ANÁLISIS DEL RETO

Estudiante 1 Juan Goyeneche, 202320863, j.goyeneches@uniandes.edu.co
Estudiante 2 Mariana González, 202311308, m.gonzalezp23@uniandes.edu.co
Estudiante 3 Sebastián Ruiz

Requerimiento <<1>>

Descripción

Este requerimiento se encarga de retornar n ofertas de trabajo ofrecidas en un país y según el nivel de experticia (valores ingresados por parámetro) Primero se accede al mapa "countries". Si hay valores en este mapa se accede a toda la información de las ofertas para hacer una iteración y filtrar la lista con los valores que cumplan con la experiencia deseada; después estos se agregan a una lista. Finalmente se hace un sort de la lista final

Entrada	Numero_ofertas, cod_pais, nivel de xp
Salidas	Total de ofertas ofrecidas por pais y nivel de experiencia, información de cada una de las ofertas
Implementado (Sí/No)	Si. Implementado por Juan Goyeneche, Mariana González

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (Obtener los elementos con esa llave del mapa)	O(N)
Paso 2 (iteración filtrar lista)	O(n)
Paso 3 (addLast)	O(n)
Paso 4 (sorted)	O (n log n)
Paso 5 (n_ofertas)	O(n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el país (US), el nivel de xp (senior) y el número de ofertas que se quieren listar 3

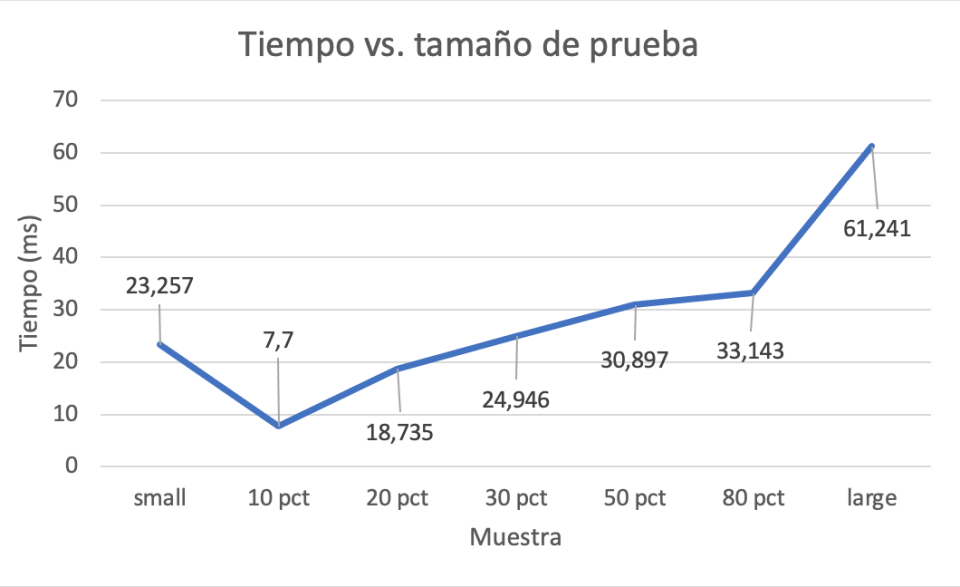
Procesadores	AMD Ryzen 7 4800HS with Radeon Graphics
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Muestra	**Tiempo (ms)**
small	23.257
10 pct	7.700

20 pct	18.735
30 pct	24.946
50 pct	30.897
80 pct	33.143
large	61.241

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este tiene que obtener un elemento de mapa (CHAINING) que tenga la misma llave que el país ingresado, por ellotiene una complejidad de O(n). Esto debido a lo que se hace después de obtener los trabajos en ese país, es buscar de todos los elementos de esa lista que tenga el mismo nivel de xp, después agregarlos a una lista y finalmente, organizar esa lista y agregar a una sublista. Debido a esto, en el peor de los casos es necesario recorrertoda la lista, seguido a esto agregar a una nueva lista y finalmente, organizar esta lista, agregar a una sublista y

```
def req_1(data_structs, n_ofertas, cod_pais, xp):
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    lt1 = lt.newList("ARRAY_LIST")
    sortiao = lt.newList("ARRAY_LIST")
    country = mp.get(data_structs['countries'], cod_pais)
    if country:
        var1 = me.getValue(country)['jobs']
        cant_of_pais = mp.size(var1)
        for i in lt.iterator(var1):
            if xp.lower() == i["experience_level"].lower():
                lt.addLast(lt1, i)

    if lt.isEmpty(lt1):
        print("Ningun resultado encontrado")
        sys.exit(0)
    elif lt.size(lt1) >= 2:
        sortiao = merg.sort(lt1, sort_r3)
    elif lt.size(lt1) <= 1:
        sortiao = lt1


    if n_ofertas > lt.size(sortiao):
        n_ofertas = lt.size(sortiao)
    elif n_ofertas <= lt.size(sortiao):
        n_ofertas = n_ofertas


    final = lt.subList(sortiao, 0, n_ofertas)

    cant_xp = lt.size(lt1)

    return final, cant_xp, cant_of_pais
```

