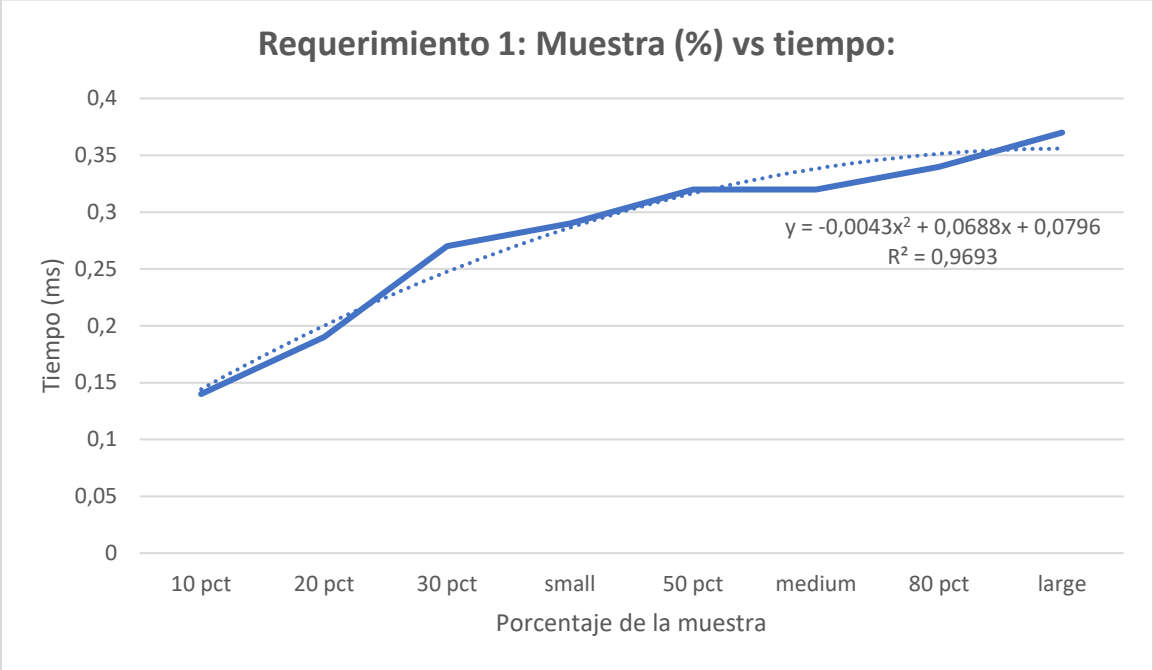
Número de ofertas: 9 País: ES Nivel de experticia: junior

Entrada	**Tiempo (ms)**
10 pct	0,14
20 pct	0,19
30 pct	0,27
small	0,29
50 pct	0,32
medium	0,32
80 pct	0,34
large	0,37

Tablas de datos

Muestra	**Salida**	**Tiempo (ms)**
10 pct	Ofertas país: 25 - Ofertas experiencia: 0 - Lista: Vacía	0,14
20 pct	Ofertas país: 66 - Ofertas experiencia: 0 - Lista: Vacía	0,19
30 pct	Ofertas país: 126 - Ofertas experiencia: 2 - Lista: 2 datos	0,27
small	Ofertas país: 145 - Ofertas experiencia: 4 - Lista: 4 datos	0,29
50 pct	Ofertas país: 206 - Ofertas experiencia: 4 - Lista: 4 datos	0,32
medium	Ofertas país: 234 - Ofertas experiencia: 5 - Lista: 5 datos	0,32
80 pct	Ofertas país: 240 - Ofertas experiencia: 6 - Lista: 6 datos	0,34
large	Ofertas país: 251 - Ofertas experiencia: 6 - Lista: 6 datos	0,37

Gráfica



Análisis

A partir de la gráfica, se puede observar cómo a medida que el porcentaje de la muestra aumenta, el tiempo de carga aumenta con una tendencia entre lineal y cuadrática. La línea de tendencia escogida no pudo ser de forma x^1.5, así que se aproximó a x^2 por las limitaciones de Word y sus gráficas. Teniendo en cuenta que la complejidad del requerimiento es de O(n^1.5) debido a el ordenamiento de los datos que se hace al final del programa, esto es en gran medida acorde con lo que muestra la gráfica. Además, hay que tener en consideración que a lo largo de la implementación de la función se utilizan otras funciones con complejidades de O(n) en el peor de los casos. Aquello puede influenciar en los resultados esperados en tanto que se tendría que hacer múltiples recorridos para aquellas funciones cuya complejidad sea O(n).

Requerimiento 2

Descripción

```python
def req_2(data_structs,empresa,ciudad,n):
    """
    Función que soluciona el requerimiento 2
    """

    empresa_seleccionada=mp.get(data_structs['empresas'],empresa)
    lista_ofertas=me.getValue(empresa_seleccionada)
    empresa_y_ciudad = lt.newList("ARRAY_LIST")
    for oferta in lt.iterator(lista_ofertas):
        if oferta["city"] == ciudad:
            lt.addLast(empresa_y_ciudad, oferta)
    if lt.size(empresa_y_ciudad)==0:
        return None
    elif lt.size(empresa_y_ciudad)<n:
        tamanio_ofertas_empresa_ciudad=lt.size(empresa_y_ciudad)
        return (tamanio_ofertas_empresa_ciudad,sa.sort(empresa_y_ciudad, fecha))
    else:
        sublista=lt.subList(empresa_y_ciudad,lt.size(empresa_y_ciudad)-n,n)
        tamanio_ofertas_empresa_ciudad=lt.size(empresa_y_ciudad)
        return (tamanio_ofertas_empresa_ciudad,sa.sort(sublista, fecha))
```

Primero se inicializan las variables separadas por empresa y posteriormente se aplica un filtro para iterar sobre las ofertas de x empresa especificada y encontrar la ciudad especificada. Y finalmente se evalúa en que condicional debe entrar y según esto retorna la respuesta que es el tamaño de ofertas y las ofertas sorteadas por fecha.

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructura de datos del modelo, número de ofertas para visualizar, ciudad de consulta, nombre de empresa.
Salidas	Lista de las ofertas disponibles según el número de ofertas para visualizar y el listado
Implementado (Sí/No)	Sí - Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Inicializa las variables empresa_seleccionada,lista_ofertas y empresa_y_ciudad para poder trabajar sobre las ofertas unicamente de una empresa especifica	O(1)
Iterar sobre las ofertas filtradas de una empresa para filtrar si estan en la ciudad deseada.	O(n)
Se hacen condicionantes para determinar como se calcula el tamaño de las ofertas y como se debe sortear.	O(1)

Se sortean con Shell Sort para retornar los elementos ordenador por fecha.	O(n^3/2)
TOTAL	***O(n^3/2)***

Pruebas Realizadas

Procesadores	**Intel ® Core ™ i7 – 7500u CPU @ 2.70 GHz 2.90 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Pro

Datos constantes de entrada:

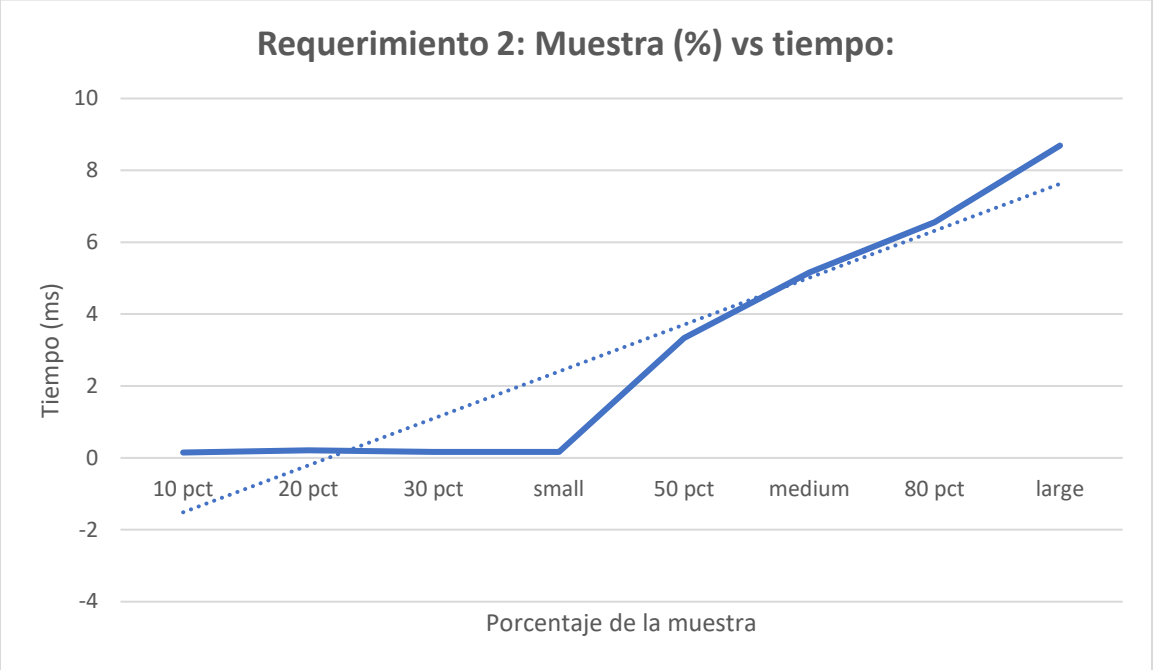
Número de ofertas: 10 Empresa: PGS Software S.A. Ciudad: Gdansk

