Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

Lucas Valbuena León Cod 202311538
Juan José Cortés Villamil Cod 202325148
Alisson Moreno Cod 202120330

Requerimiento <<Carga de datos>>

En la carga de datos lo que se hizo fue crear un mapa para todos los csv, employments_types, jobs, multilocations y skills. Pero además se creó un mapa para “specifc_jobs” en el que se guardan tuplas llaves, valor donde las llaves son compuestas del tipo:

```
keyS =
f"{data['country_code']};{data['city']};{data['company_name']};{data['expe
rience_level']};{data['published_at'][:10]}"

        existkeyS = mp.contains(data_structs["specific_jobs"],keyS)
```

De este modo en los requerimientos eventualmente se itera por el keyset y solo las llaves compuestas que contengan la información que nos interesa van a tener en el string de su llave la información que nos compete.

```
def load_data(control):
    """
    Carga los datos del reto
    """
    # TODO: Realizar la carga de datos
    catalog = control['model']
    if prueba == "Rapidez":
        start_time = get_time()
    else:
        tracemalloc.start()
        start_memory = get_memory()
```

```
    loadEmployments_types(catalog)
    loadJobs(catalog)
    loadMultilocations(catalog)
    loadSkills(catalog)



    sorted_jobs = model.sort_th(control)


    if prueba == "Rapidez":
        end_time = get_time()
        return control, delta_time(start_time, end_time), prueba,
sorted_jobs
    else:
        stop_memory = get_memory()
        tracemalloc.stop()
        A_memory = delta_memory(stop_memory, start_memory)
        return control, A_memory, prueba, sorted_jobs
```

No copiamos todo el código por su extensión. Pero en general se itera por lo que lee el csv.reader() y se va cargando en el mapa correspondiente. Para skills, employments_types y multilocations se guardo solo por 'id' en vista de que si no, a la hora de consultar por ejemplo un skill DESDE una oferta específica sería muy difícil.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Iterar por las líneas del csv	O(N)

`existkeyS = mp.contains(data_structs["specific_jobs"],keyS)`	O(N/M) en el peor caso
`mp.put(data_structs["specific_jobs"], keyS , lt.newList("ARRAY_LIST"))`	O(1)
`entry = mp.get(data_structs["specific_jobs"], keyS)`	O(N/M)
`lt.addLast(me.getValue(entry), data)`	O(1)
TOTAL	***Los pasos anteriores se repiten por cada csv, solo que specific_jobs tiene un tratamiento ligeramente diferente o más bien aislado por lo que tiene una llave compuesta pero se hacen las mismas operaciones pues entonces 4 veces.*** ***En el peor de los casos es O(N) y en realidad siempre es O(N) a menos de que se restrinja una parte del csv.***

	Máquina 1	Máquina 2	Máquina 3
Procesadores	AMD Ryzen 7 5700U with Radeon Graphics	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
Memoria RAM (GB)	16,0 GB	16,0 GB	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 11 Home Single Language	Windows 11 Pro	Windows 10 Home Single Language 22H2

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Máquina 1

Resultados

Carga de Catálogo PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	783438.284	49189.77

0.5	783438.567	49699.28
0.7	783439.954	46762.70
0.9	783435.745	46870.28

Tabla 2. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING en la Máquina 1.

Carga de Catálogo CHAINING

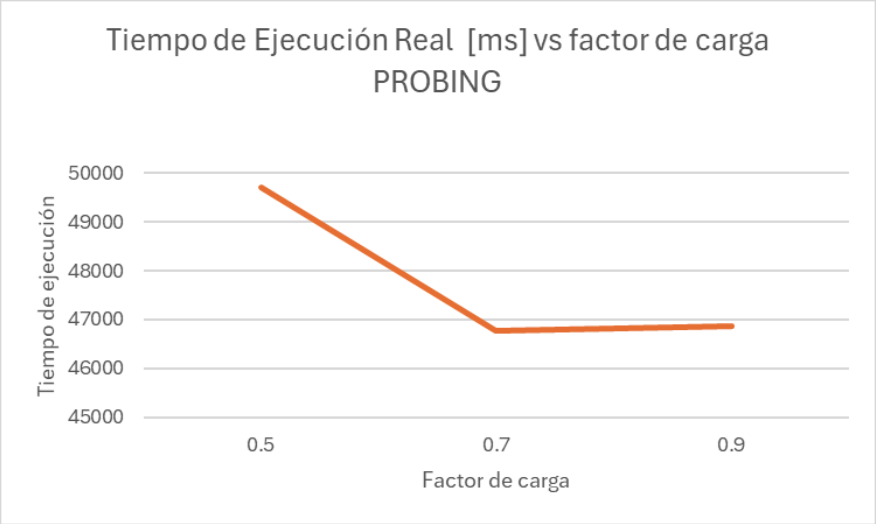
Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	1095472.841	45458.44
4.00	1095472.969	47536.40
6.00	1095472.544	48572.20
8.00	1095472.239	48733.02

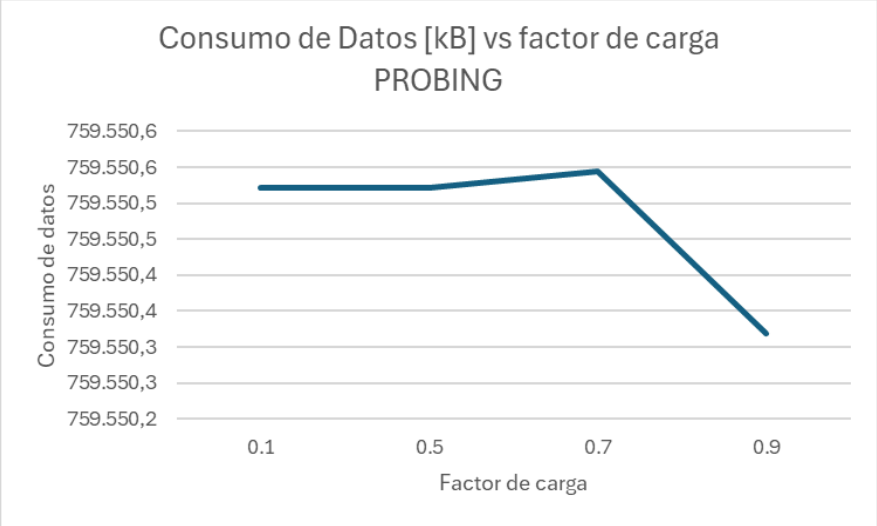
Tabla 3. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING en la Máquina 1.