devolver el resultado.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n log n), manteniendo una relación con el aumento del tiempo y el aumento de los datos. Un factor clave a tomar en cuenta es el load factor el cual puede afectar el rendimiento de este.

Requerimiento <<2>>-

Descripción

Este requerimiento se encarga de retornar el número de ofertas que realiza una empresa en una ciudad (entradas por parámetro) y la información de estas ofertas. Primero, se hace un get() para acceder a los valores de la empresa que estan en el mapa “companies”. Si hay valores de esta empresa se hace un get() para obtener la información de los trabajos, después se filtran las ofertas con la ciudad ingresada por parámetro; las que cumplan se agregan a una lista y se ordena cronológicamente. Finalmente, se retornan las primeras ofertas deseadas.

Entrada	Numero_ofertas, nombre_empresa, ciudad
Salidas	Total de ofertas ofrecidas por empresa y ciudad, información de cada una de las ofertas
Implementado (Sí/No)	Si. Implementado por Juan Goyeneche y Mariana González

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (get)	O(n)
Paso 2 (get	O (n)
Paso 3 (iteración)	O(n)
Addlast (addLast)	O(n)
Hacer el sort	O(n log n) (promedio)
TOTAL	O(n log n)

Pruebas Realizadas

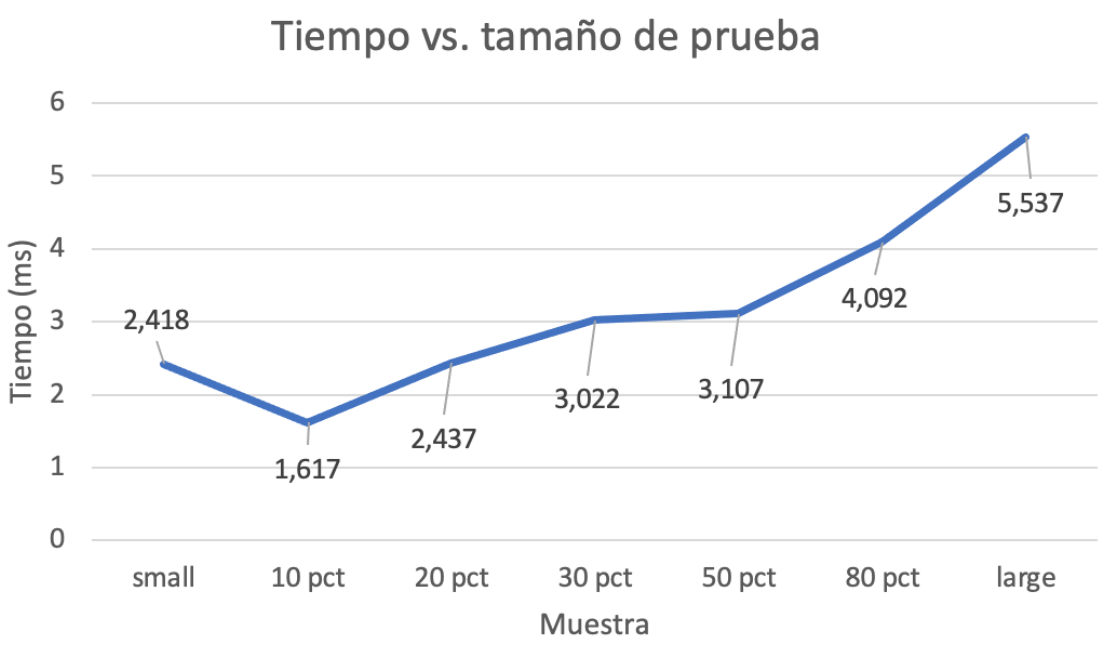
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron la ciudad (Bratislava), la empresa (cloudtalk) y el número de ofertas que se quieren listar 5

