ANÁLISIS DEL RETO

Diego Alejandro Munevar Perez, 202322221 , d.munevar@uniandes.edu.co

Maria Alejandra Lasso Perea, 202323516 , m.lassop@uniandes.edu.co

Maria Alejandra Sanabria Salazar, 202326786 , ma.sanabrias@uniandes.edu.co

Requerimiento 1

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Numero de ofertas, Código país, Nivel de experiencia
Salidas	Lista con la cantidad de ofertas de ese país y nivel de experiencia
Implementado (Sí/No)	Si se implementó, por Maria Alejandra Lasso, Maria Alejandra Sanabria y Diego Munévar

```python
def req_1(data_structs,cod_pais,lvl_exp):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    #Se obtiene el tamaño de la lista de trabajos y se crea una lista vacia para almacenar los trabajos que cumplen con el requerimiento
    size=lt.size(data_structs["jobs"])
    lista_nueva=lt.newList('ARRAY_LIST')
    #Se recorre la lista de trabajos y se añaden los trabajos que cumplen con el requerimiento a la lista de trabajos
    for i in range(1,size+1):
        if lt.getElement(data_structs["jobs"], i)["country_code"] == cod_pais and lt.getElement(data_structs["jobs"], i)["experience_level"] == lvl_exp:
                lt.addLast(lista_nueva, lt.getElement(data_structs["jobs"], i))
    #Se ordenan los trabajos de acuerdo a la fecha de publicación
    sorteado=merg.sort(lista_nueva, sort_dates)
    return sorteado
```

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Obtener el tamaño de la lista de trabajos	O(1)
Paso 2: Crear una lista nueva de DISClib	O(1)
Paso 3: Recorrer la lista por posiciones	O(N)
Paso 4: Ordenar la lista resultante por medio de Merge Sort	O(N*log(N))
TOTAL	***O(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).

Procesadores	**Intel Core i5-1135G7**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Parámetros usados:

Numero de ofertas: 100

Código de país: PL

Nivel de experiencia: mid

Entrada	Tiempo (ms)
10 pct	2385
20 pct	5224
30 pct	8614,7
50 pct	12361,08
60 pct	14862,31
70 pct	15552,48
80 pct	16926,71
large	17459

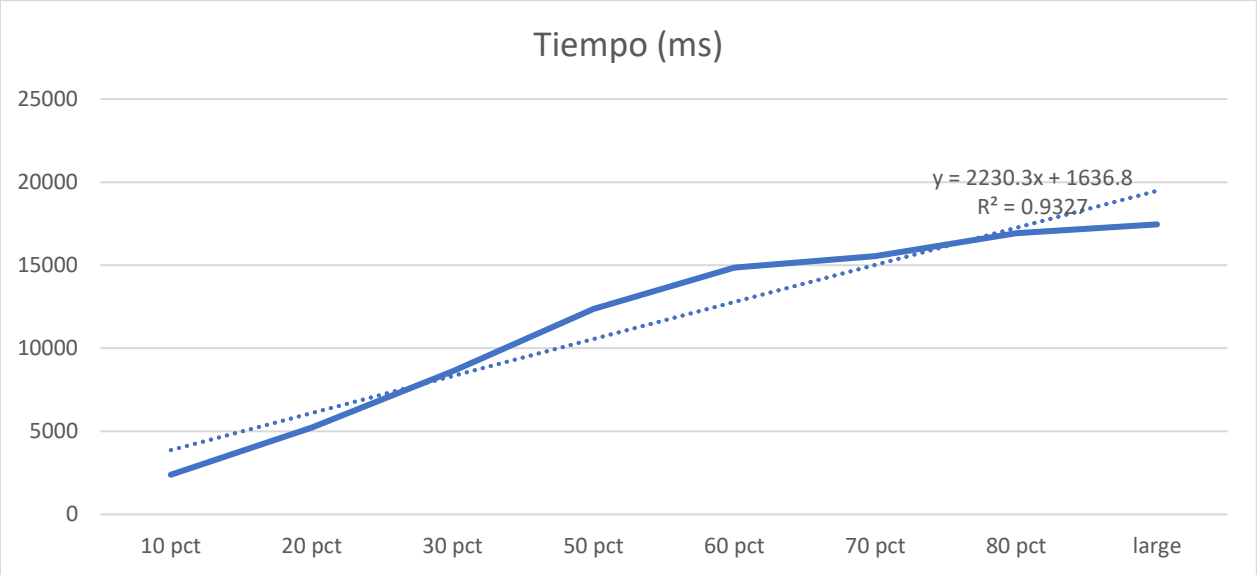
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10 pct	Lista de DISClib	2385
20 pct	Lista de DISClib	5224
30 pct	Lista de DISClib	8614,7
50 pct	Lista de DISClib	12361,08
60 pct	Lista de DISClib	14862,31
70 pct	Lista de DISClib	15552,48
80 pct	Lista de DISClib	16926,71

large	Lista de DISClib	17459

Graficas



Análisis

Debido a la complejidad O(N) del algoritmo, se ve una gráfica lineal, con un coeficiente de correlación del 0.93. En este caso, el coeficiente de correlación del 0.93 indica que hay una fuerte correlación positiva entre el tamaño de los datos y el tiempo de ejecución del algoritmo.

Dado que el coeficiente de correlación es relativamente alto (0.93), podemos inferir que la relación entre el tamaño de los datos y el tiempo de ejecución es bastante precisa. En otras palabras, podemos confiar en que el tiempo de ejecución del algoritmo aumentará de manera predecible a medida que aumente el tamaño de los datos.

Requerimiento 2

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El número (N) de ofertas a listar, Nombre completo de la empresa a consultar y Ciudad de la oferta
Salidas	Las últimas N ofertas de una empresa dado su nombre y la ciudad
Implementado (Sí/No)	Si se implementó , por Maria Alejandra Lasso, Maria Alejandra Sanabria y Diego Munévar

```python
def req_2(data_structs, nombre_empresa, ciudad):
    """
    Función que soluciona el requerimiento 2
    """
    #Se obtiene el tamaño de la lista de trabajos y se crea una lista vacia para almacenar los indices de los trabajos que cumplen con el requerimiento
    size=lt.size(data_structs["jobs"])
    lista_nueva=lt.newList('ARRAY_LIST')

    #Se recorre la lista de trabajos y se añaden los indices de los trabajos que cumplen con el requerimiento a la lista de indices
    for i in range(1,size+1):
        if lt.getElement(data_structs["jobs"], i)["company_name"].lower() == nombre_empresa.lower() and lt.getElement(data_structs["jobs"], i)["city"].lower() == ciudad.lower():
            lt.addLast(lista_nueva, lt.getElement(data_structs["jobs"], i))
    sorteado=merg.sort(lista_nueva, sort_dates)
    return sorteado

```

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1: Obtener el tamaño de toda la lista de trabajos	O(1)
Paso 2: Crear un array list de DISCLib	O(1)
Paso 3: Recorrer la lista dentro del rango	O(N)
Paso 4: Añadir el elemento a la lista nueva en formato Array List	O(1)
Paso 5: Ordenar la lista resultante por medio de merge sort	O(N*log(N))
TOTAL	***O(N*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).

Procesadores	**Intel Core i5-1135G7**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
10 pct	39,71
20 pct	62,71
30 pct	76,73
50 pct	91,4
60 pct	115,51
70 pct	114,43
80 pct	116,84

large	118,81

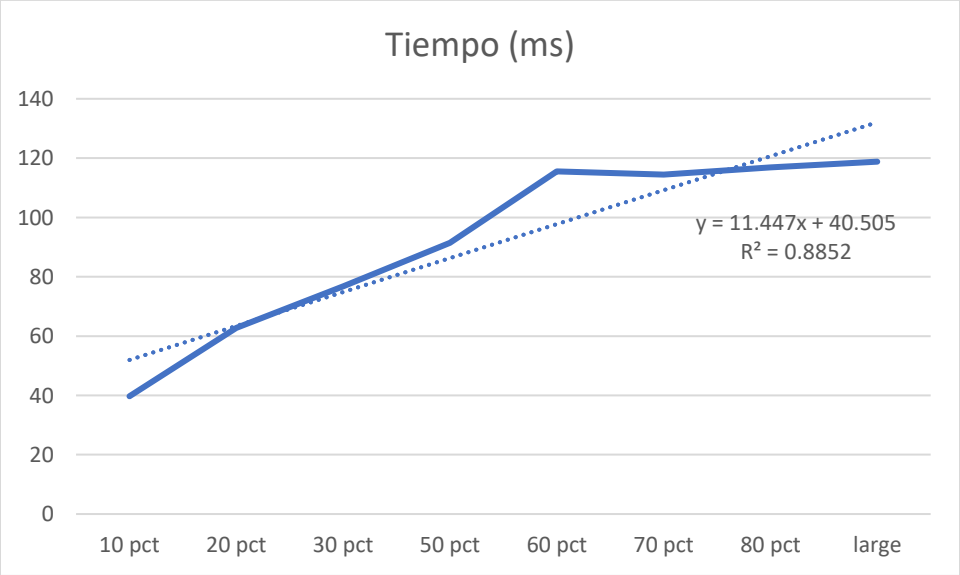
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10 pct	Lista de DISClib (Array_list)	39,71
20 pct	Lista de DISClib (Array_list)	62,71
30 pct	Lista de DISClib (Array_list)	76,73
50 pct	Lista de DISClib (Array_list)	91,4
60 pct	Lista de DISClib (Array_list)	115,51
70 pct	Lista de DISClib (Array_list)	114,43
80 pct	Lista de DISClib (Array_list)	116,84
large	Lista de DISClib (Array_list)	118,81

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Dentro del análisis del requerimiento #2, se puede ver que el algoritmo debería ser de comportamiento linearitmico, donde tiene un coeficiente de correlación respecto a una línea de tendencia lineal del 0.88, lo cual es alto, pero esto significa que los datos están un poco más dispersos con respecto a la línea de tendencia; esto podría significar que la gráfica se asemeja más a una línea de tendencia linearitmica y por ende se asemejaría más a ese tipo de curva.