Entrada	**Tiempo (ms)**
10 pct	0.15
20 pct	0.21
30 pct	0.17
small	0.17
50 pct	3.34
medium	5.16
80 pct	6.56
large	8.69

Tablas de datos

Muestra	**Salida**	**Tiempo (ms)**
10 pct	9 Ofertas - Lista: 9 Elementos	0.15
20 pct	10 Ofertas - Lista:10 elementos	0.21
30 pct	12 Ofertas - Lista: 5 elementos iniciales, 5 elementos finales	0.17
small	14 Ofertas - Lista: 5 elementos iniciales, 5 elementos finales	0.17
50 pct	29 Ofertas - Lista: 5 elementos iniciales, 5 elementos finales	3.34
medium	40 Ofertas - Lista: 5 elementos iniciales, 5 elementos finales	5.16
80 pct	42 Ofertas - Lista: 5 elementos iniciales, 5 elementos finales	6.56
large	42 Ofertas - Lista: 5 elementos iniciales, 5 elementos finales	8.69

Gráfica



Análisis

De acuerdo con el sorteo Shell short la línea muestra cómo se modela el tiempo de acuerdo con como incrementa la cantidad de datos, el programa no adapto la línea tendencia a 3/2 pero sin embargo se entiende como están creciendo el tiempo y los datos debido a que son directamente proporcionales.

Requerimiento 3

Descripción

```python
def req_3(data_structs, nombre, inicio, f_final):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    n_ofertas = 0
    junior = 0
    mid = 0
    senior = 0
    trabajos=mp.get(data_structs['empresas'], nombre)
    lista= me.getValue(trabajos)
    final=lt.newList("ARRAY_LIST")
    for t in lt.iterator(lista):
        fecha_oferta=t["published_at"][:10]
        if fecha_oferta>=inicio and fecha_oferta<=f_final:
            n_ofertas += 1
            if t["experience_level"] == "junior":
                junior += 1
            if t["experience_level"] == "mid":
                mid += 1
            if t["experience_level"] == "senior":
                senior += 1
            diccionario = {"fecha de la oferta": t['published_at'][:10], "titulo de la oferta": t["company_name"], "r
            lt.addLast(final, diccionario)

    return n_ofertas, junior, mid, senior, final
```

Este requerimiento se encarga de buscar la cantidad de ofertas y el nivel de experiencia que se requiere en un rango de tiempo con el nombre de una empresa, en donde primero se hace un for para recorrer todo el mapa de nombres de empresas, para de esta manera buscar las ofertas que coincidan en el rango de fechas y con el nombre de la empresa. Cuando encuentre alguna oferta en el rango de tiempo de dicha empresa, busca cuál es el nivel de experticia requerido para el trabajo, sea “junior”, “mid” o “senior”. Luego guarda los datos necesarios de cada oferta que cumpla las condiciones dadas por el usuario. Tras recorrer toda la lista, ordena los datos guardados en el diccionario por fecha y país.

Entrada	Estructura de datos, nombre de la empresa de consulta, fecha inicial de consulta y fecha final de consulta
Salidas	Número de ofertas, ofertas con nivel de experticia “junior”, ofertas con nivel de experticia “mid”, ofertas con nivel de experticia “senior” y diccionario ordenado con los valores asociados a cada oferta que cumplan las condiciones
Implementado (Sí/No)	Sí – Felipe Boada

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Sacar la pareja llave-valor y luego el valor del mapa que contiene al país en cuestión*(mp.get() y me.getValue())*	**O(1)**
For t in lt.iterator(lista): recorrer toda la lista de trabajos	O(n)
If: comparaciones	O(1)
lt.addLast: Añadir elementos a la última posición de la lista	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Procesadores	**11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz**
Memoria RAM	**16 GB 15,7 GB usables**
Sistema Operativo	**Windows 11**

Datos constantes de entrada:

Empresa:IntelligINTS Fecha inicial:2022-04-13 Fecha final:2022-04-15

Entrada	**Tiempo (ms)**
10 pct	0,13

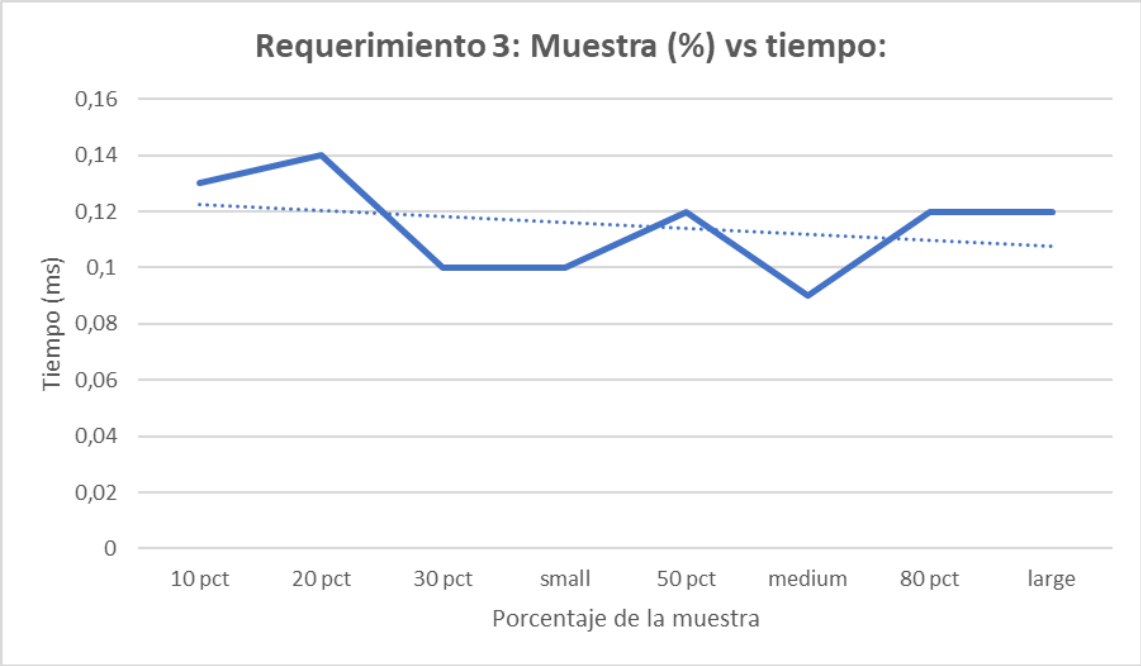
20 pct	0,14
30 pct	0,1
small	0,1
50 pct	0,12
medium	0,09
80 pct	0,12
large	0,12

Tablas de datos

Muestra	Salida	Tiempo (ms)
10 pct	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.13
20 pct	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.14
30 pct	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.1
small	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.1
50 pct	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.12
medium	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.09
80 pct	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0	0.12

	Ofertas con experiencia senior es: 2 El listado de las ofertas son:	
large	El número total de ofertas es: 2 Ofertas con experiencia junior es: 0 Ofertas con experiencias mid es: 0 Ofertas con experiencia senior es: 2 El listado de las ofertas son:	0.12

Gráfica



[1]

Análisis

A pesar de que la complejidad de la función sea igual a o(n), se puede afirmar que los resultados de tiempo son distintos, aunque todos varían en los mismos rangos, es decir, que no son de un crecimiento progresivo, son datos que estan siempre en el mismo rango.