Procesadores	1,1 GHz Intel Core i3 de dos núcleos
Memoria RAM	8 GB
Sistema Operativo	MacOS sonoma

Muestra	**Tiempo (ms)**
small	2.418
10 pct	1.617
20 pct	2.437
30 pct	1.816
50 pct	3.107
80 pct	4.092
large	5.537

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Se desea la información que está en el mapa “companies”, por lo que se debe hacer un get(), el cual tiene una complejidad de O(n), para acceder a la información de este mapa. Ahora bien, se hace otro get() para obtener los valores de la empresa que se desea conocer. Después se hace una iteración (Complejidad O(n)) para obtener las ofertas de la ciudad deseada y se añaden a un ARRAY_LIST, el cual será organizado cronológicamente después. Por esto, en el peor caso hay que recorrer todo el mapa, seguido agregar a una nueva lista y organizar esta lista y devolver el resultado.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera proporcional (n), manteniendo una relación con el aumento del tiempo y el aumentode los datos; siendo 1,617ms el menor tiempo con 34470 datos y 5,537 el mayor tiempo con 203562 datos.

```python
def req_2(data_structs, num_ofertas, empresa, ciudad):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    filtro = lt.newList('ARRAY_LIST')
    company = mp.get(data_structs["companies"], empresa)

    if company:
        valores = me.getValue(company)["jobs"]
        for job in lt.iterator(valores):
            if ciudad.lower() == job["city"].lower():
                lt.addLast(filtro, job)

    tamanho = lt.size(filtro)
    if tamanho == 0:
        print("Ningun resultado encontrado")
        sys.exit(0)
    elif num_ofertas > lt.size(filtro):
        num_ofertas = lt.size(filtro)
    elif num_ofertas <= lt.size(filtro):
        num_ofertas = num_ofertas

    sublista = lt.subList(filtro, 0, num_ofertas)

    return sublista, tamanho
```

Requerimiento <<4>>

Descripción

Este requerimiento se encarga de retornar el número de ofertas que se realizan en un pais (dado por parámetro) y la información de estas ofertas. Primero se filtra el mapa countries tomando en cuenta el pais y se recorre la lista. Después las ofertas que cumplen con las fechas se agregan a una lista para ser ordenadas cronológicamente. Finalmente se retornan las ofertas deseadas

Entrada	pais, fecha mínima, fecha maxima
Salidas	Total, de ofertas ofrecidas, empresas con al menos una oferta, ciudad con mayor ofertas, ciudad con menor ofertas y las ofertas.
Implementado (Sí/No)	Si, Juan Goyeneche

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isEmpty)	O(1)
Obtener la lista del mapa	O(n)
(Conversión formato de fecha)	O(n)
(Iteración filtro de fecha)	O(n)
Obtener el tamaño (size)	O(1)
Addlast (addLast)	O(n)
Hacer el sort	O(n log n) (promedio)
TOTAL	***O(n log n)***

Pruebas Realizadas

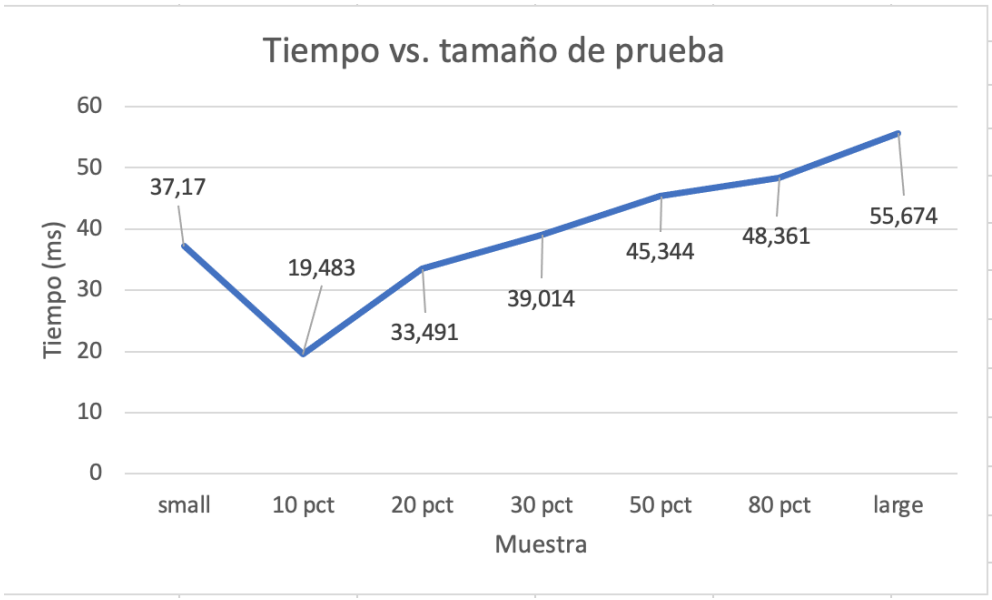
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada US, 2020-11-11, 2023-11-11.