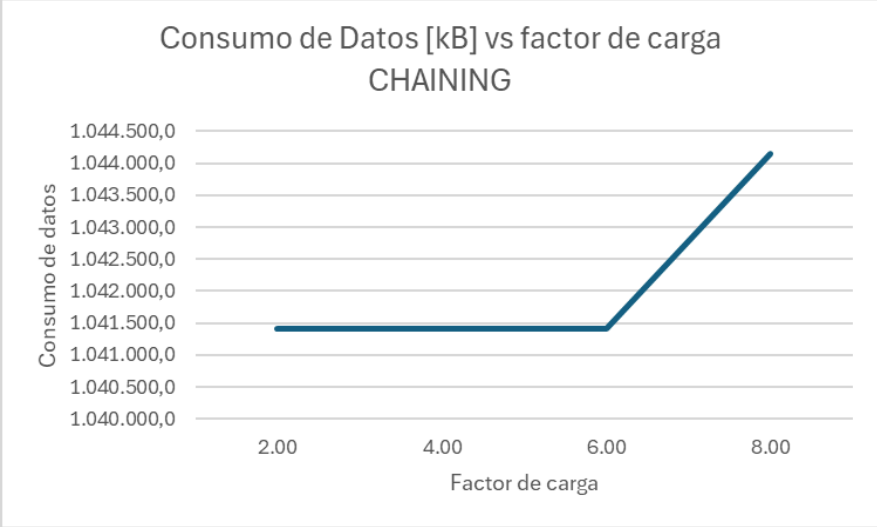
Gráficas

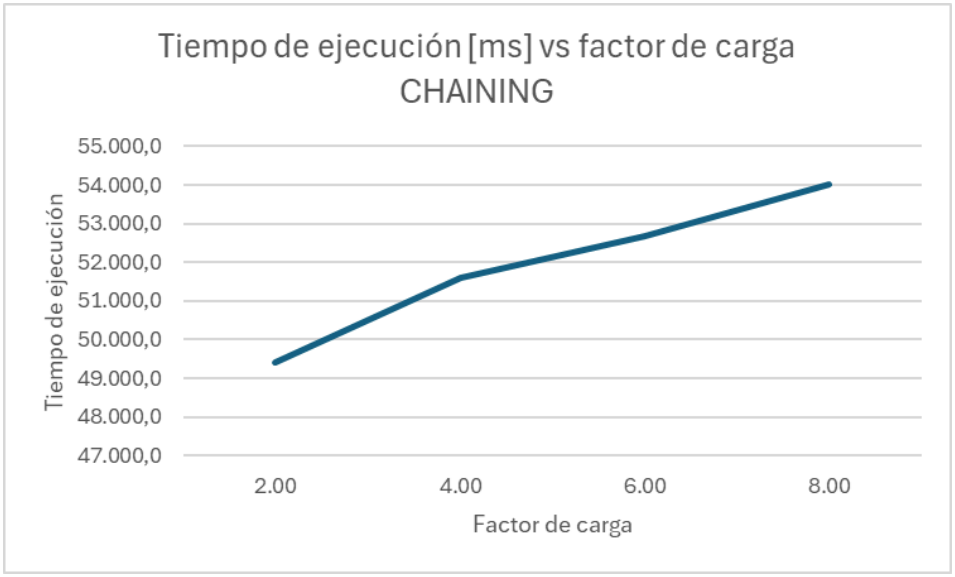
La gráfica generada por los resultados de las pruebas de rendimiento en la **Máquina 1.**

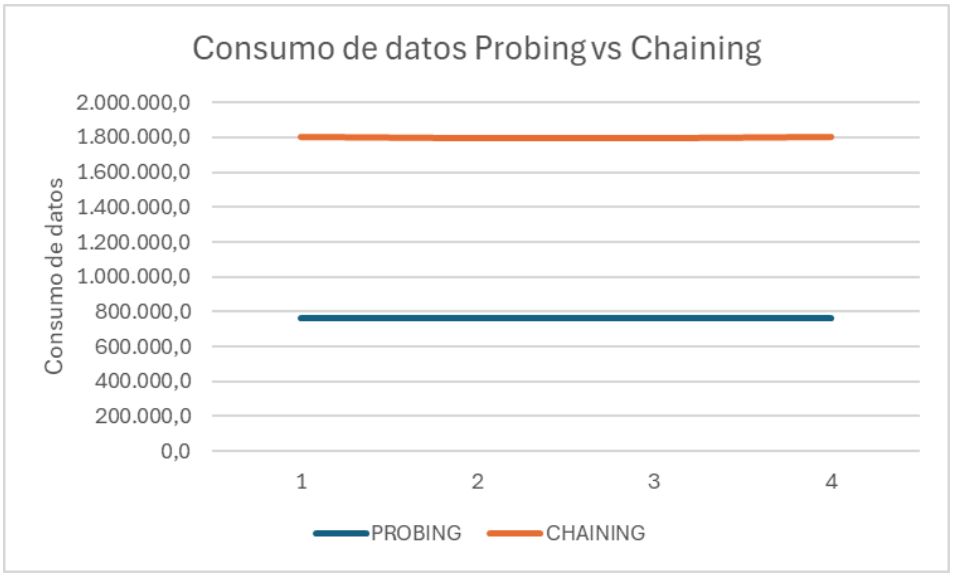
- Comparación de memoria y tiempo de ejecución para PROBING y CHAINING