Además, existe un pico en el 60% de los datos, lo cual muestra que entre el 50% y el 70% existió un aumento con respecto a la cantidad de datos que coincidían con lo ingresado por parámetro, por lo cual se ve una línea mucho más pronunciada en ese rango.

Requerimiento 3

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Nombre de la empresa, La fecha inicial y final del periodo a consultar
Salidas	Las ofertas de trabajo publicadas por una empresa en un rango de fechas dado
Implementado (Sí/No)	Si se implementó, por Maria Alejandra Lasso

```python
def cmp_req_3(data_1, data_2):
    if data_1["published_at"]==data_2["published_at"]:
        if data_1["company_name"]>data_2["company_name"]:
            return True
        else:
            return False
    elif data_1["published_at"]>data_2["published_at"]:
        return True
    else:
        return False

def req_3(data_structs, nombre_empresa, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 3
    """
    # Contador para el número total de ofertas
    n = 0
    size=lt.size(data_structs["jobs"])
    lista_n=lt.newList('ARRAY_LIST')
    # Inicializar contadores para cada nivel de experiencia
    ofertas_junior = 0
    ofertas_mid = 0
    ofertas_senior = 0
    # Filtrar el DataFrame por empresa y rango de fechas
    for i in range(1,size+1):
        job= lt.getElement(data_structs["jobs"],i)
        if (job["company_name"]) == nombre_empresa and fecha_inicial <= (job["published_at"]) <= fecha_final:
            lt.addLast(lista_n, job)
            n += 1
            # Calcular el número de ofertas por nivel de experticia
            if "junior" in job["experience_level"]:
                ofertas_junior += 1
            elif "mid" in job["experience_level"]:
                ofertas_mid += 1
            elif "senior" in job["experience_level"]:
                ofertas_senior += 1

    # Ordenar los datos por fecha y país (si es necesario)
    lista_n= merg.sort(lista_n, cmp_req_3)

    return {
        'total_ofertas': n,
        'ofertas_junior': ofertas_junior,
        'ofertas_mid': ofertas_mid,
        'ofertas_senior': ofertas_senior,
        'listado_ofertas': lista_n
    }

```

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1: Crear contadores, una lista nueva y sacar el tamaño de Jobs	O(1)
Paso 2: Filtrar el DataFrame por empresa y rango de fechas	O(n)
Paso 3: Calcular el número de ofertas por nivel de experiencia	O(1)
Paso 4: Ordenar los datos por fecha y país (se crea una función extra que ayude al proceso)	O(n log n)
TOTAL	***O(n 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).

Entrada	Tiempo (ms)
10 pct	1496.67
20 pct	2343.58
30 pct	3878.02
50 pct	6059.89
60 pct	7140.01
70 pct	7933.59
80 pct	9407.75
large	8798.82

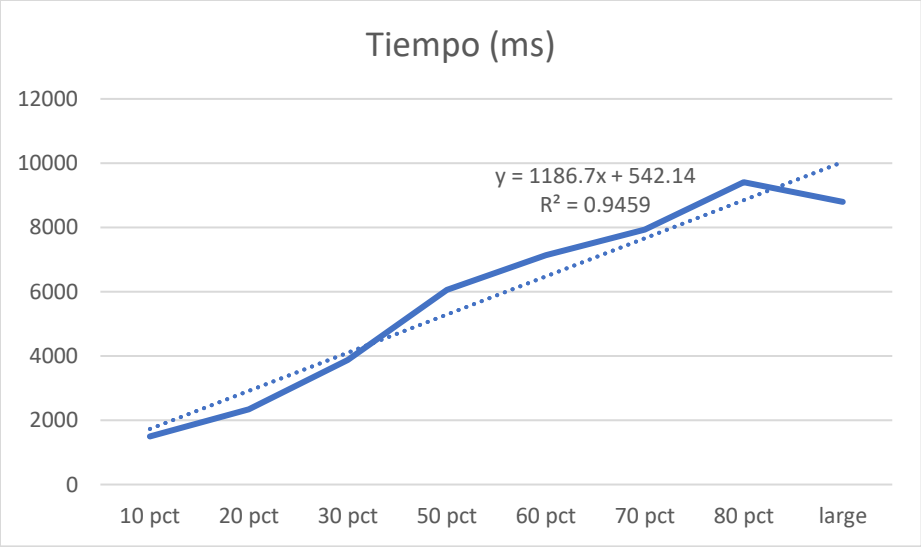
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10 pct	Lista de DISClib (Array_list)	1496.67
20 pct	Lista de DISClib (Array_list)	2343.58
30 pct	Lista de DISClib (Array_list)	3878.02
50 pct	Lista de DISClib (Array_list)	6059.89
60 pct	Lista de DISClib (Array_list)	7140.01
70 pct	Lista de DISClib (Array_list)	7933.59
80 pct	Lista de DISClib (Array_list)	9407.75
large	Lista de DISClib (Array_list)	8798.82

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Según los datos y funciones analizadas en el requerimiento 3, se observa que a medida que aumenta el tamaño de la entrada, hay un incremento en el tiempo de ejecución. La variabilidad en relación con la complejidad total del proceso sugiere que el algoritmo exhibe una complejidad que no es constante, pues el tiempo crece proporcional a n y a n log n, ya que es la combinación de lineal y logarítmico.

Basándonos en la información de la tabla, llegamos a la conclusión de que la complejidad del proceso es Linearitmica, expresada como O(n log n). Esto implica que, hay un bucle externo lineal que itera linealmente a través de `n` elementos y hay otro bucle interno logarítmico, en el cual `j` se duplica hasta que es mayor o igual a `n` y `j` aumenta exponencialmente, limitando iteraciones a $\log_2(n)$. Como se aprecia en la gráfica, se retrata el comportamiento de una complejidad de tipo Linearitmica que si bien es algo compleja de implementar, es una buena opción en grandes volúmenes de datos.