Requerimiento 4

Descripción

[1]

```python
def req_4(data_structs,pais,fecha_inicio,fecha_fin):
    """
    Función que soluciona el requerimiento 4
    """
    empresas=lt.newList()
    ciudades=mp.newMap()
    ofertas=mp.get(data_structs['paises'], pais)
    lista_ofertas=me.getValue(ofertas)
    filtrados=lt.newList('ARRAY_LIST')
    for oferta in lt.iterator(lista_ofertas):
        fecha_oferta=oferta['published_at'][:10]
        nombre_empresa=oferta['company_name']
        if fecha_fin>=fecha_oferta>=fecha_inicio:
            lt.addLast(filtrados, oferta)
            if lt.isPresent(empresas, nombre_empresa):
                pass
            else:
                lt.addLast(empresas, nombre_empresa)
            ciudad=oferta['city']
            lista_ciudades=mp.get(ciudades, ciudad)
            if lista_ciudades:
                lista_ciudades=me.getValue(lista_ciudades)
            else:
                lista_ciudades=new_entry_list(ciudad)
                mp.put(ciudades, ciudad, lista_ciudades)
            lt.addLast(lista_ciudades, oferta)
    city_list=mp.keySet(ciudades)
    offers_list=mp.valueSet(ciudades)
    mayor_ciudad=(None, 0)
    menor_ciudad= (None, 100000000000)
    i=1
    for oferta in lt.iterator(offers_list):
        tamanio=lt.size(oferta)
        if tamanio>mayor_ciudad[1]:
            mayor_ciudad= (lt.getElement(city_list, i), tamanio)
        if tamanio<menor_ciudad[1]:
            menor_ciudad= (lt.getElement(city_list, i), tamanio)
        if tamanio==mayor_ciudad[1]:
            mayor_ciudad= (comparar_promedios_ciudad(data_structs, ciudades, mayor_ciudad[0], (lt.getElement(city_list, i))), tamanio)
        if tamanio==menor_ciudad[1]:
            menor_ciudad= (comparar_promedios_ciudad(data_structs, ciudades, menor_ciudad[0], (lt.getElement(city_list, i))), tamanio)
        i+=1

    return lt.size(filtrados), lt.size(empresas), mp.size(ciudades), mayor_ciudad, menor_ciudad, sa.sort(lista_ofertas, fecha_empresa)
```

```python
def new_entry_list(id):
    id=lt.newList('ARRAY_LIST')
    return id
```

Funciones auxiliares:

```python
def comparar_promedios_ciudad(data_structs, mapa_ciudades, ciudad1, ciudad2):
    ganador=None
    mapa_salarios=data_structs['employments']
    city1=me.getValue(mp.get(mapa_ciudades, ciudad1))
    city2=me.getValue(mp.get(mapa_ciudades, ciudad2))
    con_oferta1=0
    con_oferta2=0
    total1=0
    total2=0
    promedio1=0
    promedio2=0
    for oferta1 in lt.iterator(city1):
        id=oferta1['id']
        actual=me.getValue(mp.get(mapa_salarios, id))
        if actual['salary_from'] != '' and actual['salary_to'] != '':
            salario=(int(actual['salary_from'])+int(actual['salary_to']))/2
            con_oferta1+=1
            total1+=salario

    for oferta2 in lt.iterator(city2):
        id=oferta2['id']
        actual=me.getValue(mp.get(mapa_salarios, id))
        if actual['salary_from'] != '' and actual['salary_to'] != '':
            salario=(int(actual['salary_from'])+int(actual['salary_to']))/2
            con_oferta2+=1
            total2+=salario


    if con_oferta1==0:
        promedio1=0
        if con_oferta2==0:
            promedio2=0
        else:
            promedio2 = total2/con_oferta2
    elif con_oferta2==0:
        promedio2=0
        promedio1 =  total1/con_oferta1
    else:
        promedio1 =  total1/con_oferta1
        promedio2 = total2/con_oferta2

    if promedio1>promedio2:
        ganador=ciudad1
    else:
        ganador=ciudad2
    return ganador
```

El requerimiento 4 se encarga de retornar una lista con las ofertas que se encuentren dentro del periodo ingresado por el usuario y que apliquen para determinado país. Adicionalmente, se presentan el total de ofertas, el total de empresas, el número de ciudades en total y las ciudades con más y menos ofertas. Si no hay información para dicho país en ese rango de años, se retorna None.

Entrada	Código del país, fecha inicial de consulta, fecha final de consulta.
Salidas	Total de ofertas, total de empresas, total de ciudades, ciudad con más ofertas, ciudad con menos ofertas, lista de las ofertas.
Implementado (Sí/No)	Sí – María Alejandra Carrillo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear listas y mapas vacíos *(lt.newList(), mp.newMap())*	O(1)
Sacar la pareja llave-valor y luego el valor del mapa que contiene al país en cuestión*(mp.get() y me.getValue())*	O(1)
Iterar la lista de ofertas *(lt.iterator())* donde se añaden a la lista aquellas que cumplen las condiciones de entrada y se añaden las empresas que no están en una lista. Además, dentro de la iteración hay un *mp.put()* para añadir ofertas dependiendo de la ciudad.	O(n)
Se itera una lista donde están las ciudades con sus correspondientes números de ofertas*(lt.iterator()).* Dentro de la iteración está la función *comparar_promedios_ciudad* cuya complejidad es O(n) porque cuenta las ofertas con salario y las promedia por ciudad.	O(n)
Se organiza la lista de ofertas por la fecha de publicación y el nombre de la empresa con *shellsort (sa.sort())*	O(n^1.5)
TOTAL	***O(n^1.5)***

Pruebas Realizadas

Procesadores	**Intel ® Core ™ i7 – 7500u CPU @ 2.70 GHz 2.90 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Pro

Datos constantes de entrada:

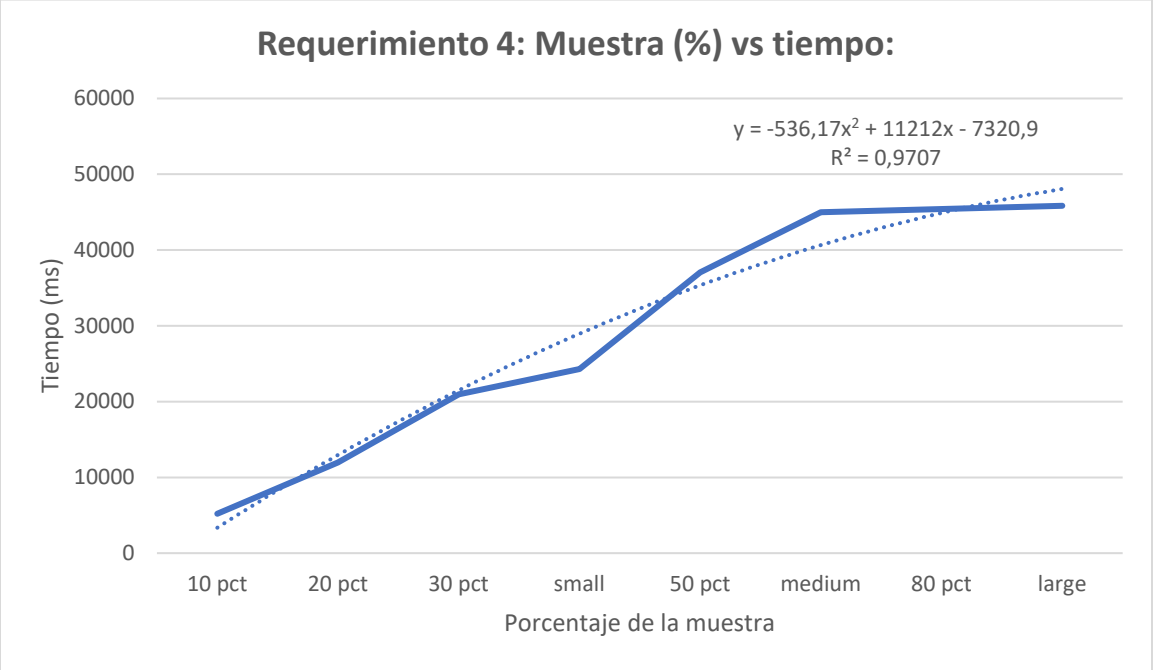
País: PL Fecha inicial: 2021-12-12 Fecha final: 2022-12-12