Tiempo de ejecución [ms] vs factor de carga
CHAINING
55.000,0
54.000,0
53.000,0
52.000,0
51.000,0
50.000,0
49.000,0
48.000,0
47.000,0
Tiempo de ejecución
2.00
4.00
6.00
8.00
Factor de carga


Consumo de datos Probing vs Chaining
2.000.000,0
1.800.000,0
1.600.000,0
1.400.000,0
1.200.000,0
1.000.000,0
800.000,0
600.000,0
400.000,0
200.000,0
0,0
Consumo de datos
1
2
3
4
PROBING
CHAINING

Máquina 2

Resultados

Carga de Catálogo PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	759550.522	57332.323
0.5	759550.522	51982.809
0.7	759550.545	51863.166
0.9	759550.319	52443.777

Tabla 4. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING en la Máquina 2.

Carga de Catálogo CHAINING

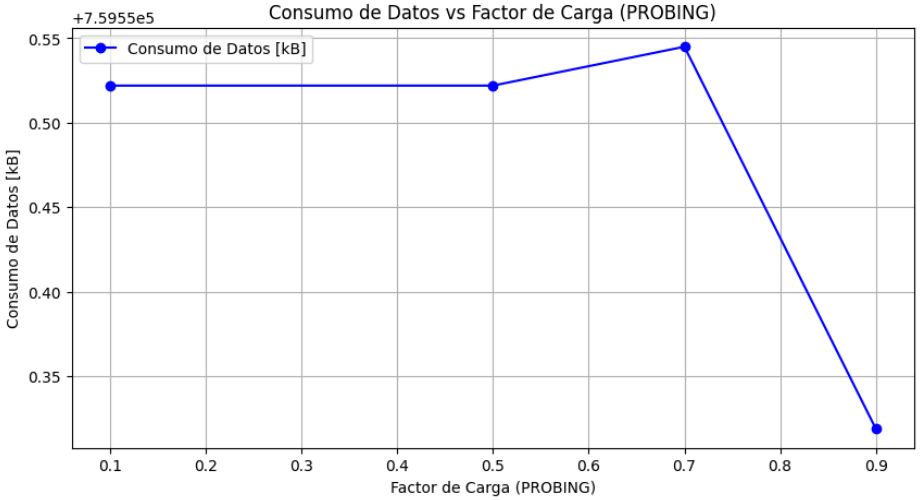
Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	1041406.358	49407.655
4.00	1041406.296	51589.833
6.00	1041405.983	52679.516
8.00	1044139.804	54009.633

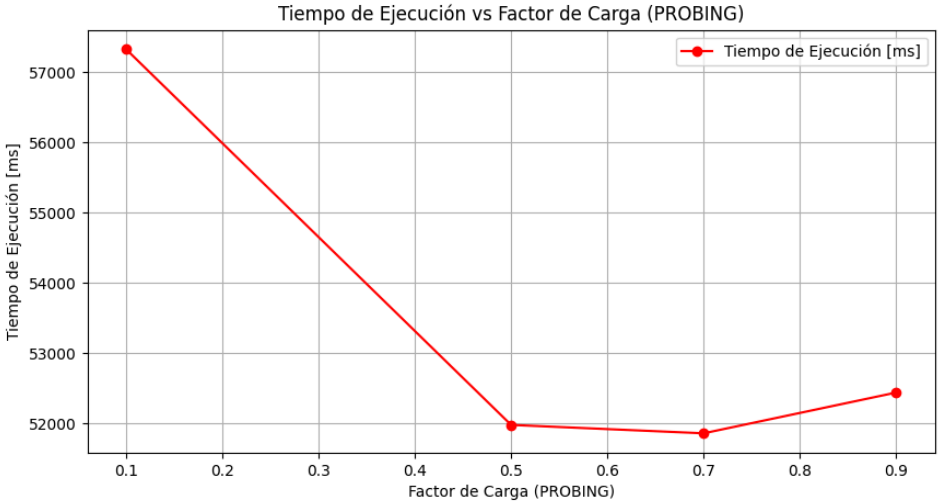
Tabla 5. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING en la Máquina 2.