Requerimiento 4

Plantilla para el documentar y analizar cada uno de los requerimientos.

```
def req_4(data_structs,codigo,fecha_inicial,fecha_final):
                lista_empresas[lt.getElement(data_structs["jobs"], i)["company_name"]]= 1
            else:
                cantidad+=1
                lista_empresas[lt.getElement(data_structs["jobs"], i)["company_name"]]= cantidad
            if lt.getElement(data_structs["jobs"], i)["city"] in lista_ciudades :
                lista_ciudades[lt.getElement(data_structs["jobs"], i)["city"]]["cantidad"]+=1
                lt.addLast(lista_ciudades[lt.getElement(data_structs["jobs"], i)["city"]]["lista"],lt.getElement (data_structs["jobs"], i))
            else:
                lista_temporal=lt.newList('ARRAY_LIST')
                lista_ciudades[lt.getElement(data_structs["jobs"], i)["city"]]={"cantidad":1,"lista":lista_temporal}
    # El listado de ofertas publicadas ordenados cronológicamente por fecha y nombre de la empresa
    sorteado=merg.sort(lista_nueva, cmp_ofertas_by_fecha_y_nombre)
    sizee=len(lista_empresas)
    totalc= len(lista_ciudades.keys())
    lista_ciudad = lt.newList('ARRAY_LIST')
    salarios={}
    for i in range(1,lt.size(data_structs["employment"])+1):
        if lt.getElement(data_structs["employment"], i)["salary_from"]!="" and int(lt.getElement(data_structs["employment"], i)["salary_to"])!="":
            salarios[lt.getElement(data_structs["employment"], i)["id"]]=int(lt.getElement(data_structs["employment"], i)["salary_from"])+int(lt.getElement(data_structs["employment"], i)["salary_to"])
        elif lt.getElement(data_structs["employment"], i)["salary_from"]!="":
            salarios[lt.getElement(data_structs["employment"], i)["id"]]=int(lt.getElement(data_structs["employment"], i)["salary_from"])
        elif lt.getElement(data_structs["employment"], i)["salary_to"]!="":
            salarios[lt.getElement(data_structs["employment"], i)["id"]]=int(lt.getElement(data_structs["employment"], i)["salary_to"])
        else:
            salarios[lt.getElement(data_structs["employment"], i)["id"]]=0

    for i in lista_ciudades:
        lt.addLast(lista_ciudad, {"city":i, "cantidad":lista_ciudades[i]["cantidad"],"lista":lista_ciudades[i]["lista"]})
    for i in range(1,lt.size(lista_ciudad)+1):
        average=0
        for j in lt.iterator(lt.getElement(lista_ciudad, i)["lista"]):
            if j["id"] in salarios:
                average+=salarios[j["id"]]
        if average!=0:
            average=average/lt.size(lt.getElement(lista_ciudad, i)["lista"])
            lt.getElement(lista_ciudad, i)["average"]=average
        else:
            lt.getElement(lista_ciudad, i)["average"]=0
    sorteado2=merg.sort(lista_ciudad, cmp_req_4)


    mayor = lt.firstElement(sorteado2)
    menor= lt.lastElement(sorteado2)
    return sorteado,sizee,totalc,mayor,menor
```

```
def cmp_req_4(oferta1, oferta2):
    if oferta1["cantidad"]==oferta2["cantidad"]:
        if oferta1["average"]<oferta2["average"]:
            if oferta1["city"]<oferta2["city"]:
                return True
            else:
                return False
        elif oferta1["average"]>oferta2["average"]:
            return True
        else:
            return False
    elif oferta1["cantidad"]>oferta2["cantidad"]:
        return True
    else:
        return False

def cmp_ofertas_by_fecha_y_nombre(oferta1, oferta2):
    """
    Devuelve verdadero (True) si la fecha de la oferta 1 es menor que en la oferta ,
    en caso de que sean iguales se analiza la empresa de la oferta laboral, de lo contrario devuelva Falso

    Args:
        oferta1: información de la primera oferta laboral que incluye "company_name" y "published_at"
        oferta2: información de la segunda oferta laboral que incluye "company_name" y "published_at"
    """
    nombre1= oferta1["company_name"].lower()
    nombre2= oferta2["company_name"].lower()
    fecha1= oferta1["published_at"]
    fecha2= oferta2["published_at"]
    return (fecha1<fecha2) or (fecha1 == fecha2 and nombre1 < nombre2)
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Codigo del país, fecha inicial, fecha final.
Salidas	Total de ofertas en el país, total de empresas que publicaron al menos una oferta, Número total de ciudades del país de consulta en las que se publicaron ofertas, ciudad del país de consulta con mayor número de ofertas y su conteo, ciudad del país de consulta con menor número de ofertas (al menos una) y su conteo y el listado de ofertas publicadas ordenados cronológicamente por fecha y nombre de la empresa.
Implementado (Sí/No)	Si se implementó, por Alejandra Sanabria

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Obtener el tamaño de todos los datos de la lista.	O(1)
Paso 2: Creación de una nueva lista Disclib	O(1)
Paso 3: Creación de diccionarios	O(1)
Paso 4: Recorrer la lista de datos	O(N)
Paso 5: Organización de los datos	O(N)
Paso 6: Obtener el tamaño de la lista de empresas	O(1)
Paso 7: Organizar lista por empleo y salario	O(N)
Paso 8: Creación de lista de ciudades	O(1)
Paso 9: Iterar por promedio de salarios	O(N^2)
Paso 10: Buscar elemento inicial/final	O(1)
TOTAL	***O(N^2)***

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).

Datos de búsqueda:

- **País=PL, desde la**
- **Fecha de inicio (año-mes-día)2020-01-01,**
- **Hasta la fecha :(año-mes-día) 2024-01-01**

Tipo de datos: Array

Entrada	**Tiempo (ms)**
10	8492,67
20	21402,86
30	34078,17

40	41829,8
50	58483,54
60	55447,54
70	49771,97
80	50933,03
90	39828,63
Large	67643,69

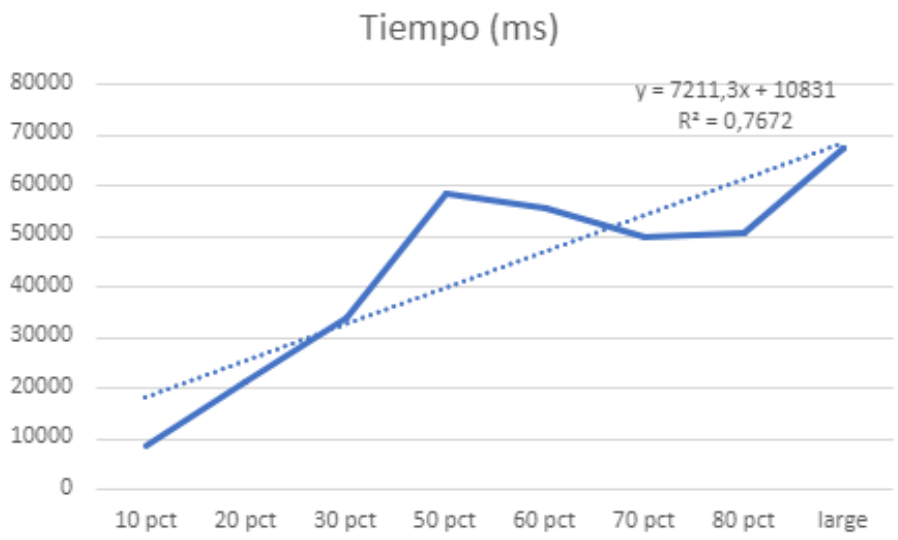
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	8492,67
20 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	21402,86
30 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	34078,17
50 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	58483,54
60 pct	Lista de Tupla de la lista, el tamaño, el mayor y menor elemento	55447,54
70 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	49771,97
80 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	50933,03
large	Tupla de la lista, el tamaño, el mayor y menor elemento	67643,69

Graficas

Las gráficas con la representación de las pruebas realizadas.



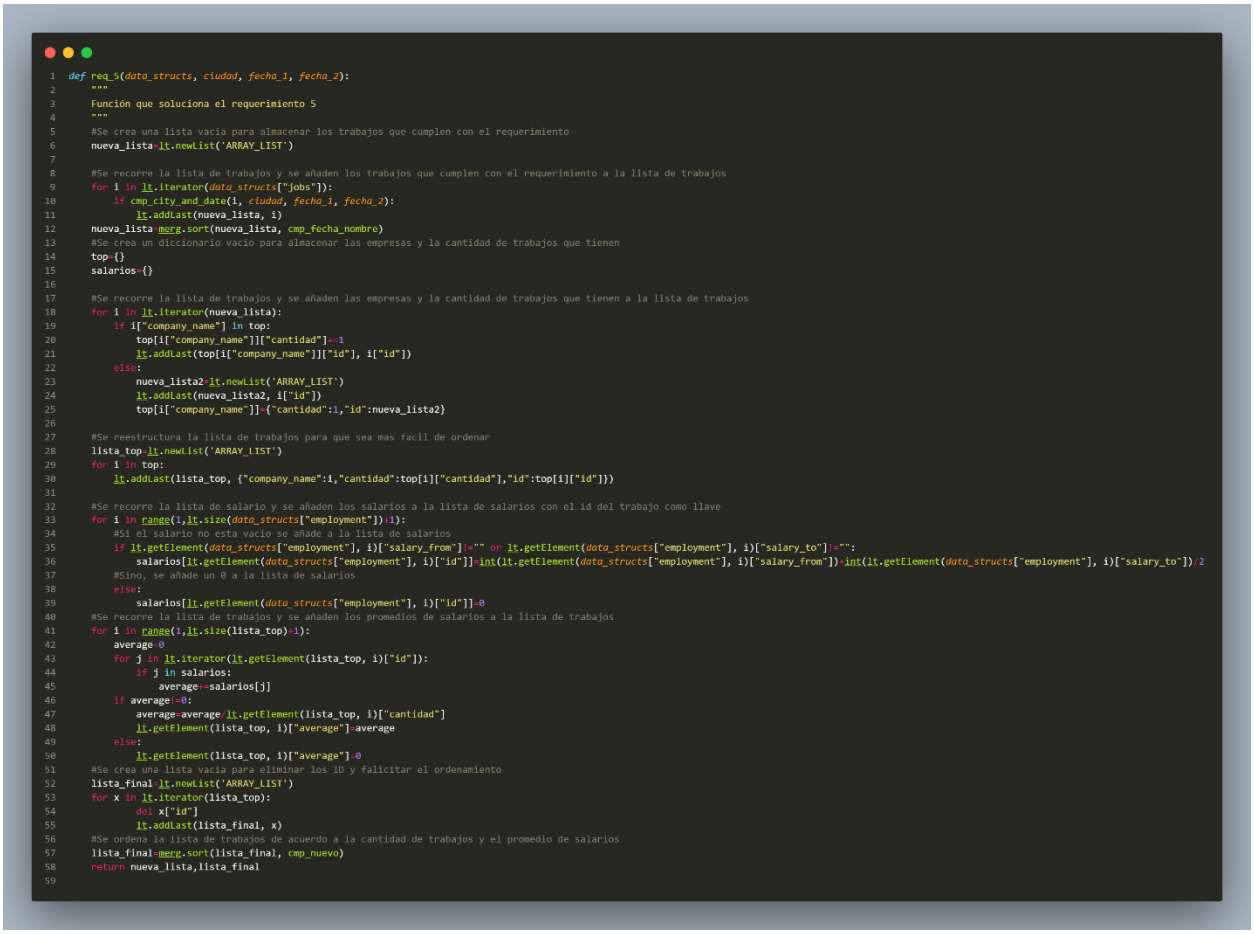
Análisis

Según los datos y funciones analizadas en el requerimiento 4, se observa que a medida que aumenta el tamaño de la entrada, hay un incremento en el tiempo de ejecución. La variabilidad en relación con la complejidad total del proceso sugiere que el algoritmo exhibe una complejidad que no es constante, sino que cambia en función del tamaño de la entrada.

Basándonos en la información de la tabla, llegamos a la conclusión de que la complejidad del proceso es cuadrática, expresada como O(N^2). Esto implica que el tiempo de ejecución del algoritmo guarda una relación proporcional con el cuadrado del tamaño de la entrada. Aunque la gráfica no refleje de manera absoluta la complejidad mencionada, se considera una mejor opción al momento de presentarlo al usuario, ya que puede ofrecerle resultados más favorables a su expectativa en lugar de resultados desfavorables a lo esperado.

Requerimiento 5

Plantilla para el documentar y analizar cada uno de los requerimientos.