Entrada	**Tiempo (ms)**
10 pct	5199,83
20 pct	11948,86
30 pct	20970,24
small	24303,22
50 pct	37057,95
medium	44988,95
80 pct	45391,31
large	45838,25

Tablas de datos

Muestra	Salida	Tiempo (ms)
10 pct	Ofertas país: 17591 – Empresas: 3184 – Ciudades: 176 – Mayor: Warzawa (7666) – Menor: Varsavia (1) – Lista: mayor a 10.	5199,83
20 pct	Ofertas país: 32116 – Empresas: 4009 – Ciudades: 320 – Mayor: Warzawa (12117) – Menor: Berlin(1) – Lista: mayor a 10.	11948,86
30 pct	Ofertas país: 50124 – Empresas: 4240 – Ciudades: 464 – Mayor: Warzawa (14289) – Menor: Berlin (1) – Lista: mayor a 10.	20970,24
small	Ofertas país: 55674 – Empresas: 4276 – Ciudades: 486 – Mayor: Warzawa (14825) – Menor: Berlin (1) – Lista: mayor a 10.	24303,22
50 pct	Ofertas país: 82882 – Empresas: 4422 – Ciudades: 598 – Mayor: Warzawa (17506) – Menor: warzawa (1) – Lista: mayor a 10.	37057,95
medium	Ofertas país: 94747– Empresas: 4473 – Ciudades: 618 – Mayor: Warzawa (18830) – Menor: Катовіце (1) – Lista: mayor a 10.	44988,95
80 pct	Ofertas país: 99121– Empresas: 4531 – Ciudades: 623 – Mayor: Warzawa (19551) – Menor: Катовіце (1) – Lista: mayor a 10.	45391,31
large	Ofertas país: 100384 – Empresas: 4548 – Ciudades: 629 – Mayor: Warzawa (19745) – Menor: Катовіце (1) – Lista: mayor a 10.	45838,25

Gráficas



Análisis

A partir de la gráfica, se puede observar cómo a medida que el porcentaje de la muestra aumenta, el tiempo de carga se vuelve mayor; la gráfica tiene un comportamiento que se aproxima al de una cuadrática. La línea de tendencia escogida no pudo ser de forma x^1.5, así que se aproximó a x^2 por las limitaciones de Word y sus gráficas. Teniendo en cuenta que la complejidad del requerimiento es de O(n^1.5) debido a el ordenamiento de los datos que se hace al final del programa, esto es acorde con lo que muestra la gráfica. Además, hay que tener en consideración que a lo largo de la implementación de la función se utilizan otras funciones con complejidades de O(n) y O (n^1.5) nuevamente en el peor de los casos. Aquello puede influenciar en los resultados esperados en tanto que se tendría que hacer múltiples recorridos para aquellas funciones cuya complejidad sea O(n).

Requerimiento 5

```python
def req_5(data_structs,ciudad,fecha_inicio,fecha_fin):

    ofertas=mp.get(data_structs['ciudades'],ciudad)
    ofertas_ciudad=me.getValue(ofertas)
    filtrados=mp.newMap(maptype='PROBING',loadfactor=0.5)
    lista_ofertas_filtrados=lt.newList('ARRAY_LIST')
    totales_ofertas=0
    for oferta in lt.iterator(ofertas_ciudad):
        fecha_oferta=oferta['published_at'][:10]
        if (fecha_inicio<=fecha_oferta) and (fecha_oferta<=fecha_fin):
            totales_ofertas+=1
            lt.addLast(lista_ofertas_filtrados,oferta)
            if not mp.contains(filtrados,oferta['company_name']):
                mp.put(filtrados,oferta['company_name'],lt.newList('ARRAY_LIST'))
            empresa=mp.get(filtrados,oferta['company_name'])
            lt.addLast(me.getValue(empresa),oferta)
    total_empresas=mp.size(filtrados)

    menor=(None,1000000000000)
    mayor=(None,0)
    for empresa in lt.iterator(mp.keySet(filtrados)):
        lista_empresa = mp.get(filtrados, empresa)
        if lista_empresa is not None:
            num_ofertas = lt.size(me.getValue(lista_empresa))
            if menor is None or num_ofertas < menor[1]:
                menor = (empresa, num_ofertas)
            if mayor is None or num_ofertas > mayor[1]:
                mayor = (empresa, num_ofertas)

    if menor[1]==1000000000000:
        menor=(None,0)



    return totales_ofertas,total_empresas,mayor,menor,sa.sort(lista_ofertas_filtrados,fecha_empresa)
```

Se cargan inicialmente las estructuras de mapas divididas por ciudad y se aplica una iteración para filtrar entre el rango de fechas ingresado por el usuario, posteriormente se crea un mapa con todas las ofertas filtradas organizadas por empresa y se crea una lista array_list para almacenarlas todas de manera indefinida. Seguido a este proceso se itera sobre el mapa de filtrados se saca el tamaño de ofertas que tiene cada oferta y se calcula de esta forma la empresa con mayor y menor número de ofertas y finalmente se devuelve el total de ofertas que cumplen los filtros, el total de empresas que publicaron, el mayor, el menor y la lista de todas las ofertas ordenadas por fecha de publicación.

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Data_structs,ciudad,fecha inicio, fecha final
Salidas	El total de ofertas publicadas en la ciudad en el periodo de consulta. • El total de empresas que publicaron por lo menos una

	oferta en la ciudad de consulta. • Empresa con mayor número de ofertas y su conteo • Empresa con menor número de ofertas (al menos una) y su conteo • El listado de ofertas publicadas ordenadas cronológicamente por fecha y nombre de la empresa (v.gr. Para dos ofertas con la misma fecha, el orden lo decide la empresa de forma alfabética). Cada uno de los elementos debe presentar la siguiente información: o Fecha de publicación de la oferta o Título de la oferta o Nombre de la empresa de la oferta o Tipo de lugar de trabajo de la oferta o Tamaño de la empresa de la oferta o Tipo de lugar de trabajo de la oferta
Implementado (Sí/No)	Sí – Nelson Felipe Celis

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Se inicializan las variables con las estructuras creadas en la carga de datos.	O(1)
Iterar sobre las ofertas filtradas de una ciudad para filtrar si están en el rango de fechas deseados.	O(n)
Se itera dentro del mapa de filtrados para buscar el mayor y el menor dentro de todas las ofertas filtradas por los parámetros ingresados.	O(n)
Finalmente se retornan todos los datos y se sortea la lista de ofertas por fecha y en caso de empate se resuelve por la menor empresa alfabéticamente.	O(n^3/2)
TOTAL	***O(n^3/2)***

Pruebas Realizadas

Procesadores	**Intel ® Core ™ i7 – 7500u CPU @ 2.70 GHz 2.90 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Pro

Datos constantes de entrada:

Ciudad: Warszawa Fecha inicial: 2020-03-10 Fecha final: 2023-03-10

Entrada	**Tiempo (ms)**
10 pct	235.75
20 pct	351.47
30 pct	403.46
small	443.16
50 pct	974.31
medium	1838.87