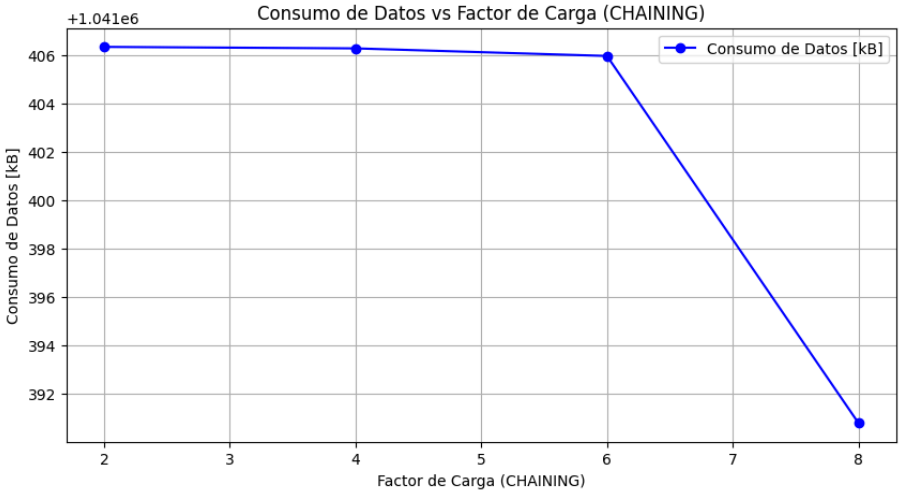
Gráficas

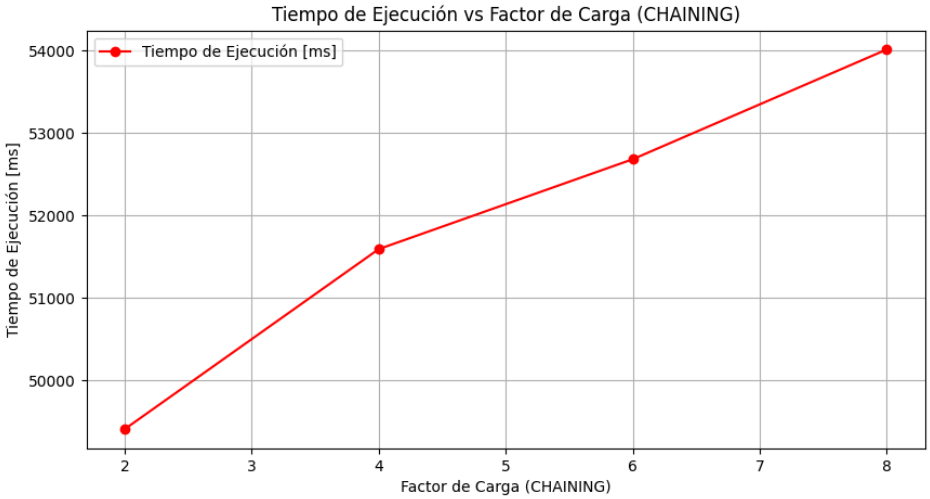
La gráfica generada por los resultados de las pruebas de rendimiento en la **Máquina 2.**

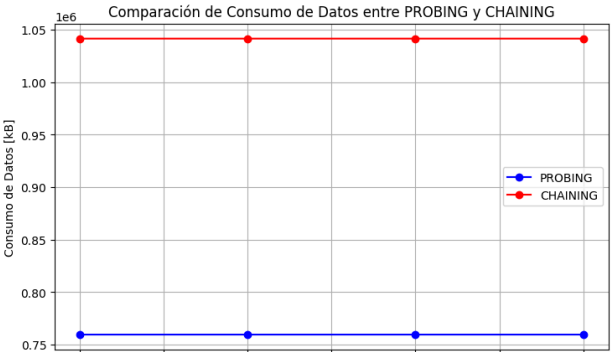
- Comparación de memoria y tiempo de ejecución para PROBING y CHAINING

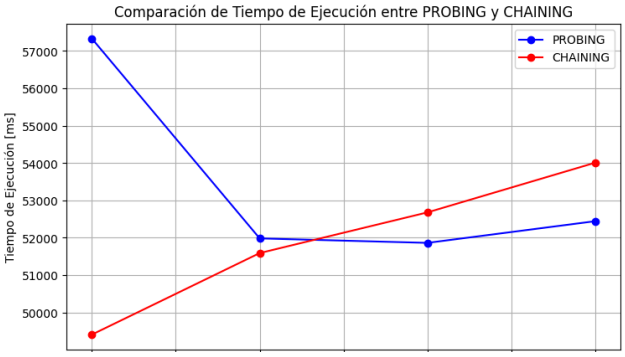

Consumo de Datos vs Factor de Carga (PROBING)
+7.5955e5
Consumo de Datos [kB]
0.55
0.50
0.45
0.40
0.35
Consumo de Datos [kB]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Factor de Carga (PROBING)


Tiempo de Ejecución vs Factor de Carga (PROBING)
Tiempo de Ejecución [ms]
57000
56000
55000
54000
53000
52000
Tiempo de Ejecución [ms]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Factor de Carga (PROBING)


Consumo de Datos vs Factor de Carga (CHAINING)
+1.041e6
Consumo de Datos [kB]
406
404
402
400
398
396
394
392
Consumo de Datos [kB]
2
3
4
5
6
7
8
Factor de Carga (CHAINING)







Maquina 3

Resultados

Carga de Catálogo PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @LP [ms]
0.1	Memory error	39469.8279002227
0.5	1054709.435546875	35874.756600022316
0.7	1054709.5546875	36836.89010000229
0.9	1054709.4326171875	40395.90680000186

Tabla 6. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING en la Máquina 3.

Carga de Catálogo CHAINING

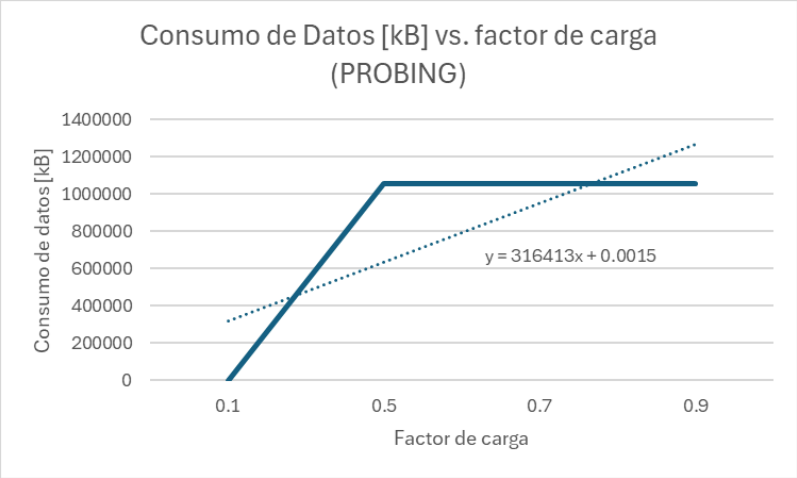
Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución Real @SC [ms]
2.00	Memory error	33915.21440002322
4.00	1418611.68554875	34863.69390001893
6.00	1418611.796875	35504.44620001316
8.00	1418611.8505859375	37257.8819998503