```python
def req_5(data_structs, ciudad, fecha_1, fecha_2):
    """
    Función que soluciona el requerimiento 5
    """
    #Se crea una lista vacia para almacenar los trabajos que cumplen con el requerimiento
    nueva_lista=lt.newList('ARRAY_LIST')

    #Se recorre la lista de trabajos y se añaden los trabajos que cumplen con el requerimiento a la lista de trabajos
    for i in lt.iterator(data_structs["jobs"]):
        if cmp_city_and_date(i, ciudad, fecha_1, fecha_2):
            lt.addLast(nueva_lista, i)
    nueva_lista=merg.sort(nueva_lista, cmp_fecha_nombre)
    #Se crea un diccionario vacio para almacenar las empresas y la cantidad de trabajos que tienen
    top={}
    salarios={}

    #Se recorre la lista de trabajos y se añaden las empresas y la cantidad de trabajos que tienen a la lista de trabajos
    for i in lt.iterator(nueva_lista):
        if i["company_name"] in top:
            top[i["company_name"]]["cantidad"]+=1
            lt.addLast(top[i["company_name"]]["id"], i["id"])
        else:
            nueva_lista2=lt.newList('ARRAY_LIST')
            lt.addLast(nueva_lista2, i["id"])
            top[i["company_name"]]={"cantidad":1,"id":nueva_lista2}

    #Se reestructura la lista de trabajos para que sea mas facil de ordenar
    lista_top=lt.newList('ARRAY_LIST')
    for i in top:
        lt.addLast(lista_top, {"company_name":i,"cantidad":top[i]["cantidad"],"id":top[i]["id"]})

    #Se recorre la lista de salario y se añaden los salarios a la lista de salarios con el id del trabajo como llave
    for i in range(1,lt.size(data_structs["employment"])+1):
        #Si el salario no esta vacio se añade a la lista de salarios
        if lt.getElement(data_structs["employment"], i)["salary_from"]!="" or lt.getElement(data_structs["employment"], i)["salary_to"]!="":
            salarios[lt.getElement(data_structs["employment"], i)["id"]]=int(lt.getElement(data_structs["employment"], i)["salary_from"])+int(lt.getElement(data_structs["employment"], i)["salary_to"])/2
        #Sino, se añade un 0 a la lista de salarios
        else:
            salarios[lt.getElement(data_structs["employment"], i)["id"]]=0
    #Se recorre la lista de trabajos y se añaden los promedios de salarios a la lista de trabajos
    for i in range(1,lt.size(lista_top)+1):
        average=0
        for j in lt.iterator(lt.getElement(lista_top, i)["id"]):
            if j in salarios:
                average+=salarios[j]
        if average!=0:
            average=average/lt.getElement(lista_top, i)["cantidad"]
            lt.getElement(lista_top, i)["average"]=average
        else:
            lt.getElement(lista_top, i)["average"]=0
    #Se crea una lista vacia para eliminar los ID y falicitar el ordenamiento
    lista_final=lt.newList('ARRAY_LIST')
    for x in lt.iterator(lista_top):
            del x["id"]
            lt.addLast(lista_final, x)
    #Se ordena la lista de trabajos de acuerdo a la cantidad de trabajos y el promedio de salarios
    lista_final=merg.sort(lista_final, cmp_nuevo)
    return nueva_lista,lista_final
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Nombre de la ciudad, fecha inicial, fecha final.
Salidas	Total de ofertas publicadas, total de empresas que publicaron por lo menos una oferta, empresa con mayor número de ofertas y su conteo, empresa con menor número de ofertas (al menos una) y su conteo y listado de ofertas publicadas ordenadas cronológicamente por fecha y nombre de la empresa.
Implementado (Sí/No)	Si se implementó, por Diego Munevar.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Crear un array list de DISClib	O(1)
Paso 2: Iterar sobre toda la lista de trabajos	O(N)
Paso 3: Añadir un elemento a la última posición en un Array	O(1)
Paso 4: Creación de diccionarios	O(1)

Paso 5: Iterar sobre la lista creada	O(N)
Paso 6: Añadir un elemento a la última posición en Array list	O(1)
Paso 7: Crear un Array List de DISCLib	O(1)
Paso 8: Iterar sobre un diccionario	O(N)
Paso 9: Añadir un elemento a la última posición de un array list	O(1)
Paso 10: Recorrer la lista de salarios	O(N)
Paso 11: Calcular el promedio a partir de datos obtenidos de un array	O(1)
Paso 12: Asignar un valor a un diccionario	O(1)
Paso 13: Recorrer la lista de trabajos ordenados y la lista de IDs	O(N^2)
Paso 14: Añadir un elemento a un diccionario	O(1)
Paso 15: Crear un nuevo array list	O(1)
Paso 16: Iterar sobre la lista de trabajos ordenados	O(N)
Paso 17: Añadir un elemento a la posición final de un array list	O(1)
Paso 18: Ordenar la lista por medio de merge sort	O(N*log(N)
TOTAL	***O(N^2)***

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).

Procesadores	**Intel Core i5-1135G7**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Entrada	Tiempo (ms)
10 pct	1111.44
20 pct	1808.85
30 pct	2987.59
50 pct	4174.02
60 pct	5393.35
70 pct	5621.39
80 pct	5672.97
large	5811.22

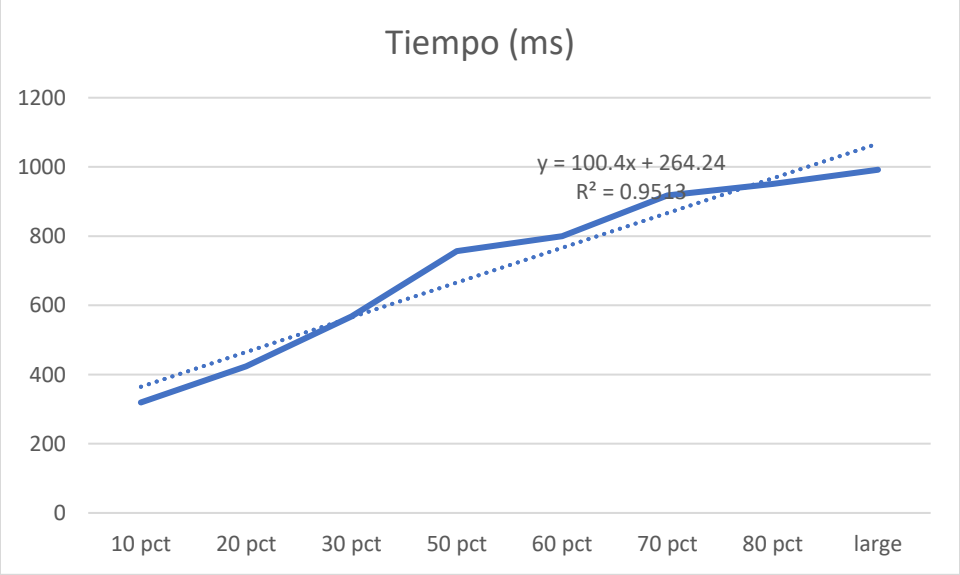
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	1111.44
20 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	1808.85
30 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	2987.59
50 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	4174.02
60 pct	Lista de Tupla de la lista, el tamaño, el mayor y menor elemento	5393.35
70 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	5621.39
80 pct	Tupla de la lista, el tamaño, el mayor y menor elemento	5672.97
large	Tupla de la lista, el tamaño, el mayor y menor elemento	5811.22

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

En el análisis del requerimiento #5, podemos observar que, en la primera tabla, se ha determinado que el peor caso de complejidad el algoritmo es cuadrático, O(N^2). Esto implica que el tiempo de ejecución del algoritmo crece cuadráticamente con el tamaño de la entrada.

Sin embargo, al observar la gráfica y cada tiempo de ejecución con respecto al tamaño de la entrada, se ha identificado que se puede representar como un algoritmo de complejidad lineal, O(N). Esto sugiere que, aunque en el peor caso la complejidad es cuadrática, en la práctica y para estos casos específicos analizados, el algoritmo tiende a comportarse de manera eficiente y proporcional al tamaño de la entrada lineal.

Es importante destacar que, aunque el peor caso sea cuadrático, la eficiencia en el rendimiento real puede variar dependiendo de las condiciones específicas de uso y de las instancias reales de entrada.