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Muestra	**Tiempo (ms)**
small	37.170
10 pct	19.483
20 pct	33.491
30 pct	39.014
50 pct	45.344
80 pct	48.361
large	55.674

Gráficas

Las gráficas con la representación de las pruebas realizadas



Análisis

Este tiene que obtener un elemento de mapa (CHAINING) que tenga la misma llave que el país ingresado, por ello tiene una complejidad de O(n). Esto debido a lo que se hace después de obtener los trabajos en ese país, es buscar de todos los elementos de esa lista que tenga misma fecha, después agregarlos a una lista y finalmente, organizar esa lista. Debido a esto, en el peor de los casos es necesario recorrertoda la lista, seguido a esto agregar a una nueva lista y finalmente, organizar esta lista.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n log n), manteniendo una relación con el aumento del tiempo y el aumento de los datos. Un factor clave a tomar en cuenta es el load factor el cual puede afectar el rendimiento de este.

```python
def req_4(data_structs, cod_pais, fecha_inicial_consulta, fecha_final_consulta):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    fecha_inicial_dt = str(dt.strptime(fecha_inicial_consulta, "%Y-%m-%d"))
    fecha_final_dt = str(dt.strptime(fecha_final_consulta, "%Y-%m-%d"))
    chili = lt.newList("ARRAY_LIST")
    lt1 = lt.newList("ARRAY_LIST")
    country = mp.get(data_structs['countries'], cod_pais)
    #print(country)
    if country:
        var1 = me.getValue(country)['jobs']
        for i in lt.iterator(var1):
            fecha_diccionario = dt.strptime(i["published_at"], '%Y-%m-%dT%H:%M:%S.%fZ')
            fecha_diccionario_dt = fecha_diccionario.strftime("%Y-%m-%d")
            if fecha_inicial_dt <= fecha_diccionario_dt and fecha_diccionario_dt <= fecha_final_dt:
                lt.addLast(chili, i)

    if lt.size(chili) >= 2:
        sortiao = merg.sort(chili, sort_r3)
    elif lt.size(chili) <= 1:
        sortiao = chili

    if lt.isEmpty(chili):
        print("Ningun resultado encontrado")
        sys.exit(0)

    cant_of_pais = lt.size(chili)


    return sortiao, cant_of_pais
```

Requerimiento <<5>>

Descripción

Este requerimiento se encarga de encontrar las ofertas de trabajo realizadas en una ciudad durante cierto rango de tiempo. Primero cambia el formato de la fecha ingresada por el usuario para que pueda ser comparada con el formato de la fecha contenida en los diccionarios de cada oferta. Después, en el mapa de cities obtiene la lista con las ofertas en esta e itera sobre cada una de estas para comparar las fechas de publicación; si es acorde a los parámetros ingresados se compara la ciudad y se agrega la oferta a una nueva lista que contiene las ofertas funcionales. Finalmente se ordena cronológicamente la lista para ser retornada

Entrada	Ciudad, fecha inicial consulta, fecha final consulta
Salidas	-Total de ofertas publicadas en la ciudad en el rango de fechas dado -Total de empresas que publicaron al menos 1 oferta en la ciudad -Empresa con mayor número de ofertas y cuantas ofertas publicó - Empresa con menor número de ofertas y cuantas ofertas publicó -Listado de las ofertas ordenadas cronológicamente
Implementado (Sí/No)	Si. Implementado por Mariana González Puentes