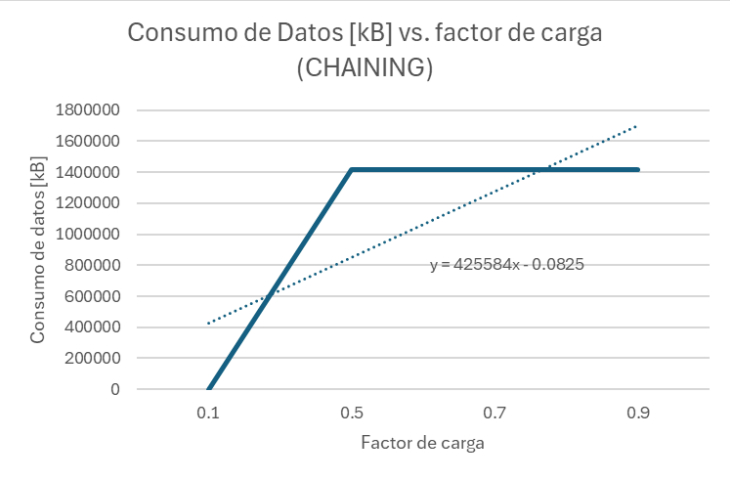
Tabla 7. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING en la Máquina 3.

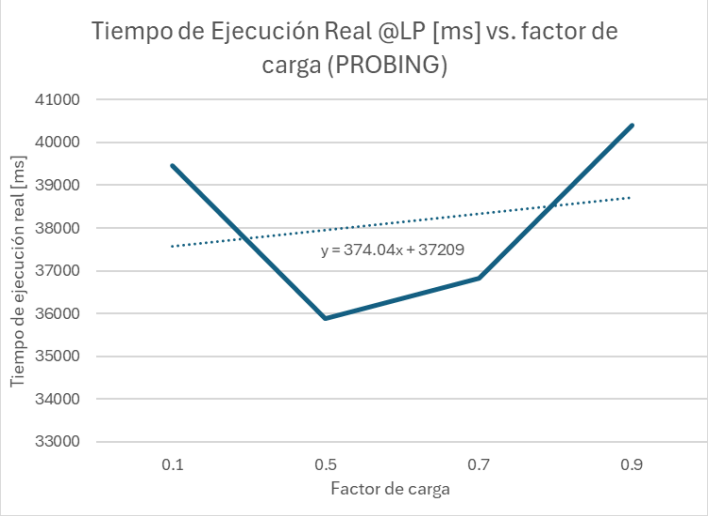
Gráficas

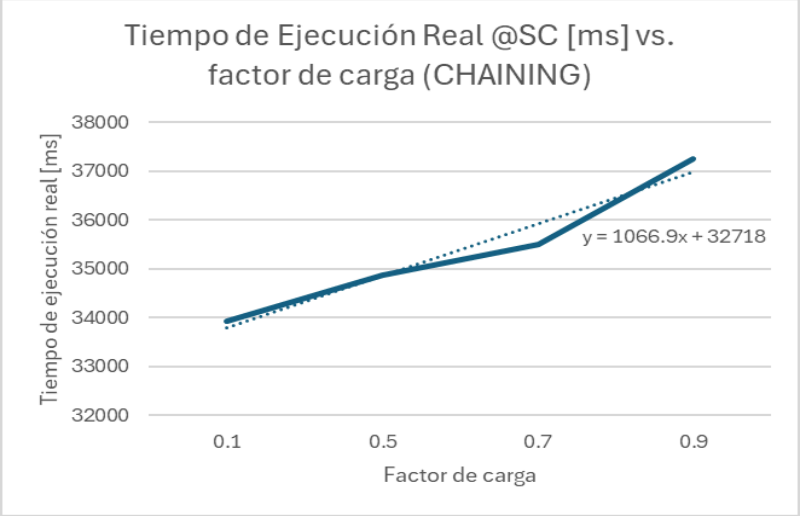
La gráfica generada por los resultados de las pruebas de rendimiento en la **Máquina 3.**

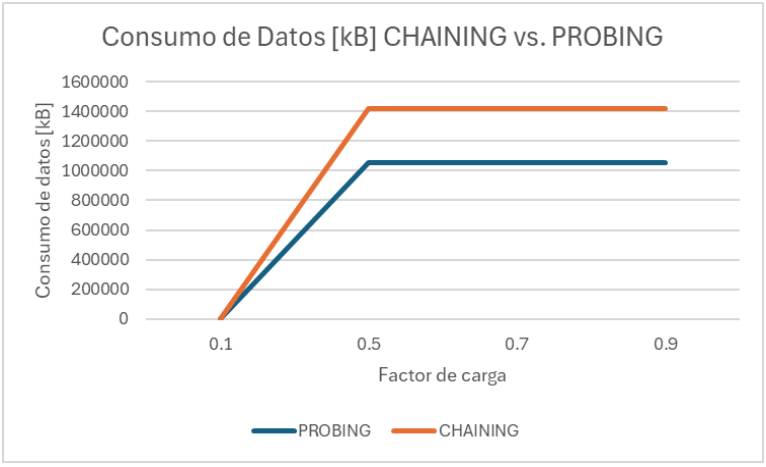
- Comparación de memoria y tiempo de ejecución para PROBING y CHAINING