Requerimiento 6

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El número (N) de ciudades, código del país, nivel de experticia de las ofertas de interés, fecha inicial y fecha final.
Salidas	Total de ciudades que cumplen con las condiciones de la consulta, total de empresas que cumplen con las condiciones de la consulta, total de ofertas publicadas que cumplen con las condiciones de la consulta, promedio del salario ofertado de todas las ofertas que cumplen con las condiciones, nombre de la ciudad con mayor cantidad de ofertas de empleos y su conteo, nombre de la ciudad con menor cantidad de ofertas de empleos y su conteo, y listado de las ciudades ordenadas por el número de ofertas publicadas y nombre de la ciudad
Implementado (Sí/No)	Si se implementó y por Maria Alejandra Lasso, Diego Munévar y Maria Alejandra Sanabria.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Crear un array list de DISClib	O(1)
Paso 2: Iterar sobre toda la lista de trabajos	O(N)
Paso 3: Añadir un elemento a la última posición en un Array	O(1)
Paso 4: Creación de diccionarios	O(1)
Paso 5: Iterar sobre la lista creada	O(N)
Paso 6: Añadir un elemento a la última posición en Array list	O(1)
Paso 7: Crear un Array List de DISCLib	O(1)

Paso 8: Iterar sobre el diccionario de ciudades	O(N)
Paso 9: Añadir un elemento a la última posición en Array list	O(1)
Paso 10: Ordenar la lista de ciudades	O(N*log(N))
Paso 11: Crear un Array List de DISCLib	O(1)
Paso 12: Iterar sobre un rango de números	O(N)
Paso 13: Añadir un elemento a la última posición en Array list	O(1)
Paso 14: Iterar sobre la lista de trabajos	O(N)
Paso 15: Añadir un elemento a la última posición en Array list	O(1)
Paso 16: Iterar sobre el diccionario de ciudades	O(N)
Paso 17: Iterar sobre el diccionario de empresas	O(N)
Paso 18: Iterar sobre la lista de trabajos	O(N)
Paso 19: Añadir un elemento a la última posición en Array list	O(1)
Paso 20: Iterar sobre el diccionario de ciudades	O(N)
Paso 21: Iterar sobre el diccionario de empresas	O(N)
Paso 22: Iterar sobre la lista de trabajos	O(N)
Paso 23: Añadir un elemento a la última posición en Array list	O(1)
Paso 24: Iterar sobre el diccionario de ciudades	O(N)
Paso 25: Iterar sobre el diccionario de empresas	O(N)
Paso 26: Iterar sobre la lista de trabajos	O(N)
Paso 27: Añadir un elemento a la última posición en Array list	O(1)
Paso 28: Iterar sobre el diccionario de ciudades	O(M)
Paso 29: Añadir un elemento a la última posición en Array list	O(1)
TOTAL:	***O(N*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).

Procesadores	**Intel Core i5-11300H**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Ingrese el número de ciudades que desea ver: 10

Ingrese el nivel de experticia que desea ver (junior, mid, senior): mid

Ingrese la fecha de inicio en formato YYYY-MM-DD: 2022-01-01

Ingrese la fecha de fin en formato YYYY-MM-DD: 2024-01-01

Desea ver los datos de un pais en especifico (S/N): N

Entrada	Tiempo (ms)
10 pct	122.02
20 pct	321.07
30 pct	439.7
50 pct	597.13
60 pct	652.22
70 pct	666.23
80 pct	704.8
large	776.33

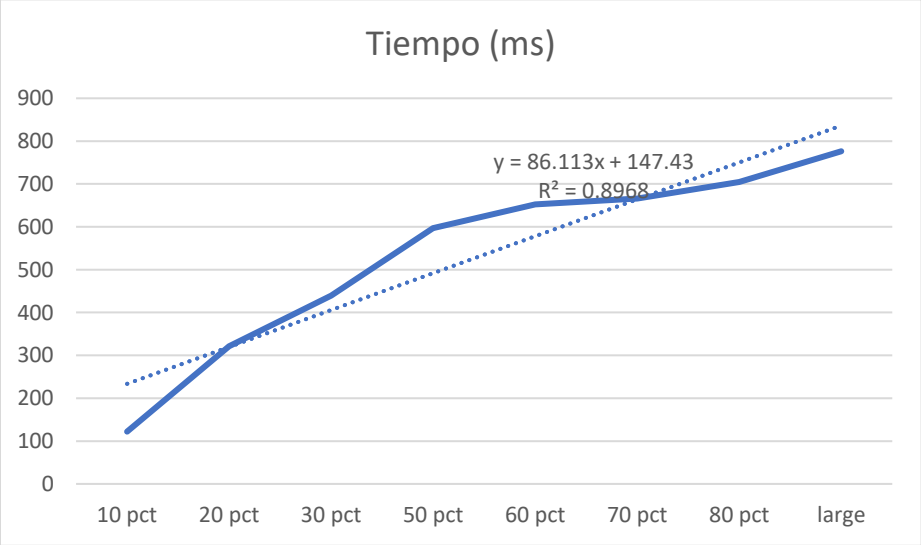
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
10 pct	Lista de DISClib	122.02
1220	Lista de DISClib	321.07
30 pct	Lista de DISClib	439.7
50 pct	Lista de DISClib	597.13
60 pct	Lista de DISClib	652.22
70 pct	Lista de DISClib	666.23
80 pct	Lista de DISClib	704.8
large	Lista de DISClib	776.33

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al examinar detenidamente el requerimiento 6, notamos que, según la tabla de complejidad, el algoritmo muestra una complejidad linearitmica, representada como O(N*log(N)). Esta notación sugiere que la tasa de crecimiento del tiempo de ejecución está en proporción logarítmica al tamaño de la entrada, además la evidencia de esta complejidad se refleja no solo en los datos tabulados sino también en la información visualizada en la gráfica.

Esta complejidad particular es especialmente interesante ya que combina elementos lineales y logarítmicos, lo que puede indicar una eficiencia significativa en el manejo de tamaños de entrada más grandes en comparación con enfoques lineales o cuadráticos.

Requerimiento 7

Plantilla para el documentar y analizar cada uno de los requerimientos.