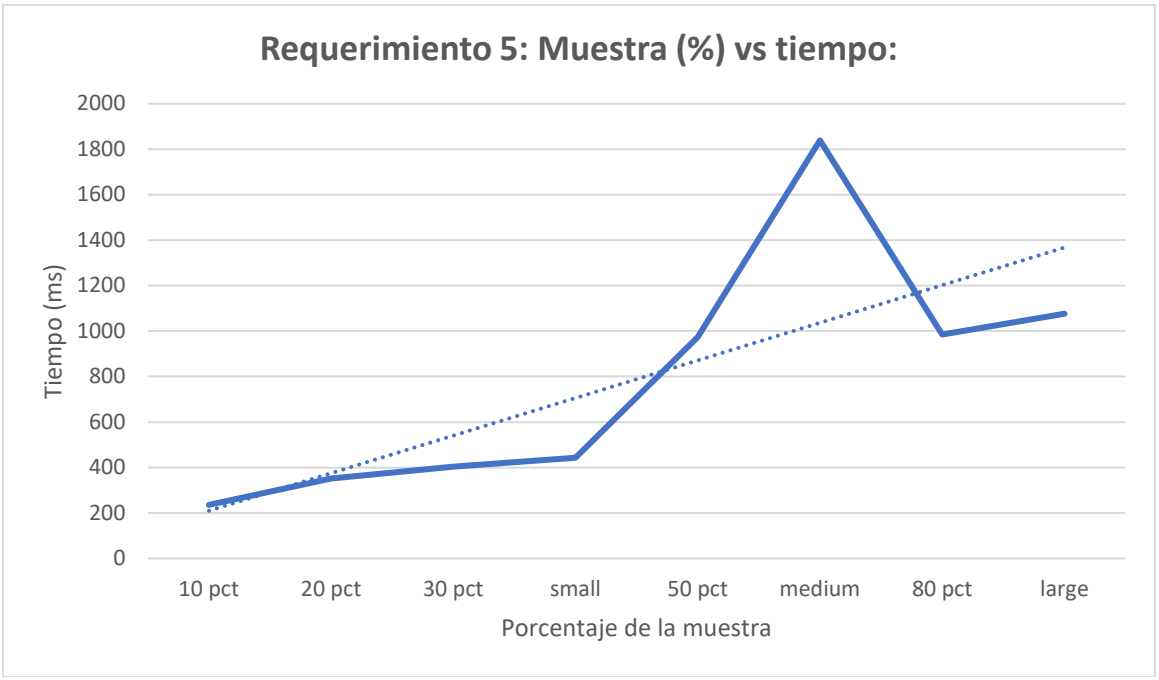
80 pct	985.5
large	1076.87

Tablas de datos

Muestra	Salida	Tiempo (ms)
10 pct	Hay 10198 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 1708 empresas que publicaron al menos una oferta en Warszawa . DevsData LLC fue la empresa con mayor número de ofertas. Hay 182 ofertas. G2A fue la empresa con menor número de ofertas. Hay 1 ofertas.	235.75
20 pct	Hay 15518 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 2299 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 246 ofertas. GetResponse S.A. fue la empresa con menor número de ofertas. Hay 1 ofertas.	351.47
30 pct	Hay 18066 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 2667 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 259 ofertas. Zowie fue la empresa con menor número de ofertas. Hay 1 ofertas.	403.46
small	Hay 18701 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 2751 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 259 ofertas. iteo fue la empresa con menor número de ofertas. Hay 1 ofertas.	443.16
50 pct	Hay 22007 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 3153 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 261 ofertas.	974.31

	Valio Sp. z o.o. fue la empresa con menor número de ofertas. Hay 1 ofertas.	
medium	Hay 24045 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 3344 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 278 ofertas. INFOTARGET fue la empresa con menor número de ofertas. Hay 1 ofertas.	1838.87
80 pct	Hay 24587 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 3387 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 283 ofertas. Exea Data Center fue la empresa con menor número de ofertas. Hay 1 ofertas.	985.5
large	Hay 24709 ofertas en Warszawa entre 2020-03-10 y 2023-03-10 . Hay 3390 empresas que publicaron al menos una oferta en Warszawa . Bosch Polska fue la empresa con mayor número de ofertas. Hay 283 ofertas. Bitpanda fue la empresa con menor número de ofertas. Hay 1 ofertas.	1076.87

Gráfica



Análisis

Por lo general la gráfica es congruente con el análisis previamente hecho de la complejidad sin embargo, hay un dato que rompe completamente la tendencia en médium siendo esto un factor externo el cual hizo que tomara más tiempo la respuesta sin embargo, todos los datos fueron tomados bajos las mismas condiciones por lo que debería ser más precisa esta información.

Requerimiento 6

Descripción

```python
def req_6(data_structs,n, exp, anio):
    """
    Función que soluciona el requerimiento 6
    """
    rta=None
    if exp != "indiferente":
        mapa_exp_anios= data_structs["experiencia"]
        mapa_anios= me.getValue(mp.get(mapa_exp_anios, exp))
        if mp.get(mapa_anios, anio) == None:
            return rta
        lista_ofertas= me.getValue(mp.get(mapa_anios, anio))

    else:
        mapa=data_structs['anios']
        lista_ofertas= me.getValue(mp.get(mapa, anio))
        if lista_ofertas == None:
            return rta

    mapa_ciudades=mp.newMap()
    lista_empresas=lt.newList()
    for oferta in lt.iterator(lista_ofertas):
        key_ciudad=oferta['city']
        ciudad=mp.get(mapa_ciudades,key_ciudad)
        if ciudad:
            ciudad=me.getValue(ciudad)
        else:
            ciudad=new_entry_list(key_ciudad)
            mp.put(mapa_ciudades,key_ciudad,ciudad)
        lt.addLast(ciudad,oferta)
        if lt.isPresent(lista_empresas, oferta['company_name'])==0:
            lt.addLast(lista_empresas, oferta['company_name'])

    mapa_counter_ciudades= mp.newMap()
    llaves_ciudad= mp.keySet(mapa_ciudades)
    valores_ciudad= mp.valueSet(mapa_ciudades)
    i = 1
    for ciudad in lt.iterator(llaves_ciudad):
        size = lt.size(lt.getElement (valores_ciudad, i))
        promedio=promedio_por_ciudad(data_structs, mapa_ciudades, ciudad)
        mp.put(mapa_counter_ciudades, ciudad, (ciudad, size, promedio))
        i += 1
    mapa_sorteado_ciudades=mp.newMap()
    valores_ciudad_pre= mp.valueSet(mapa_counter_ciudades)
    sa.sort(valores_ciudad_pre, numero_ofertas)
    if lt.size(valores_ciudad_pre)==0:
        return None
    elif lt.size(valores_ciudad_pre)<n:
        lista_sorteada=valores_ciudad_pre
    else:
        lista_sorteada=lt.subList(valores_ciudad_pre, 1, n)

    for city in lt.iterator(lista_sorteada):
        lista_ofertas_city=me.getValue(mp.get(mapa_ciudades, city[0]))
        pais=lt.getElement(lista_ofertas_city, 1)['country_code']
        total_empresas, mayor_empresa = calcular_empresas(mapa_ciudades, city[0])
        mejor_oferta , peor_oferta = calcular_mejor_peor_oferta(data_structs, mapa_ciudades, city[0])
        mp.put(mapa_sorteado_ciudades, city[0], [city[0], pais, city[1], city[2], total_empresas, mayor_empresa, mejor_oferta, peor_oferta]
    numero_ciudades=lt.size(lista_sorteada)
    numero_empresas=lt.size(lista_empresas)
    mejor_ciudad=(lt.getElement(lista_sorteada, 1)[0], lt.getElement(lista_sorteada, 1)[1])
    menor_ciudad=(lt.getElement(lista_sorteada, numero_ciudades)[0], lt.getElement(lista_sorteada, numero_ciudades)[1])
    rta=(numero_ciudades, numero_empresas, lt.size(lista_ofertas), mejor_ciudad, menor_ciudad, mp.valueSet(mapa_sorteado_ciudades))

    return rta
```

Funciones auxiliares:

```python
def calcular_empresas(mapa, item):
    lista_ofertas=me.getValue(mp.get(mapa, item))
    mapa_empresas=mp.newMap()
    for oferta in lt.iterator(lista_ofertas):
        if mp.contains(mapa_empresas, oferta['company_name']):
            mp.put(mapa_empresas, oferta['company_name'], (oferta['company_name'], me.getValue(mp.get(mapa_empresas, oferta['company_name'])
        else:
            mp.put(mapa_empresas, oferta['company_name'], (oferta['company_name'],1))
    valores=mp.valueSet(mapa_empresas)
    sa.sort(valores, oferta_nombre_pais)
```

```python
def promedio_por_ciudad(data_structs, mapa_ciudades, ciudad):
    promedio=0
    mapa_salarios=data_structs['employments']
    city=me.getValue(mp.get(mapa_ciudades, ciudad))
    con_oferta=0
    total=0
    for oferta in lt.iterator(city):
        id=oferta['id']
        actual=me.getValue(mp.get(mapa_salarios, id))
        if actual['salary_from'] != '' and actual['salary_to'] != '':
            salario=(int(actual['salary_from'])+int(actual['salary_to']))/2
            con_oferta+=1
            total+=salario

    if con_oferta==0:
        promedio=0
    else:
        promedio=total/con_oferta
    return promedio
```

```python
def calcular_mejor_peor_oferta(data_structs, mapa, item):
    lista_ofertas=me.getValue(mp.get(mapa, item))
    mapa_salarios=data_structs['employments']
    ofertas_oredenadas_max=lt.newList()
    ofertas_oredenadas_min=lt.newList()
    for oferta in lt.iterator(lista_ofertas):
        id=oferta['id']
        actual=me.getValue(mp.get(mapa_salarios, id))
        if actual['salary_from'] != '' and actual['salary_to'] != '':
            high=int(actual['salary_to'])
            low=int(actual['salary_from'])
            lt.addLast(ofertas_oredenadas_max, (oferta, high))
            lt.addLast(ofertas_oredenadas_min, (oferta, low))
    sa.sort(ofertas_oredenadas_max, mayor_salario)
    sa.sort(ofertas_oredenadas_min, mayor_salario)


    mejor_oferta=lt.firstElement(ofertas_oredenadas_max)
    peor_oferta=lt.lastElement(ofertas_oredenadas_min)

    return mejor_oferta, peor_oferta
```

El requerimiento 6 le pide al usuario un número de ciudades para consular, un nivel de experticia (el cual puede ser opcional) y un año para la consulta. Si no hay ciudades que tengan ofertas en ese año junto con dicho nivel de experticia, se retorna None. De lo contrario, se presentará el total de ciudades, el total de empresas, el total de ofertas publicadas, la ciudad con más empleos, la ciudad con menos empleos, y una lista para cada ciudad. Dentro de esta se encuentra el nombre de la ciudad, el país, el total de ofertas, el promedio del salario (sólo contando las ofertas que tuviesen salario), el total de empresas, la empresa con más ofertas, la mejor oferta por salario y la peor oferta por salario.