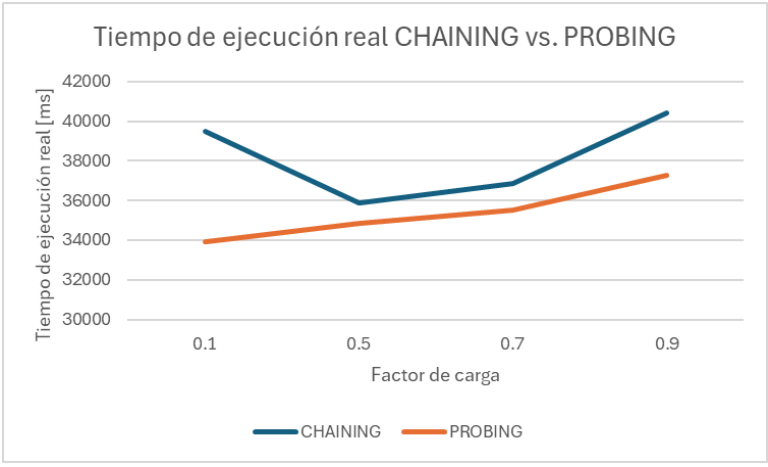












Análisis

Al modificar el esquema de colisiones es decir de PROBING a CHAINING nos damos cuenta que el tiempo de ejecución en el caso de CHAINING es más estable en su tendencia y en general tiene valores más pequeños que los de PROBING. Esto probablemente se da a que en CHAINING se da menos agrupamiento y si hay colisiones se guardan en un SINGLE_LINKED dentro del índice, mientras que en PROBING va a tener que recorrer en el peor de los casos N posiciones para encontrar donde meter un elemento que hizo colisión.

Al modificar el esquema de colisiones, el consumo de memoria es más significativo en CHAINING que en PROBING. Esto seguramente se debe a que CHAINING tiene una lista de tipo SINGLE_LINKED en cada índice del mapa, de modo que debe consumir mucha más memoria al tenerlos aunque estén vacíos.

Requerimiento <<1>>

```
def req_1(data_structs,n,pais,experticia):
    """
    Función que soluciona el requerimiento 1


    """
    catalogo= mp.keySet(data_structs["specific_jobs"])
    lista= lt.newList("ARRAY_LIST")
    conteo_pais= 0




    for key in lt.iterator(catalogo):
        keysplit= key.split(";")
        if experticia == keysplit[3] and pais == keysplit[0]:
            pareja=mp.get(data_structs["specific_jobs"],key)
```

```
            valor=me.getValue(pareja)
            for element in lt.iterator(valor):
                lt.addLast(lista, element)
        if pais==keysplit[0]:
            pareja=mp.get(data_structs["specific_jobs"],key)
            valor=me.getValue(pareja)
            for element in lt.iterator(valor):
                conteo_pais +=1




    lista_final=merg.sort(lista,sort_recientes_req1)
    if lt.size(lista_final) == 0:


        sublista = None
    if lt.size(lista_final) < n:
        sublista = lista_final
    else:
        sublista = lt.subList(lista_final,1,n)
    return sublista,lt.size(lista_final),conteo_pais
```

Descripción

En este requerimiento se filtran las ofertas de trabajo por país y nivel de experiencia, de modo que retorne la cantidad de ofertas que cumplan con dichas condiciones deseadas por el usuario. En primer lugar se obtiene la lista de llaves del mapa “specific_jobs” y se le asigna a la variable “catalogo”, además se genera un arreglo de nombre “lista” donde serán almacenadas todas las ofertas de trabajo que cumplan con los filtros y una variable “conteo_pais” para obtener el total de ofertas que fueron publicadas en un pais. Una vez hecho esto, se Itera con lt.iterator sobre cada llave compuesta del mapa, para luego filtrar de acuerdo a un país y nivel de experiencia especifico. Por último, si cumple con estas

condiciones, obtenemos la pareja llave valor correspondiente e iteramos sobre el valor teniendo en cuenta que es una lista para obtener todas las ofertas de trabajo y agregarlas a la lista, si solo coincide con el país se le suma 1 al contador del país.

Entrada	El catálogo en el apartado de “specific_Jobs” (en model), el número de ofertas a imprimir, el código del país y el nivel de experticia.
Salidas	Las ofertas filtradas por país y nivel de experticia.
Implementado (Sí/No)	Si se implementó y lo realizó Alisson Moreno

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (crear un nuevo arreglo),crear el contador y obtener la lista de llaves del mapa “specific_jobs”	O(1)
Paso 2 (iterar sobre las llaves del mapa *data_structs*)	O(M)
Paso 3 (función get)	O(N/M)
Paso 4 (función getValue)	O(1)
Paso 5 (iterar sobre la lista del SC)	O(N)
Paso 6 (función AddLast)	O(1)
Paso 7 (ordenamiento con merge)	O(NlogN)
Paso 8 (funciones para sacar sublista)	O(1)
TOTAL	***O(NlogN)***

Pruebas Realizadas

Input:

Ofertas a imprimir: 5, Código de país: ES, nivel de experticia: mid

Procesadores	**AMD Ryzen 7 5300U with Radeon Vega Mobile Gfx 2.30 GHz**
Memoria RAM	**16,0 GB**
Sistema Operativo	**Windows 11**