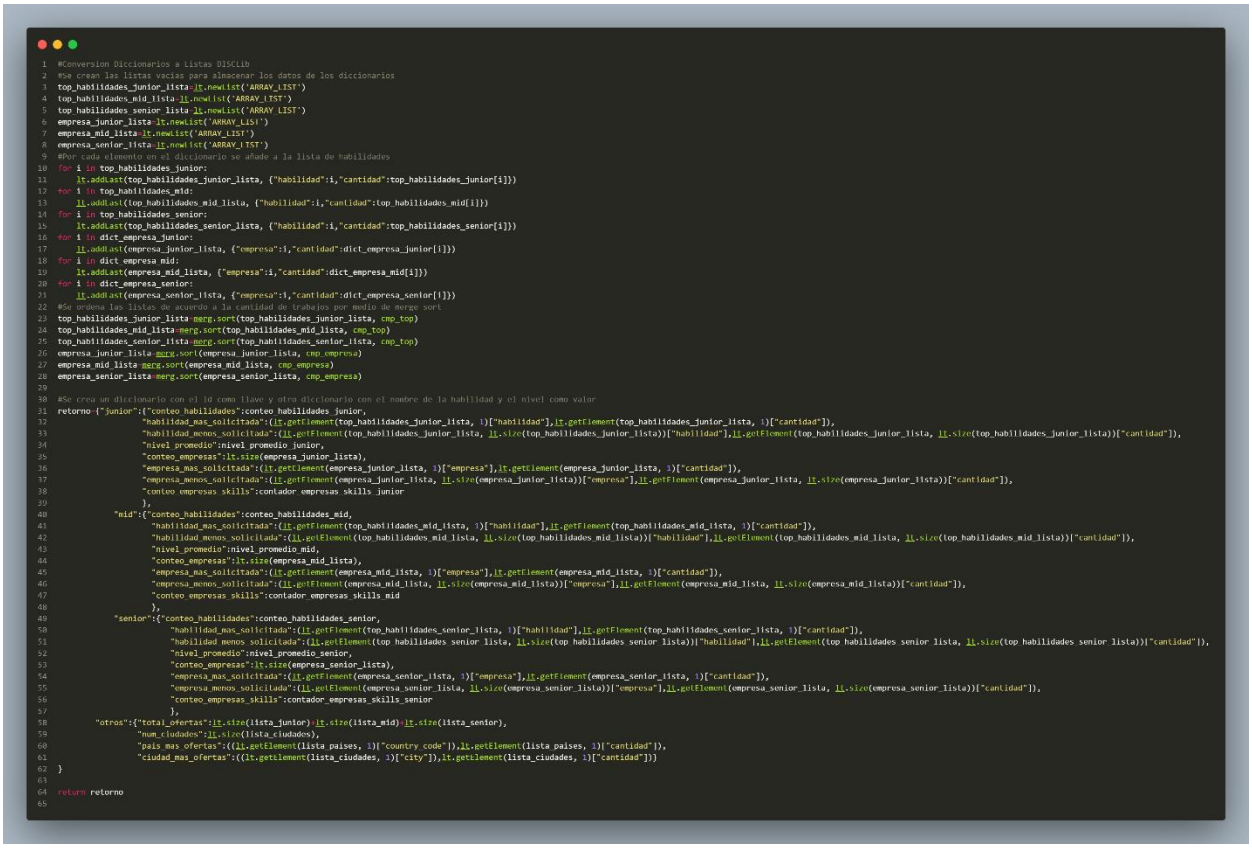
```python
def req_7(data_structs,num_paises, fecha_ini, fecha_fin):
    """
    Función que soluciona el requerimiento 7
    """
    #======FILTRADO Y DIVISION DE DATOS=======
    #Obtener los datos de los trabajos entre las fechas dadas
    datos = data_structs["jobs"]
    paises = lt.newList('ARRAY_LIST')
    dict_pais = {"paises": {}}
    #Se itera la lista de trabajos por medio de un iterador
    for i in lt.iterator(datos):
        #Si la fecha de publicación del trabajo esta entre las fechas dadas se añade a la lista de trabajos
        if fecha_a_iso(i["published_at"]) >= fecha_ini and fecha_a_iso(i["published_at"]) <= fecha_fin:
            lt.addLast(paises, i)
            #Si el pais del trabajo esta en el diccionario de paises se añade uno a la cantidad de trabajos en ese pais
            if i["country_code"] in dict_pais["paises"]:
                dict_pais["paises"][i["country_code"]]["cantidad"] += 1
                #Si la ciudad del trabajo esta en el diccionario de ciudades se añade uno a la cantidad de trabajos en esa ciudad
                if i["city"] in dict_pais["paises"][i["country_code"]]["ciudades"]:
                    dict_pais["paises"][i["country_code"]]["ciudades"][i["city"]] += 1
                #Sino, se añade la ciudad al diccionario de ciudades
                else:
                    dict_pais["paises"][i["country_code"]]["ciudades"][i["city"]] = 1
            #Sino, se añade el pais al diccionario de paises
            else:
                dict_pais["paises"][i["country_code"]] = {"cantidad": 1, "ciudades": {i["city"]: 1}}

    #Si el dato ingresado es mayor al tamaño de la lista de paises se iguala el dato al tamaño de la lista de paises
    if num_paises>len(dict_pais["paises"]):
        num_paises=len(dict_pais["paises"])-1

    #Se crea una lista vacia para almacenar los paises en forma de diccionario con el codigo del pais y la cantidad de trabajos
    lista_paises=lt.newList('ARRAY_LIST')

    #Separar los datos en lista con numero de ofertas por ciudad con una estructura de diccionario {"country_code":PL,"cantidad":1}
    for i in dict_pais["paises"]:
        lt.addLast(lista_paises, {"country_code":i,"cantidad":dict_pais["paises"][i]["cantidad"]})

    #Se ordena la lista de paises de acuerdo a la cantidad de trabajos y se obtiene la sublista con el numero de paises ingresado
    lista_paises=lt.subList(merg.sort(lista_paises, cmp_paises),1,num_paises)

    #Se crea una lista vacia para almacenar las ciudades en forma de diccionario {"city":Warsaw,"cantidad":1}
    lista_ciudades=lt.newList('ARRAY_LIST')
    for i in lt.iterator(lista_paises):
        for j in dict_pais["paises"][i["country_code"]]["ciudades"]:
            lt.addLast(lista_ciudades, {"city":j,"cantidad":dict_pais["paises"][i["country_code"]]["ciudades"][j]})

    #Se ordena la lista de ciudades de acuerdo a la cantidad de trabajos
    lista_ciudades=merg.sort(lista_ciudades, cmp_ciudades)

    #Se crea unas listas vacias para almacenar los trabajos de acuerdo al nivel de experiencia
    lista_junior=lt.newList('ARRAY_LIST')
    lista_mid=lt.newList('ARRAY_LIST')
    lista_senior=lt.newList('ARRAY_LIST')

```

```
    #Separar los datos en listas de acuerdo al nivel de experiencia
    for i in lt.iterator(paises):
        #Se recorre la lista de paises recortada
        for j in lt.iterator(lista_paises):
            #Si el nivel de experiencia del trabajo es junior y el codigo del pais es igual al codigo del pais de la lista de paises se añade a la lista de trabajos junior
            if i["experience_level"]=="junior" and j["country_code"]==i["country_code"]:
                lt.addLast(lista_junior, i)
            #Si el nivel de experiencia del trabajo es mid y el codigo del pais es igual al codigo del pais de la lista de paises se añade a la lista de trabajos mid
            elif i["experience_level"]=="mid" and j["country_code"]==i["country_code"]:
                lt.addLast(lista_mid, i)
            #Si el nivel de experiencia del trabajo es senior y el codigo del pais es igual al codigo del pais de la lista de paises se añade a la lista de trabajos senior
            else:
                if j["country_code"]==i["country_code"]:
                    lt.addLast(lista_senior, i)

    #====CONVERSION DE SKILLS A DICCIONARIO====
    datos_skills=data_structs["skills"]
    #Crear un diccionario con el id como llave y otro diccionario con el nombre de la habilidad y el nivel como valor
    dict_skills_id={}
    #Se itera la lista de habilidades por medio de un iterador
    for i in lt.iterator(datos_skills):
        #Se crea una lista temporal para almacenar los nombres y los niveles de las habilidades
        lista_tem_id=lt.newList('ARRAY_LIST')
        #Si el id ya esta en el diccionario se añade la habilidad a la lista de habilidades
        if i["id"] in dict_skills_id:
            lt.addLast(dict_skills_id[i["id"]],{"name":i["name"],"level":i["level"]})
        #Sino, se añade el id al diccionario y se añade la habilidad a la lista de habilidades
        else:
            lt.addLast(lista_tem_id,{"name":i["name"],"level":i["level"]})
            dict_skills_id[i["id"]]=lista_tem_id

    #====CONVERSION DE MULTILOCATION A DICCIONARIO===
    datos_multilocation=data_structs["multilocation"]
    #Crear un diccionario con el id como llave y otro diccionario con la ciudad y la cantidad de trabajos como valor
    dict_multilocation={}
    for i in lt.iterator(datos_multilocation):
        lista_tem_id=lt.newList('ARRAY_LIST')
        if i["id"] in dict_multilocation:
            lt.addLast(dict_multilocation[i["id"]],i["city"])
        else:
            lt.addLast(lista_tem_id,i["city"])
            dict_multilocation[i["id"]]=lista_tem_id

```

```python
    #===CREACION INFORMACION PARA MOSTRAR====

    #====JUNIOR=====
    #Se crean las variables para almacenar los datos de los trabajos junior
    #Se cuentan el numero de habilidades
    conteo_habilidades_junior=0
    #Se obtienen las habilidades más y menos solicitadas
    top_habilidades_junior={}
    #Se obtiene una sumatoria de los niveles de las habilidades para despues dividir sobre la cantidad de habilidades
    nivel_promedio_junior=0
    #Se obtienen las empresas más y menos solicitadas
    dict_empresa_junior={}
    #Se obtienen las empresas con trabajos con diferentes ID
    empresas_sede_junior_id={}
    #Se obtienen las empresas que ya se han añadido
    empresas_sede_junior={}
    #Se añade un contador a las empresas de nivel junior
    contador_empresas_skills_junior=0

    #Se recorre toda la lista junior
    for i in range(1,lt.size(lista_junior)+1):
        #Se obtiene el elemento del diccionario de habilidades con el id del trabajo y se itera sobre la lista resultante
        for j in lt.iterator(dict_skills_id[lt.getElement(lista_junior, i)["id"]]):
            #Se cuenta una habilidad
            conteo_habilidades_junior+=1
            #Si la habilidad ya esta en el diccionario se añade uno a la cantidad de veces que se ha solicitado
            if j["name"] in top_habilidades_junior:
                top_habilidades_junior[j["name"]]+=1
            #Sino, se añade la habilidad al diccionario
            else:
                top_habilidades_junior[j["name"]]=1
            #Se suma el nivel de la habilidad al nivel promedio
            nivel_promedio_junior+=int(j["level"])

        #Si la empresa ya esta en el diccionario se añade uno a la cantidad de veces que se ha solicitado
        if lt.getElement(lista_junior, i)["company_name"] in dict_empresa_junior:
            dict_empresa_junior[lt.getElement(lista_junior, i)["company_name"]]+=1
        #Sino, se añade la empresa al diccionario
        else:
            dict_empresa_junior[lt.getElement(lista_junior, i)["company_name"]]=1

        #Por a busqueda del ID en el diccionario de multilocation se itera en la lista de multilocation resultante
        for j in lt.iterator(dict_multilocation[lt.getElement(lista_junior, i)["id"]]):
            #Si la empresa no esta en el diccionario de IDs y la empresa no esta en el diccionario de empresas se añade la empresa al diccionario de empresas
            if j not in empresas_sede_junior_id and lt.getElement(lista_junior, i)["company_name"] not in empresas_sede_junior:
                empresas_sede_junior[j]=lt.getElement(lista_junior, i)["company_name"]
                empresas_sede_junior_id[j]=lt.getElement(lista_junior, i)["id"]
                #Se añade 1 al contador de empresas
                contador_empresas_skills_junior+=1

    #Se obtiene el promedio de los niveles de las habilidades
    nivel_promedio_junior=nivel_promedio_junior/conteo_habilidades_junior