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (obtener la lista por llave del mapa)	O(n)
Paso 2 (Conversión formato de fecha)	O(n)
Paso 3 (Iteración filtro de fecha)	O(n)
Paso 4 (addLast)	O(n)
Paso 5 (sorted)	O(n log n)
TOTAL	***O (n log n)***

Pruebas Realizadas

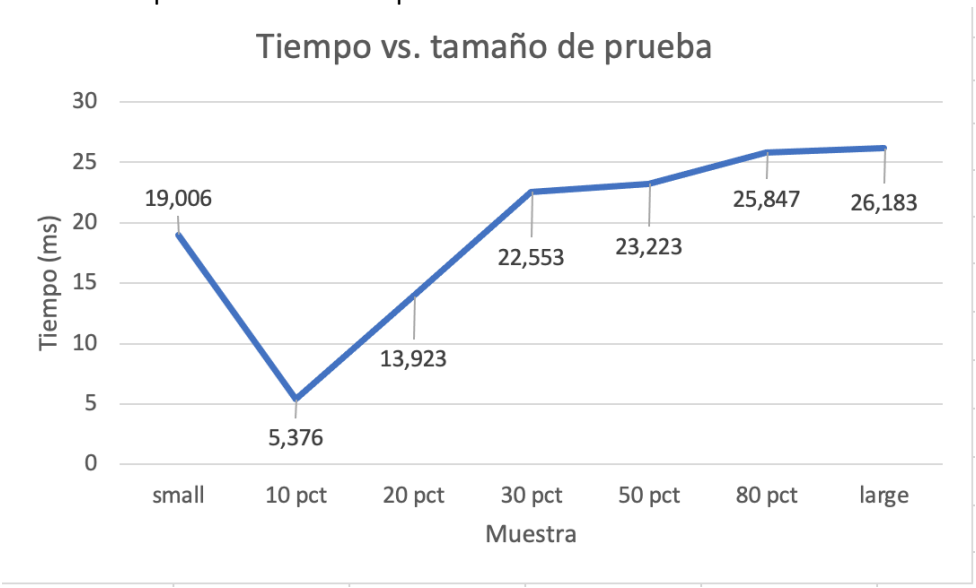
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el nombre de la ciudad (San Francisco), la fecha inicial de consulta (2020-11- 11) y la fecha final de consulta (2023-11-11)

1,1 GHz Intel Core i3 de dos núcleos	
8 GB	
MacOS sonoma	
Muestra	**Tiempo (ms)**
small	19.006
10 pct	5.375
20 pct	13.923

30 pct	22.553
50 pct	23.223
80 pct	25.847
large	26.183

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este tiene que obtener un elemento de mapa (CHAINING) que tenga la misma llave que la ciudad ingresada, por ello tiene una complejidad de O(n). Esto debido a lo que se hace después de obtener los trabajos en esa ciudad, es buscar de todos los elementos de esa lista que tenga misma fecha, después agregarlos a una lista y finalmente, organizar esa lista. Debido a esto, en el peor de los casos es necesario recorrer toda la lista, seguido a esto agregar a una nueva lista y finalmente, organizar esta lista.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n log n), manteniendo una relación con el aumento del tiempo y el aumento de los datos. Sin embargo, se presenta un mayor incremento lineal desde 10 pct hasta 30 pct, a partir de este se reduce drásticamente la diferencia de tiempo entre las muestras.

```python
def req_5(data_structs, ciudad, fecha_inicial_consulta, fecha_final_consulta):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    fecha_inicial_dt = str(dt.strptime(fecha_inicial_consulta, "%Y-%m-%d"))
    fecha_final_dt = str(dt.strptime(fecha_final_consulta, "%Y-%m-%d"))
    lttot = lt.newList("ARRAY_LIST")
    ltm = lt.newList("ARRAY_LIST")
    ltsr = lt.newList("ARRAY_LIST")
    chili = lt.newList("ARRAY_LIST")

    city = mp.get(data_structs['cities'], ciudad)
    #print(city)
    if city:
        var1 = me.getValue(city)['jobs']
        for i in lt.iterator(var1):
            fecha_diccionario = dt.strptime(i["published_at"], '%Y-%m-%dT%H:%M:%S.%fZ')
            fecha_diccionario_dt = fecha_diccionario.strftime("%Y-%m-%d")
            if fecha_inicial_dt <= fecha_diccionario_dt and fecha_diccionario_dt <= fecha_final_dt:
                lt.addLast(chili, i)

    tot_of = lt.size(chili)

    if lt.size(chili) >= 2:
        sortiao = merg.sort(chili, sort_r3)
    elif lt.size(chili) <= 1:
        sortiao = chili

    if lt.isEmpty(chili):
        print("Ningun resultado encontrado")
        sys.exit(0)

    return sortiao, tot_of
```