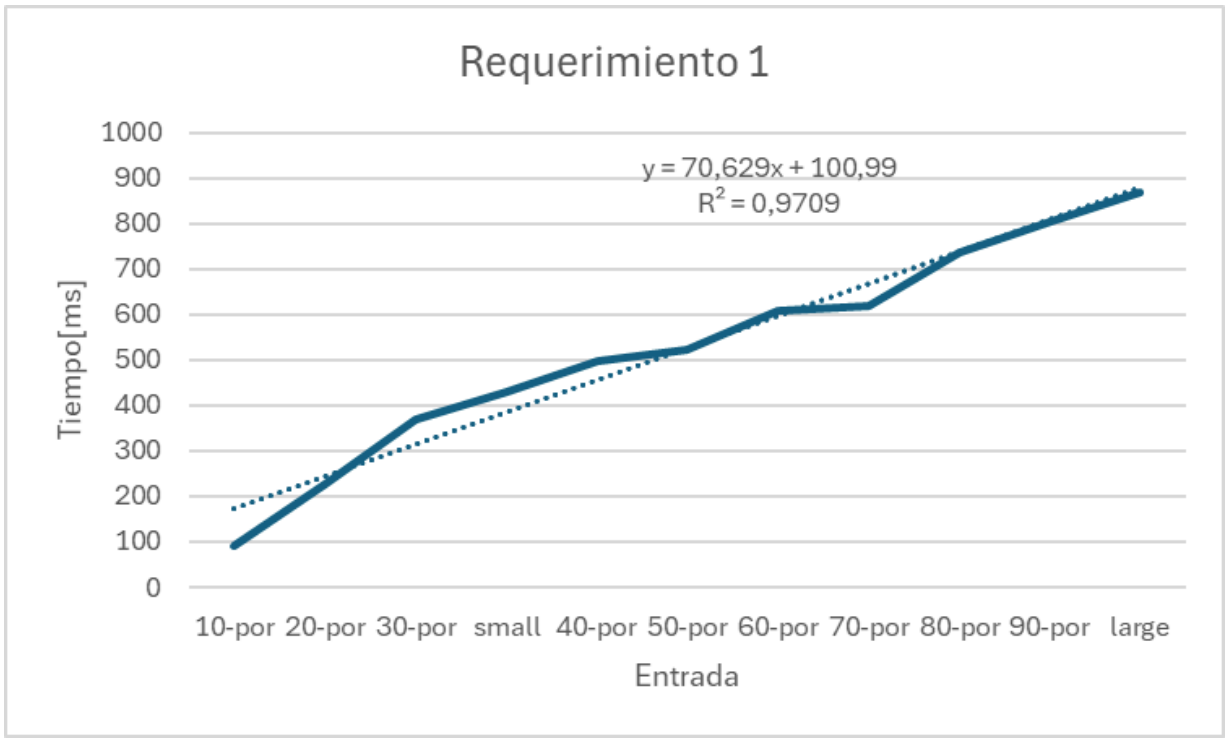
Tablas de datos

Las tablas con la recopilación de datos de las pruebas

Entrada	Tiempo (ms)
10-por	91.76
20-por	227.52
30-por	370.62
small	430.46
40-por	495.82
50-por	523.19
60-por	607.29
70-por	617.60
80-por	735.6
90-por	805.07
large	867.42

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

En la gráfica de cantidad de datos cargados vs el tiempo de ejecución del requerimiento es posible observar una relación lineal entre ambas variables, que es respaldado por el coeficiente de correlación R^2 de 0.97. Además tal como se pudo ver en el análisis de complejidad, el orden de crecimiento está

dado por N, que es el número de tuplas y específicamente para nuestros mapas es el número de ofertas de trabajo, por lo que hace sentido que al aumentar este número, así mismo aumente el tiempo de ejecución.

Requerimiento <<2>>

```
def req_2(data_structs, offer_number, company_name, city):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    offers = lt.newList("ARRAY_LIST")
    final = lt.newList("ARRAY_LIST")

    for i in lt.iterator(mp.keySet(data_structs)):
        if company_name in i and city in i:
            filterr = mp.get(data_structs, i)
            adds = me.getValue(filterr)
            for offer in lt.iterator(adds):

                    lt.addLast(offers, offer)



    #if lt.size(offers) >= 10:
    if lt.size(offers) >= 10:
        sb1 = lt.subList(offers, 1, 5)
        for zc in lt.iterator(sb1):
            lt.addLast(final, zc)
```

```
        sb2 = lt.subList(offers, (lt.size(offers)-5), 5)
        for zt in lt.iterator(sb2):
            lt.addLast(final, zt )


       # primeros cinco = lt.addLast(final, lt.subList(offers, 1, 5))
       # ultimos_cinco = lt.addLast(final, lt.subList(offers,
lt.size(offers)-5, lt.size(offers)))




    elif lt.size(offers) < 10 and lt.size(offers)>0 :
        if lt.size(offers)>= int(offer_number):
            final = lt.subList(offers, 1, int(offer_number))
        else:
            final = offers
    else:
        final = None
    return final, lt.size(offers)
```

Descripción

En este requerimiento se filtran las ofertas de trabajo por empresa y ciudad y devuelvo el número de ofertas que el usuario quiere ver. Primero se generan dos arreglos de nombre “offers” y “final”, el primero cumplirá la función de almacenar todas las ofertas de trabajo que cumplan con los filtros y la segunda solo va a contener el número de ofertas a imprimir. Itero con lt.iterator sobre las ofertas de trabajo para luego pasarlas por los filtros de compañía y ciudad utilizando la llave compleja que desarrollamos para el mapa, finalmente si pasan ambos filtros buscamos la pareja llave valor y depsues solo el valor para añadirlo, como estamos utilizando separate chaining las ofertas estan guardadas en listas entonces iteramos una vez más sobre esa lista para añadir esas ofertas correctamente.

Al final se especifica muy claro qué hacer si hay más o menos de diez ofertas, entonces si hay más de diez ofertas separamos las cinco primeras y las cinco últimas y las juntamos en un arreglo para imprimirlas. Si hay menos de diez ofertas pero al menos hay ofertas, imprimimos el número de ofertas deseadas por el usuario pero si no son suficientes solo imprimimos las que haya.

Entrada	El catálogo en el apartado de "Jobs" (en model), el número de ofertas a imprimir, el nombre de la compañía, la ciudad de la consulta.
Salidas	Las ofertas filtradas por compañía y ciudad.
Implementado (Sí/No)	Si se implementó y lo realizo el estudiante Lucas Valbuena Leon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (crear un nuevo arreglo)	O(1)
Paso 2 (iterar sobre el mapa *data_structs*)	O(N)
Paso 3 (función get)	O(N/M)
Paso 4 (función getValue)	O(1)
Paso 5 (iterar sobre la lista del SC)	O(N/M)
Paso 6 (función AddLast)	O(1)
Paso 7 (funciones para sacar sublista)	O(1)
TOTAL	***O(N)***

Pruebas Realizadas

Input:

Ofertas a imprimir: 4, ciudad: Warszawa, nombre de la compañía: Bitfinex.