```

```
    #=======MID=======
    #Se crean las variables para almacenar los datos de los trabajos mid
    #Se cuentan el numero de habilidades
    conteo_habilidades_mid=0
    #Se obtienen las habilidades más y menos solicitadas
    top_habilidades_mid={}
    #Se obtiene una sumatoria de los niveles de las habilidades para despues dividir sobre la cantidad de habilidades
    nivel_promedio_mid=0
    #Se obtienen las empresas más y menos solicitadas
    dict_empresa_mid={}
    #Se obtienen las empresas con trabajos con diferentes ID
    empresas_sede_mid_id={}
    #Se obtienen las empresas que ya se han añadido
    empresas_sede_mid={}
    #Se añade un contador a las empresas de nivel mid
    contador_empresas_skills_mid=0

    #Se recorre toda la lista mid
    for i in range(1,lt.size(lista_mid)+1):
        #Se obtiene el elemento del diccionario de habilidades con el id del trabajo y se itera sobre la lista resultante
        for j in lt.iterator(dict_skills_id[lt.getElement(lista_mid, i)["id"]]):
            #Se cuenta una habilidad
            conteo_habilidades_mid+=1
            #Si la habilidad ya esta en el diccionario se añade uno a la cantidad de veces que se ha solicitado
            if j["name"] in top_habilidades_mid:
                top_habilidades_mid[j["name"]]+=1
            #Sino, se añade la habilidad al diccionario
            else:
                top_habilidades_mid[j["name"]]=1
            nivel_promedio_mid+=int(j["level"])

        #Si la empresa ya esta en el diccionario se añade uno a la cantidad de veces que se ha solicitado
        if lt.getElement(lista_mid, i)["company_name"] in dict_empresa_mid:
            dict_empresa_mid[lt.getElement(lista_mid, i)["company_name"]]+=1
        #Sino, se añade la empresa al diccionario
        else:
            dict_empresa_mid[lt.getElement(lista_mid, i)["company_name"]]=1

        #Se recorre la lista de multilocation resultante de obtener el ID
        for j in lt.iterator(dict_multilocation[lt.getElement(lista_mid, i)["id"]]):
            #Si no esta en el diccionario de IDs y no esta en el diccionario de empresas se añade la empresa a ambos
            if j not in empresas_sede_mid_id and lt.getElement(lista_mid, i)["company_name"] not in empresas_sede_mid:
                empresas_sede_mid[j]=lt.getElement(lista_mid, i)["company_name"]
                empresas_sede_mid_id[j]=lt.getElement(lista_mid, i)["id"]
                #Se añade 1 al contador de empresas
                contador_empresas_skills_mid+=1
    #Se calcula el promedio con la sumatoria de los niveles de las habilidades y la cantidad de habilidades
    nivel_promedio_mid=nivel_promedio_mid/conteo_habilidades_mid