Requerimiento <<6>>

Este requerimiento se encarga de clasificar las N ciudades con mayor cantidad de ofertas de trabajo dado un año y nivel de experticia de la oferta. Lo primero que hace es encontrar los elementos que cumplen con las condiciones (esto lo hace dependiendo de si se provee un nivel de xp o no, ya que, si lo hace, se saca la info de un mapa llamado xps donde la llave es el nombre del nivel de xp. Por otro lado, si no se provee un nivel de xp, se sacarán todas las ofertas dado un año del mapa years), después busca los elementos en el otro mapa (infos) que tienen el mismo id de los que cumplían con las condiciones y seguido a esto, en la otra función, crea nuevos diccionarios con los datos deseados. De no existir, retorna Ningún resultado encontrado y termina el programa.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructuras de datos del modelo, numero de ciudades, nivel de experiencia y año.
Salidas	Una nueva tad lista organizada y cada elemento de la tad lista contiene: Nombre de la ciudad. o País de la ciudad. o El total de ofertas hechas en la ciudad. o Promedio de salario ofertado en la ciudad o Número de empresas que publicaron por lo menos una oferta en la ciudad o Nombre de la empresa con mayor número de ofertas y su conteo o La información de la mejor oferta por salario en la ciudad (considerando el tope más alto dado en una oferta). Se consideran solo ofertas que tengan información de salario. o La información de la peor oferta por salario en la ciudad (considerando el tope más bajo dado en una oferta). Se consideran solo ofertas que tengan información de salario.
Implementado (Sí/No)	Si. Implementado por Juan Goyeneche y Mariana González

Análisis de complejidad

Pasos	**Complejidad**
Obtener el elemento del mapa (ya sea por año o por xp)	O(n)
Añadir a la lista	O(n)
Obtener el id del mapa infos	O(n)
(Conversión formato de fecha)	O(n)
Addlast (addLast)	O(n)
Iterar por la nueva lista	O(n)
Crear los diccionarios	O(n)
addLast	O(n)
Hacer el sort	O(n log n) (promedio)
TOTAL	***O(n log n)***

Pruebas Realizadas

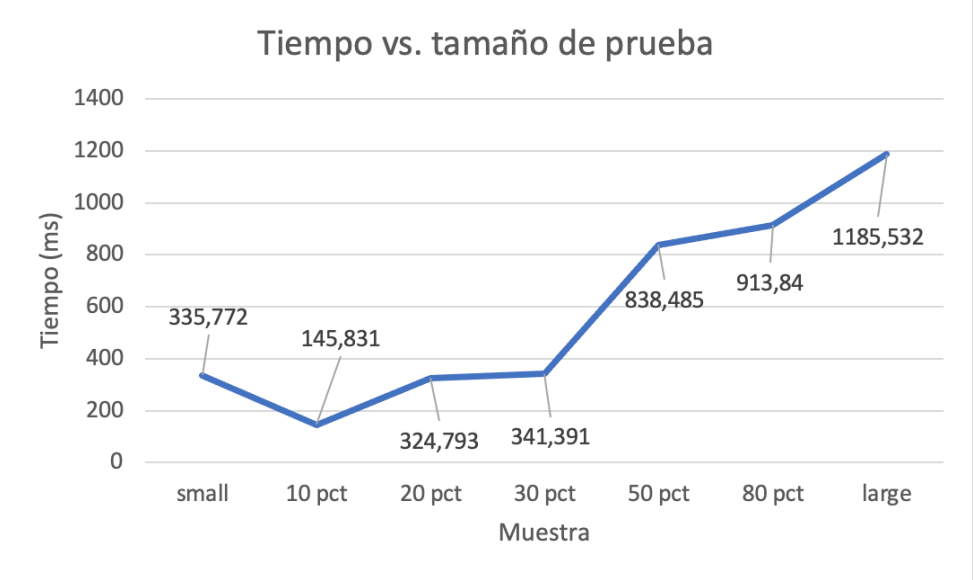
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el 5, junior, 2023.