Entrada	Número de ciudades, nivel de experticia (se puede omitir) y el año de consulta.
Salidas	Total de ciudades, total de empresas, total de ofertas, ciudad con más ofertas, ciudad con menos ofertas y el listado de las ciudades.
Implementado (Sí/No)	Sí - Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Tanto para cuando se ingresa un nivel de experiencia o no, se hacen múltiples *get* para acceder a las estructuras que tienen la lista de ofertas correspondientes al año especificado.	O(1)
Creación de listas y mapas *(lt.newList() y mp.newMap())*	O(1)
Se itera a lista de ofertas para poner en el mapa las ciudades de cada una y agregar las empresas que hay	O(n)

junto con el conteo de cada una. Se hacen *getValue(), isPresent() y addLast()* dentro de la iteración.	
Se obtienen listas de las llaves y valores del mapa de las ciudades resultantes del paso anterior.	O(n)
Se itera sobre las llaves de las ciudades en el mapa para invocar una función que determinará el promedio por cada ciudad (O(n)). Luego este resultado se agrega al mapa *(mp.put())*	O(n)
Se hace un ordenamiento de los valores por ciudad, el cual es la serie de valores del mapa de las ciudades *(sa.sort())*	O(n^1.5)
Se hace una sublista con los n valores solicitados por el usuario. *(lt.sublist())*	O(n)
Se itera sobre la lista sorteada para obtener el país, total de empresas, y mejor/peor oferta por ciudad, lo cual se agrega al mapa de las n ciudades actualizado. Allí dentro de invocan dos funciones *(calcular_empresas y calcular_mejor_peor_oferta)* con complejidad O(n^1.5) cada una ya que dentro de cada ciudad se ordenan tanto las ofertas por salario como las empresas por número de ofertas.	O(n^1.5)
TOTAL	***O(n^1.5)***

Pruebas Realizadas

Procesadores	**Intel ® Core ™ i7 – 7500u CPU @ 2.70 GHz 2.90 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 10 Pro

Datos constantes de entrada:

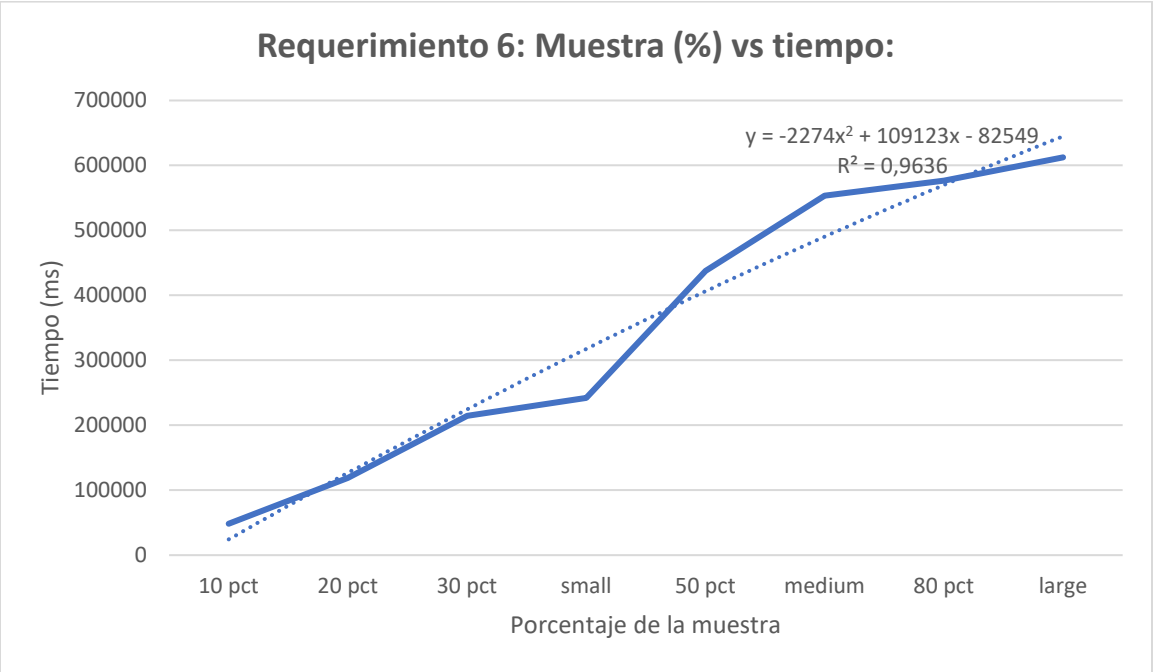
Número de ciudades: 7 Nivel de experticia: mid Año de consulta: 2022

Entrada	**Tiempo (ms)**
10 pct	48381,67
20 pct	118997,96
30 pct	214736,91
small	242079,28
50 pct	437604,81
medium	553472,47
80 pct	576436,76
large	612416,27

Tablas de datos

Muestra	Salida	Tiempo (ms)
10 pct	- Ciudades: 7 – Empresas: 2655 - Ofertas: 10377 - Mayor ciudad: Warzawa (4392) – Menor ciudad: Lodz (321) – Lista: 7 ciudades	48381,67
20 pct	- Ciudades: 7 – Empresas: 3355 - Ofertas: 18581 - Mayor ciudad: Warzawa (6918) – Menor ciudad: Lodz (600) – Lista: 7 ciudades	118997,96
30 pct	- Ciudades: 7 – Empresas: 3526 - Ofertas: 29976 - Mayor ciudad: Warzawa (8129) – Menor ciudad: Lodz (1124) – Lista: 7 ciudades	214736,91
small	- Ciudades: 7 – Empresas: 3534- Ofertas: 33389 - Mayor ciudad: Warzawa (8456) – Menor ciudad: Lodz(1279) – Lista: 7 ciudades	242079,28
50 pct	- Ciudades: 7 – Empresas: 3580 - Ofertas: 50223 - Mayor ciudad: Warzawa (9968) – Menor ciudad: Lodz(2069) – Lista: 7 ciudades	437604,81
medium	- Ciudades: 7 – Empresas: 3609 - Ofertas: 56861 - Mayor ciudad: Warzawa (10694) – Menor ciudad: Lodz (2364) – Lista: 7 ciudades	553472,47
80 pct	- Ciudades: 7 – Empresas: 3623 - Ofertas: 58121 - Mayor ciudad: Warzawa (10891) – Menor ciudad: Lodz (2403) – Lista: 7 ciudades	576436,76
large	- Ciudades: 7 – Empresas: 3625 - Ofertas: 58921 - Mayor ciudad: Warzawa (11006) – Menor ciudad: Lodz (2426) – Lista: 7 ciudades	612416,27

Gráfica



Análisis

A partir de la gráfica, se puede observar cómo a medida que el porcentaje de la muestra aumenta, el tiempo de carga aumenta en grandes cantidades. El crecimiento temporal se asemeja al de una función lineal, más no es la recta de mejor ajuste. La línea de tendencia escogida no pudo ser de forma x^1.5, así que se aproximó a x^2 por las limitaciones de Word y sus gráficas. Teniendo en cuenta que la complejidad del requerimiento es de O(n^1.5) debido a el ordenamiento de los datos que se hace al final del programa, esto es medianamente acorde a lo mostrado por la gráfica. Además, hay que tener en consideración que a lo largo de la implementación de la función se utilizan otras funciones con complejidades de O(n) en el peor de los casos. Aquello puede influenciar en los resultados esperados en tanto que se tendría que hacer múltiples recorridos para aquellas funciones cuya complejidad sea O(n).