```

```
    #=======SENIOR=======
    #Contar el numero de habilidades de nivel senior
    conteo_habilidades_senior=0
    #Obtener las habilidades mas y menos solicitadas
    top_habilidades_senior={}
    #Obtener una sumatoria de los niveles de las habilidades para despues dividir sobre la cantidad de habilidades
    nivel_promedio_senior=0
    #Obtener las empresas mas y menos solicitadas
    dict_empresa_senior={}
    #Obtener las empresas con trabajos con diferentes ID
    empresas_sede_senior_id={}
    #Obtener las empresas que ya se han añadido
    empresas_sede_senior={}
    #Añadir un contador a las empresas de nivel senior
    contador_empresas_skills_senior=0
    #Recorrer toda la lista senior y obtener los datos
    for i in range(1,lt.size(lista_senior)+1):
        #Obtener el elemento del diccionario de habilidades con el id del trabajo y recorrer la lista resultante
        for j in lt.iterator(dict_skills_id[lt.getElement(lista_senior, i)["id"]]):
            #Contar una habilidad de nivel senior al contador
            conteo_habilidades_senior+=1
            #Si la habilidad ya esta en el diccionario se añade uno a la cantidad de veces que se ha solicitado
            if j["name"] in top_habilidades_senior:
                top_habilidades_senior[j["name"]]+=1
            #Sino, se añade la habilidad al diccionario
            else:
                top_habilidades_senior[j["name"]]=1
            nivel_promedio_senior+=int(j["level"])
        #Si la empresa ya esta en el diccionario se añade uno a la cantidad de veces que se ha solicitado
        if lt.getElement(lista_senior, i)["company_name"] in dict_empresa_senior:
            dict_empresa_senior[lt.getElement(lista_senior, i)["company_name"]]+=1
        #Sino, se añade la empresa al diccionario de empresas
        else:
            dict_empresa_senior[lt.getElement(lista_senior, i)["company_name"]]=1
        #Se recorre la lista de multilocation resultante de obtener el ID en el diccionario de multilocation
        for j in lt.iterator(dict_multilocation[lt.getElement(lista_senior, i)["id"]]):
            #Si no esta en el diccionario de IDs y no esta en el diccionario de empresas se añade la empresa a ambos
            if j not in empresas_sede_senior_id and lt.getElement(lista_senior, i)["company_name"] not in empresas_sede_senior:
                empresas_sede_senior[j]=lt.getElement(lista_senior, i)["company_name"]
                empresas_sede_senior_id[j]=lt.getElement(lista_senior, i)["id"]
                #Se añade 1 al contador de empresas
                contador_empresas_skills_senior+=1
    #Se calcula el promedio con la sumatoria de los niveles de las habilidades y la cantidad de habilidades
    nivel_promedio_senior=nivel_promedio_senior/conteo_habilidades_senior
```



```
#Conversion Diccionarios a Listas DISClib
#Se crean las listas vacias para almacenar los datos de los diccionarios
top_habilidades_junior_lista=lt.newlist('ARRAY_LIST')
top_habilidades_mid_lista=lt.newlist('ARRAY_LIST')
top_habilidades_senior_lista=lt.newlist('ARRAY_LIST')
empresa_junior_lista=lt.newlist('ARRAY_LIST')
empresa_mid_lista=lt.newlist('ARRAY_LIST')
empresa_senior_lista=lt.newlist('ARRAY_LIST')
#Por cada elemento en el diccionario se añade a la lista de habilidades
for i in top_habilidades_junior:
    lt.addLast(top_habilidades_junior_lista, {"habilidad":i,"cantidad":top_habilidades_junior[i]})
for i in top_habilidades_mid:
    lt.addLast(top_habilidades_mid_lista, {"habilidad":i,"cantidad":top_habilidades_mid[i]})
for i in top_habilidades_senior:
    lt.addLast(top_habilidades_senior_lista, {"habilidad":i,"cantidad":top_habilidades_senior[i]})
for i in dict_empresa_junior:
    lt.addLast(empresa_junior_lista, {"empresa":i,"cantidad":dict_empresa_junior[i]})
for i in dict_empresa_mid:
    lt.addLast(empresa_mid_lista, {"empresa":i,"cantidad":dict_empresa_mid[i]})
for i in dict_empresa_senior:
    lt.addLast(empresa_senior_lista, {"empresa":i,"cantidad":dict_empresa_senior[i]})
#Se ordena las listas de acuerdo a la cantidad de trabajos por medio de merge sort
top_habilidades_junior_lista=merg.sort(top_habilidades_junior_lista, cmp_top)
top_habilidades_mid_lista=merg.sort(top_habilidades_mid_lista, cmp_top)
top_habilidades_senior_lista=merg.sort(top_habilidades_senior_lista, cmp_top)
empresa_junior_lista=merg.sort(empresa_junior_lista, cmp_empresa)
empresa_mid_lista=merg.sort(empresa_mid_lista, cmp_empresa)
empresa_senior_lista=merg.sort(empresa_senior_lista, cmp_empresa)

#Se crea un diccionario con el id como llave y otro diccionario con el nombre de la habilidad y el nivel como valor
retorno={"junior":{"conteo habilidades":conteo habilidades_junior,
                "habilidad_mas_solicitada":(lt.getElement(top_habilidades_junior_lista, 1)["habilidad"],lt.getElement(top_habilidades_junior_lista, 1)["cantidad"]),
                "habilidad_menos_solicitada":(lt.getElement(top_habilidades_junior_lista, lt.size(top_habilidades_junior_lista))["habilidad"],lt.getElement(top_habilidades_junior_lista, lt.size(top_habilidades_junior_lista))["cantidad"]),
                "nivel_promedio":nivel_promedio_junior,
                "conteo_empresas":lt.size(empresa_junior_lista),
                "empresa_mas_solicitada":(lt.getElement(empresa_junior_lista, 1)["empresa"],lt.getElement(empresa_junior_lista, 1)["cantidad"]),
                "empresa_menos_solicitada":(lt.getElement(empresa_junior_lista, lt.size(empresa_junior_lista))["empresa"],lt.getElement(empresa_junior_lista, lt.size(empresa_junior_lista))["cantidad"]),
                "conteo empresas skills":contador empresas skills junior
                },
            "mid":{"conteo_habilidades":conteo_habilidades_mid,
                   "habilidad_mas_solicitada":(lt.getElement(top_habilidades_mid_lista, 1)["habilidad"],lt.getElement(top_habilidades_mid_lista, 1)["cantidad"]),
                   "habilidad_menos_solicitada":(lt.getElement(top_habilidades_mid_lista, lt.size(top_habilidades_mid_lista))["habilidad"],lt.getElement(top_habilidades_mid_lista, lt.size(top_habilidades_mid_lista))["cantidad"]),
                   "nivel_promedio":nivel_promedio_mid,
                   "conteo_empresas":lt.size(empresa_mid_lista),
                   "empresa_mas_solicitada":(lt.getElement(empresa_mid_lista, 1)["empresa"],lt.getElement(empresa_mid_lista, 1)["cantidad"]),
                   "empresa_menos_solicitada":(lt.getElement(empresa_mid_lista, lt.size(empresa_mid_lista))["empresa"],lt.getElement(empresa_mid_lista, lt.size(empresa_mid_lista))["cantidad"]),
                   "conteo empresas skills":contador empresas skills mid
                   },
            "senior":{"conteo_habilidades":conteo_habilidades_senior,
                      "habilidad_mas_solicitada":(lt.getElement(top_habilidades_senior_lista, 1)["habilidad"],lt.getElement(top_habilidades_senior_lista, 1)["cantidad"]),
                      "habilidad menos solicitada":(lt.getElement(top habilidades senior lista, lt.size(top habilidades senior lista))["habilidad"],lt.getElement(top habilidades senior lista, lt.size(top habilidades senior lista))["cantidad"]),
                      "nivel_promedio":nivel_promedio_senior,
                      "conteo_empresas":lt.size(empresa_senior_lista),
                      "empresa_mas_solicitada":(lt.getElement(empresa_senior_lista, 1)["empresa"],lt.getElement(empresa_senior_lista, 1)["cantidad"]),
                      "empresa_menos_solicitada":(lt.getElement(empresa_senior_lista, lt.size(empresa_senior_lista))["empresa"],lt.getElement(empresa_senior_lista, lt.size(empresa_senior_lista))["cantidad"]),
                      "conteo_empresas_skills":contador_empresas_skills_senior
                      },
        "otros":{"total_ofertas":lt.size(lista_junior)+lt.size(lista_mid)+lt.size(lista_senior),
                 "num_ciudades":lt.size(lista_ciudades),
                 "pais mas ofertas":((lt.getElement(lista paises, 1)["country code"]),lt.getElement(lista_paises, 1)["cantidad"]),
                 "ciudad_mas_ofertas":((lt.getElement(lista_ciudades, 1)["city"]),lt.getElement(lista_ciudades, 1)["cantidad"])}
}

return retorno
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Número de países, fecha inicial y fecha final
Salidas	Diccionario de valores con listas de DISClib
Implementado (Sí/No)	Si, Diego Alejandro Munévar, Maria Alejandra Lasso y Maria Alejandra Sanabria.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Crear un array list de DISClib	O(1)
Paso 2: Iterar sobre toda la lista de trabajos	O(N)
Paso 3: Añadir un elemento a la última posición	O(1)
Paso 4: Verificar si el país del trabajo está en el diccionario de países	O(1)
Paso 5: Incrementar la cantidad de trabajos en ese país	O(1)
Paso 6: Verificar si la ciudad del trabajo está en el diccionario de ciudades	O(1)
Paso 7: Incrementar la cantidad de trabajos en esa	O(1)

Paso 8: Añadir la ciudad al diccionario de ciudades	O(1)
Paso 9: Añadir el país al diccionario de países	O(1)
Paso 10: Crear una lista vacía para almacenar los países en forma de diccionario	O(1)
Paso 11: Iterar sobre el diccionario de países	O(N)
Paso 12: Añadir cada país a la lista de países	O(1)
Paso 13: Ordenar la lista de países	O(N*log(N))
Paso 14: Crear una lista vacía para almacenar las ciudades en forma de diccionario	O(1)
Paso 15: Iterar sobre la lista de países	O(N)
Paso 16: Iterar sobre el diccionario de ciudades	O(N)
Paso 17: Añadir cada ciudad a la lista de ciudades	O(1)
Paso 18: Ordenar la lista de ciudades	O(N*log(N))
Paso 19: Crear listas vacías para almacenar los trabajos de acuerdo al nivel de experiencia	O(1)
Paso 20: Iterar sobre la lista de trabajos	O(N)
Paso 21: Añadir trabajos a las listas de trabajos junior, mid y senior	O(1)
Paso 22: Crear un diccionario con el id como llave y otro diccionario con el nombre de la habilidad y el nivel como valor	O(1)
Paso 23: Iterar sobre la lista de habilidades	O(P)
Paso 24: Añadir habilidades al diccionario de habilidades	O(1)
Paso 25: Crear un diccionario con el id como llave y otro diccionario con la ciudad y la cantidad de trabajos como valor	O(1)
Paso 26: Iterar sobre la lista de multilocation	O(N)
Paso 27: Añadir ciudades al diccionario de multilocation	O(1)
Paso 28: Crear variables para almacenar los datos de los trabajos junior, mid y senior	O(1)
Paso 29: Iterar sobre las listas de trabajos junior, mid y senior	O(N)
Paso 30: Calcular el conteo de habilidades, las habilidades más y menos solicitadas, el nivel promedio, las empresas más y menos solicitadas	O(1)
Paso 31: Crear listas para almacenar los datos de los diccionarios	O(1)
Paso 32: Iterar sobre los diccionarios	O(N)
Paso 33: Añadir elementos a las listas	O(1)
Paso 34: Ordenar las listas	O(N*log(N))
Paso 35: Crear un diccionario para almacenar todos los datos	O(1)
TOTAL	***O(N*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).

Procesadores	**Intel Core i5-1135G7**
Memoria RAM	16 GB
Sistema Operativo	Windows 11

Datos ingresados:

Ingrese el numero de paises que desea ver: 10

Ingrese la fecha de inicio en formato YYYY-MM-DD: 2022-01-01

Ingrese la fecha de fin en formato YYYY-MM-DD: 2024-01-01

Entrada	**Tiempo (ms)**
10 pct	1111.44
20 pct	1808.85
30 pct	2987.59
50 pct	4174.02
60 pct	5393.35
70 pct	5621.39
80 pct	5672.97
large	5811.22

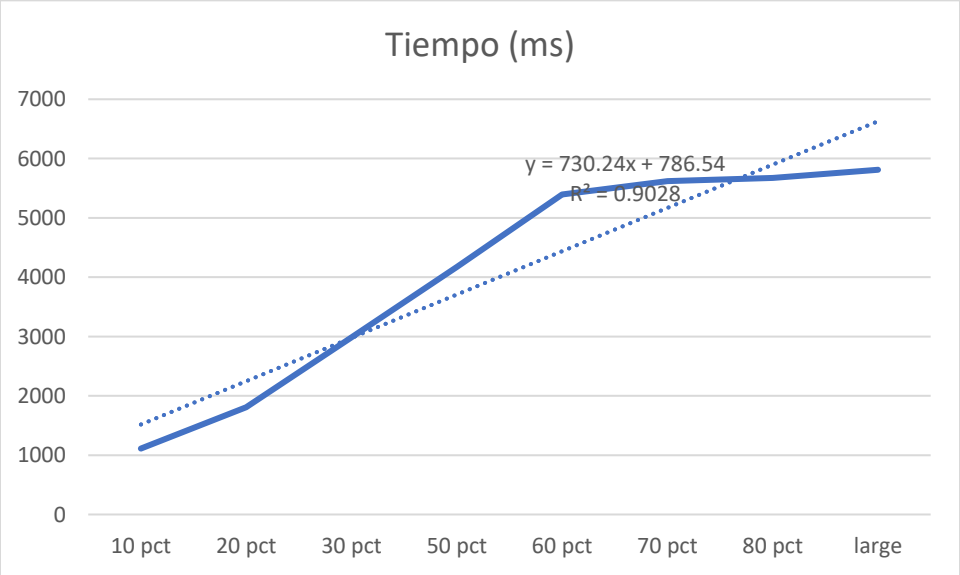
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
10 pct	Lista de DISClib	1111.44
20 pct	Lista de DISClib	1808.85
30 pct	Lista de DISClib	2987.59
50 pct	Lista de DISClib	4174.02
60 pct	Lista de DISClib	5393.35
70 pct	Lista de DISClib	5621.39
80 pct	Lista de DISClib	5672.97
large	Lista de DISClib	5811.22

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

La implementación, de acuerdo con el análisis de complejidad hecho, debería ser de una complejidad linearitmica, es decir O(N*log(N)), donde es un poco más compleja que una escala lineal, pero cumple con su función de manera correcta, ahora, de acuerdo con la gráfica, tiene una correlación de 0.9, lo cual indica que es un método bastante eficiente para el problema en cuestión. Esta correlación cercana a 1 sugiere que existe una fuerte relación entre el tamaño de la entrada y el tiempo de ejecución del algoritmo, lo cual es consistente con una complejidad de O(N*log(N)).

En términos prácticos, esto significa que a medida que aumenta un poco más que la lineal, pero al mismo tiempo conforme va incrementando el tamaño de los datos, no se presenta un incremento significativo frente a una escala lineal.