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Muestra	**Tiempo (ms)**
small	335.772
10 pct	145.831
20 pct	324.793
30 pct	341.391
50 pct	838.485
80 pct	913.840
large	1185.532

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este tiene que obtener un elemento de mapa (CHAINING) que tenga la misma llave que la xp o fecha ingresada, por ello tiene una complejidad de O(n). Esto debido a lo que se hace después de obtener los trabajos por ese año o xp, es buscar de todos los elementos de esa lista que tenga mismo año (en caso de que la xp no sea indiferente) , después agregarlos a una lista y obtener el id de los elementos en esa lista para así obtener las ofertas del otro mapa (infos). Seguido a esto, se crea un nuevo diccionario con cada ciudad, ofertas económicas, etc y cada uno de estos se agrega a una nueva lista. Finalmente, organizar esa lista. Debido a esto, en el peor de los casos es necesario recorrer toda la lista, seguido a esto agregar a una nueva lista y finalmente, organizar esta lista.

```python
def req62(lst):
    cr7 = lt.newList("ARRAY_LIST")

    for i in lt.iterator(lst):
        if lt.isPresent(cr7, i["company_name"]) == 0:
            lt.addLast(cr7, i["company_name"])
        else:
            batman = 0

    return cr7
```

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n log n), manteniendo una relación con el aumento del tiempo y el aumento de los datos. Un factor clave a tomar en cuenta es el load factor el cual puede afectar el rendimiento de este.

```python
def req64(lst, info_lt):
    fin = {}
    for i in lt.iterator(lst):
        fin[i["id"]] = i

    info_ciu = defaultdict(dict)

    for x in lt.iterator(info_lt):
        of = fin[x["id"]]
        ciudad = of["city"]
        pais = of["country_code"]
        empresa = of["company_name"]
        if x["salary_from"] == "" or x["salary_to"] == "":
            sal_min = 0
            sal_max = 0
        else:
            sal_min = int(x["salary_from"])
            sal_max = int(x["salary_to"])

        if "total_ofertas" not in info_ciu[ciudad]:
            info_ciu[ciudad]["total_ofertas"] = 0
            info_ciu[ciudad]["salario"] = []
            info_ciu[ciudad]["cant_empresas"] = defaultdict(int)
            info_ciu[ciudad]["pais"] = pais
        elif info_ciu[ciudad]["pais"] != pais:
            info_ciu[ciudad]["pais"] = pais

        info_ciu[ciudad]["total_ofertas"] += 1
        info_ciu[ciudad]["salario"].extend([sal_min, sal_max])
        info_ciu[ciudad]["cant_empresas"][empresa] += 1

    for q, k in info_ciu.items():
        salarios = k["salario"]
        salario_promedio = sum(salarios) / len(salarios)
        k['salario_promedio'] = salario_promedio

    for q, k in info_ciu.items():
        empresa_mas_ofertas = max(k["cant_empresas"], key=k["cant_empresas"].get)
        cantidad_ofertas_empresa_mas = k["cant_empresas"][empresa_mas_ofertas]
        k['empresa_mas_ofertas'] = empresa_mas_ofertas
        k['cantidad_ofertas_empresa_mas'] = cantidad_ofertas_empresa_mas

    for q, k in info_ciu.items():
        salarios = k["salario"]
        mejor_oferta = max(salarios)
        peor_oferta = min(salarios)
        k['mejor_oferta'] = mejor_oferta
        k['peor_oferta'] = peor_oferta

    finalissima = lt.newList("ARRAY_LIST")
    for ciudad, info in info_ciu.items():
        lt.addLast(finalissima, {'ciudad': ciudad,
                                 'pais': info['pais'],
                                 'total_ofertas': info['total_ofertas'],
                                 'salario_promedio': info['salario_promedio'],
                                 "cant_empresas": len(info["cant_empresas"]),
                                 'empresa_mas_ofertas': info['empresa_mas_ofertas'],
                                 'cantidad_ofertas_empresa_mas': info['cantidad_ofertas_empresa_mas'],
                                 'mejor_oferta': info['mejor_oferta'],
                                 'peor_oferta': info['peor_oferta']})

    return finalissima
```