Requerimiento 7

```python
def req_7(data_structs,n_paises,anio_mes):
    """
    Función que soluciona el requerimiento 7
    """
    mapa=data_structs['anio_mes']
    mapa_exp= data_structs["skills"]
    lst_sedes=data_structs['multilocations_list']
    mayores_paises=mp.newMap(maptype='CHAINING',loadfactor=1.5)
    mapa_anio_mes_pais= me.getValue(mp.get(mapa, anio_mes))
    #####Tomamos varias estructuras que necesitaremos para las operaciones
    mapa_counter_pais= mp.newMap()
    llaves_pais= mp.keySet(mapa_anio_mes_pais)
    valores_pais= mp.valueSet(mapa_anio_mes_pais)
    #Se llena el mapa_counter_pais con el nombre del pais y su tamaño de ofertas
    i = 1
    for pais in lt.iterator(llaves_pais):
        size = lt.size(lt.getElement (valores_pais, i))
        mp.put(mapa_counter_pais, pais, (pais,size))
        i += 1
    #Se sortea y se toman las n ofertas con mayor numero de ofertas
    valores_pais_pre= mp.valueSet(mapa_counter_pais)
    sa.sort(valores_pais_pre, oferta_nombre_pais )
    lista_sorteada=lt.subList(valores_pais_pre, 1, n_paises)
    mayor_pais=lt.getElement(lista_sorteada,1)
    num_ofertas=0
    #Se itera sobre la lista sorteada para llenar el mapa mayores_paises y poder trabajar sobre las ofertas filtradas
    for pais in lt.iterator(lista_sorteada):
        num_ofertas+=pais[1]
        pais=pais[0]
        ofertas_fecha=mp.get(mapa_anio_mes_pais,pais)
        ofertas=me.getValue(ofertas_fecha)
        mp.put(mayores_paises,pais,ofertas)
```

```
mayores_ciudades=mp.newMap()
llaves_pais_mayores=mp.keySet(mayores_paises)
#Se itera para sacar un mapa de ciudades y calcular la información requerida
total_ofertas_sorteadas=lt.newList('ARRAY_LIST')
for pais in lt.iterator(llaves_pais_mayores):
    ofertas_pais=mp.get(mayores_paises,pais)
    for oferta in lt.iterator(me.getValue(ofertas_pais)):
        lt.addLast(total_ofertas_sorteadas,oferta)
        if not mp.contains(mayores_ciudades,oferta['city']):
            mp.put(mayores_ciudades,oferta['city'],(oferta['city'],0))
        valor=me.getValue(mp.get(mayores_ciudades,oferta['city']))
        mp.put(mayores_ciudades,oferta['city'],(oferta['city'],valor[1]+1))
ciudades_totales=mp.valueSet(mayores_ciudades)
sa.sort(ciudades_totales,oferta_nombre_pais)
num_ciudades=lt.size(ciudades_totales)
mayor_ciudad=lt.getElement(ciudades_totales,1)

#Se crean mapas y se llenan con las diferentes habilidades
habilidades_junior=mp.newMap()
habilidades_mid=mp.newMap()
habilidades_senior=mp.newMap()

level_j=0
level_m=0
level_s=0

empresas_j=mp.newMap()
empresas_m=mp.newMap()
empresas_s=mp.newMap()
```

```
empresas_j=mp.newMap()
empresas_m=mp.newMap()
empresas_s=mp.newMap()

sedes_j=0
sedes_m=0
sedes_s=0

emp_multi_j=lt.newList('ARRAY_LIST')
emp_multi_m=lt.newList('ARRAY_LIST')
emp_multi_s=lt.newList('ARRAY_LIST')

num_of_j=0
num_of_m=0
num_of_s=0

for oferta in lt.iterator(total_ofertas_sorteadas):
    id=oferta['id']
    ofer_habilidad=me.getValue(mp.get(mapa_exp,id))
    nom_empresa=oferta['company_name']

    if oferta['experience_level']=='junior':
        if not mp.contains(empresas_j,nom_empresa) :
            mp.put(empresas_j,nom_empresa,(nom_empresa,0))
        valor=me.getValue(mp.get(empresas_j,nom_empresa))
        mp.put(empresas_j,nom_empresa,(nom_empresa,valor[1]+1))
        for habilidad in lt.iterator(ofer_habilidad):
            if not mp.contains(habilidades_junior,habilidad['name']):
                mp.put(habilidades_junior,habilidad['name'],(habilidad['name'],0))
            valor=me.getValue(mp.get(habilidades_junior,habilidad['name']))
            mp.put(habilidades_junior,habilidad['name'],(habilidad['name'],valor[1]+1))
```

```
            level_j+=int(habilidad['level'])
            num_of_j+=1
        if lt.isPresent(lst_sedes,id) and not lt.isPresent(emp_multi_j,nom_empresa):
            sedes_j+=1
            lt.addLast(emp_multi_j,nom_empresa)



    elif oferta['experience_level']=='mid':
        if not mp.contains(empresas_m,nom_empresa) :
            mp.put(empresas_m,nom_empresa,(nom_empresa,0))
        valor=me.getValue(mp.get(empresas_m,nom_empresa))
        mp.put(empresas_m,nom_empresa,(nom_empresa,valor[1]+1))
        for habilidad in lt.iterator(ofer_habilidad):
            if not mp.contains(habilidades_mid,habilidad['name']):
                mp.put(habilidades_mid,habilidad['name'],(habilidad['name'],0))
            valor=me.getValue(mp.get(habilidades_mid,habilidad['name']))
            mp.put(habilidades_mid,habilidad['name'],(habilidad['name'],valor[1]+1))
            level_m+=int(habilidad['level'])
            num_of_m+=1
        if lt.isPresent(lst_sedes,id) and not lt.isPresent(emp_multi_m,nom_empresa):
            sedes_m+=1
            lt.addLast(emp_multi_m,nom_empresa)


    elif oferta['experience_level']=='senior':
        if not mp.contains(empresas_s,nom_empresa) :
            mp.put(empresas_s,nom_empresa,(nom_empresa,0))
        valor=me.getValue(mp.get(empresas_s,nom_empresa))
        mp.put(empresas_s,nom_empresa,(nom_empresa,valor[1]+1))
        for habilidad in lt.iterator(ofer_habilidad):
```

```
        for habilidad in lt.iterator(ofer_habilidad):
            if not mp.contains(habilidades_senior,habilidad['name']):
                mp.put(habilidades_senior,habilidad['name'],(habilidad['name'],0))
            valor=me.getValue(mp.get(habilidades_senior,habilidad['name']))
            mp.put(habilidades_senior,habilidad['name'],(habilidad['name'],valor[1]+1))
            level_s+=int(habilidad['level'])
            num_of_s+=1
        if lt.isPresent(lst_sedes,id) and not lt.isPresent(emp_multi_s,nom_empresa):
            sedes_s+=1
            lt.addLast(emp_multi_s,nom_empresa)


    #Se calcula el tamaño, mayor, menor y el promedio de las habilidades

    menor_hab_j,mayor_hab_j=calcular_mayor_y_menor(mp.valueSet(habilidades_junior))
    menor_hab_m,mayor_hab_m=calcular_mayor_y_menor(mp.valueSet(habilidades_mid))
    menor_hab_s,mayor_hab_s=calcular_mayor_y_menor(mp.valueSet(habilidades_senior))

    tamanio_hab_junior=mp.size(habilidades_junior)
    tamanio_hab_mid=mp.size(habilidades_mid)
    tamanio_hab_senior=mp.size(habilidades_senior)

    prom_j=promedio(num_of_j,level_j)
    prom_m=promedio(num_of_m,level_m)
    prom_s=promedio(num_of_s,level_s)

    tamanio_emp_junior=mp.size(empresas_j)
    tamanio_emp_mid=mp.size(empresas_m)
    tamanio_emp_senior=mp.size(empresas_s)

    menor_emp_j,mayor_emp_j=calcular_mayor_y_menor(mp.valueSet(empresas_j))
    menor_emp_m,mayor_emp_m=calcular_mayor_y_menor(mp.valueSet(empresas_m))
    menor_emp_s,mayor_emp_s=calcular_mayor_y_menor(mp.valueSet(empresas_s))

    junior=[tamanio_hab_junior,mayor_hab_j,menor_hab_j,prom_j,tamanio_emp_junior,mayor_emp_j,menor_emp_j,sedes_j]
    mid=[tamanio_hab_mid,mayor_hab_m,menor_hab_m,prom_m,tamanio_emp_mid,mayor_emp_m,menor_emp_m,sedes_m]
    senior=[tamanio_hab_senior,mayor_hab_s,menor_hab_s,prom_s,tamanio_emp_senior,mayor_emp_s,menor_emp_s,sedes_s]

    return num_ofertas,num_ciudades,mayor_pais,mayor_ciudad,junior,mid,senior
```

La función inicializa las estructuras y posteriormente empieza a hacer iteraciones para llenar otras estructuras auxiliares como mapas y array_list para de esta manera facilitar como se recorre la información y se sacan los datos requeridos, a medida que se van filtrando las ofertas se va a reducir el número de ofertas sobre el cual se trabaja y de esta manera es más amigable para la memoria este trabajo. Se accede a las habilidades y se crean nuevas estructuras para almacenar toda la información de cada uno de los niveles de experticia que requiere la respuesta.

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Numero de países, año y mes de consulta
Salidas	El total de ofertas de empleo. • Número de ciudades donde se ofertó en los países resultantes de la consulta. • Nombre del país con mayor cantidad de ofertas y su conteo • Nombre de la ciudad con mayor cantidad de ofertas y su conteo • Para el conjunto de las ofertas de trabajo en los países resultantes de la consulta, por cada uno de los tres niveles de experticia (junior, mid y senior) calcule y presente la siguiente información: o Conteo de habilidades

	diferentes solicitadas en ofertas de trabajo o Nombre de la habilidad más solicitada y su conteo en ofertas de trabajo o Nombre de la habilidad menos solicitada y su conteo en ofertas de trabajo o Nivel mínimo promedio de las habilidades o Conteo de empresas que publicaron una oferta con este nivel de experticia o Nombre de la empresa con mayor número de ofertas y su conteo o Nombre de la empresa con menor número de ofertas (al menos una) y su conteo o Número de empresas que publicaron una oferta en este nivel de experticia que tienen una o más
Implementado (Sí/No)	Si-Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Se inicializan todas las estructuras cargadas de la carga de datos para acceder más fácilmente a las ofertas que buscamos por acceso directo.	O(1)
Se hace un pequeño sorteo por número de ofertas y en caso de empate por orden alfabéticamente	O(nˆ3/2)
Se hace un ciclo para llenar la estructura de counter_paises el cual nos ayudara para saber cuáles son los n países con mayor cantidad de ofertas	O(n)
Se hace un ciclo sobre los n países para sacar otro mapa con las ciudades con mayor número de ofertas	O(n)
Se crean múltiples estructuras y contadores para sacar la información de cada nivel de experticia requerida.	O(1)
Se recorren todas las ofertas totales y se anida otro ciclo para ir recorriendo las habilidades y conseguir la información correspondiente.	O(nˆ2)
Se hacen otros sorteos para definir información en casos de empate y ordenarlos	O(nˆ3/2)
TOTAL	***O(*nˆ2)**

Pruebas Realizadas

Procesadores	**Intel ® Core ™ i7 – 7500u CPU @ 2.70 GHz 2.90 GHz**	
Memoria RAM	16.0 GB	
Sistema Operativo	Windows 10 Pro	Windows?

Datos constantes de entrada:

Número de países: 10 Año y mes de consulta: 2022-09

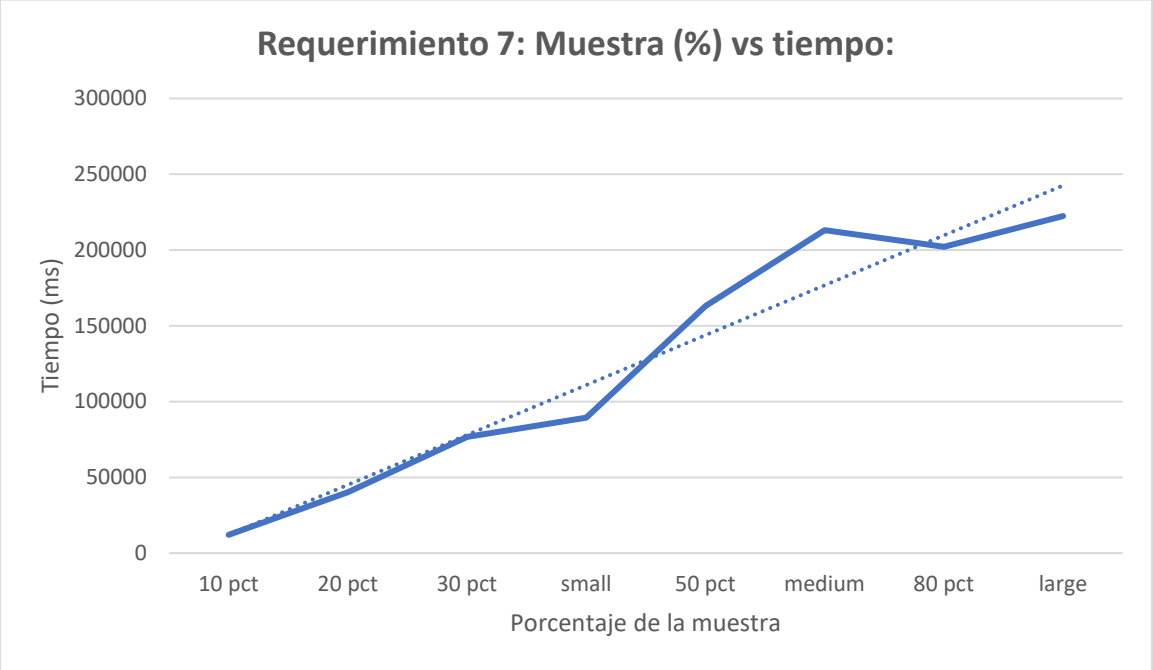
Entrada	Tiempo (ms)
10 pct	12100,48
20 pct	40236,78
30 pct	76691,48
small	89314,81
50 pct	163118,58
medium	213020,18
80 pct	202162,69
large	222423,53

Tablas de datos

Muestra	Salida	Tiempo (ms)
10 pct	Hay 2314 ofertas en 10 paises en 2022-09 . Hay 89 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 2254 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 976 ofertas. (Informacion de Junior,Mid y Senior)	12100.48
20 pct	Hay 4349 ofertas en 10 paises en 2022-09 . Hay 155 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 4251 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 1368 ofertas. (Informacion de Junior,Mid y Senior)	40236.78
30 pct	Hay 7528 ofertas en 10 paises en 2022-09 . Hay 242 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 7406 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 1574 ofertas. (Informacion de Junior,Mid y Senior)	76691.48
small	Hay 8172 ofertas en 10 paises en 2022-09 . Hay 290 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 8027 ofertas.	89314.81

	Warszawa fue la ciudad con mayor número de ofertas. Hay 1608 ofertas. (Informacion de Junior,Mid y Senior)	
50 pct	Hay 11457 ofertas en 10 paises en 2022-09 . Hay 396 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 11262 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 1777 ofertas. (Informacion de Junior,Mid y Senior)	163118.58
medium	Hay 13395 ofertas en 10 paises en 2022-09 . Hay 408 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 13184 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 1998 ofertas. (Informacion de Junior,Mid y Senior)	213020.18
80 pct	Hay 12976 ofertas en 10 paises en 2022-09 . Hay 401 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 12749 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 1965 ofertas. (Informacion de Junior,Mid y Senior)	202162.69
large	Hay 13543 ofertas en 10 paises en 2022-09 . Hay 436 ciudades que publicaron al menos una oferta en los 10 paises. PL fue el pais con mayor número de ofertas. Hay 13294 ofertas. Warszawa fue la ciudad con mayor número de ofertas. Hay 2019 ofertas. (Informacion de Junior,Mid y Senior)	222423.53

Gráfica



Análisis

Dentro de la gráfica podemos ver que a pesar de tener algunos casos excepcionales, en su mayoría se sigue la complejidad de ***O(*n^2)** por lo que el análisis es muy correcto y acertado. De igual forma las variaciones pueden ocurrir por factores externos como el computador desde el cual se corre y el tipo de rendimiento que tiene el computador para probar el programa.