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

Tablas de datos

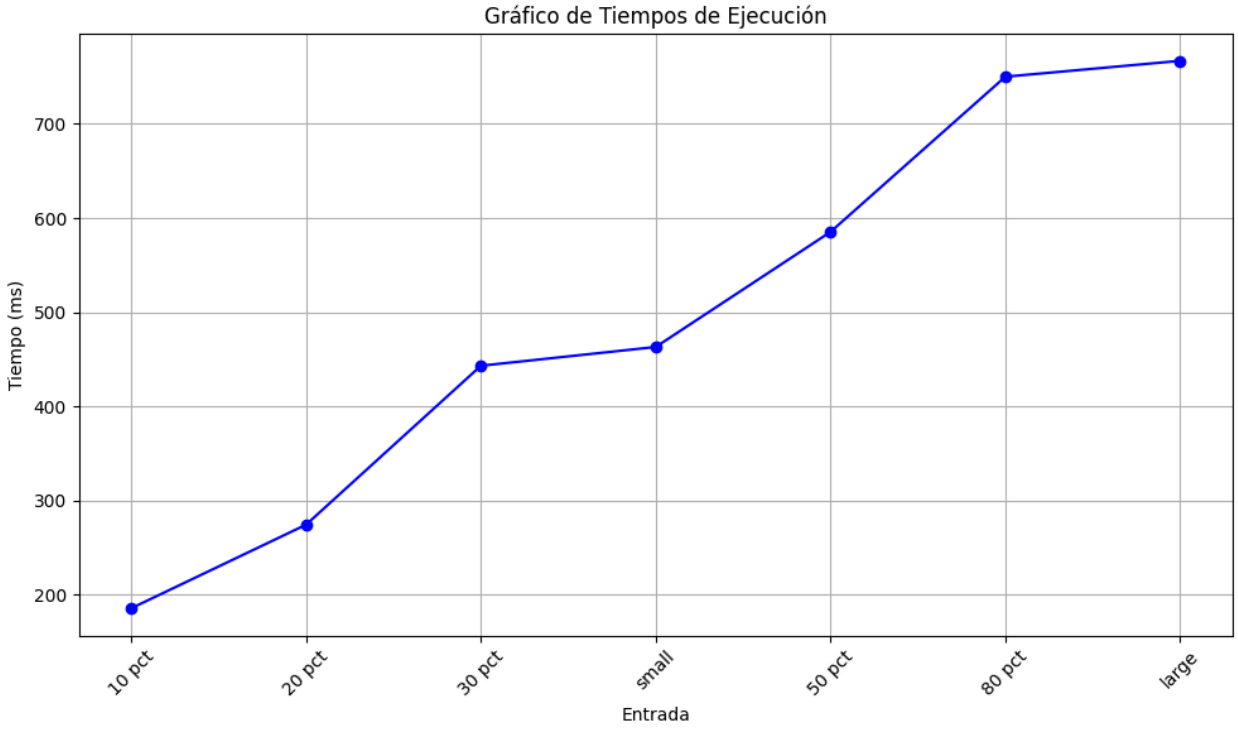
Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (ms)**
10 pct	185.390

20 pct	274.182
30 pct	443.054
small	463.002
50 pct	585.309
80 pct	750.170
large	767.042

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El análisis de los resultados denota una clara tendencia hacia una complejidad enteramente lineal y siempre va a seguir con esa misma tendencia ya que va a ejecutar el código estrictamente por la cantidad de datos que se le pasen, para después sacar sublistas, por lo tanto solo se puede esperar una complejidad lineal O(N). Si nos remitimos a la complejidad espacial podríamos esperar una complejidad O(1) ya que las operaciones que se manejan tienen un costo de O(1), pero igual va a seguir iterando todo el mapa para pasar los filtros y el tamaño de los datos afecta de manera transversal todo. También cabe destacar que los tiempos de ejecución utilizando mapas fueron significativamente más reducidos que solo utilizando arreglos.

Requerimiento <<3>>

```
def req_3(data_structs,empresa,fecha1,fecha2):
```

```
    """
    Función que soluciona el requerimiento 3
    """
    catalogo= mp.keySet(data_structs["specific_jobs"])
    lista= lt.newList("ARRAY_LIST")
    fecha_1= datetime.strptime(fecha1,"%Y-%m-%d")
    fecha_2= datetime.strptime(fecha2,"%Y-%m-%d")
    contador_mid=0
    contador_senior=0
    contador_junior=0
    mid=str("mid")
    junior= str("junior")
    senior= str("senior")


    for key in lt.iterator(catalogo):
        keysplit= key.split(";")
        fecha_of=datetime.strptime(keysplit[4],"%Y-%m-%d")
        if empresa == keysplit[2]:
            if fecha_of >= fecha_1 and fecha_of <= fecha_2 and
keysplit[3]== mid:
                pareja=mp.get(data_structs["specific_jobs"],key)
                valor=me.getValue(pareja)
                for element in lt.iterator(valor):
                    lt.addLast(lista, element)
                    contador_mid += 1
            elif fecha_of >= fecha_1 and fecha_of <= fecha_2 and
keysplit[3]== junior:
```

```
                pareja=mp.get(data_structs["specific_jobs"],key)
                valor=me.getValue(pareja)
                for element in lt.iterator(valor):
                    lt.addLast(lista, element)
                    contador_junior += 1
            elif fecha_of >= fecha_1 and fecha_of <= fecha_2 and
keysplit[3]== senior:
                pareja=mp.get(data_structs["specific_jobs"],key)
                valor=me.getValue(pareja)
                for element in lt.iterator(valor):
                    lt.addLast(lista, element)
                    contador_senior += 1



    lista_final=merg.sort(lista,sort_crit_req3)
    return
contador_junior,contador_mid,contador_senior,lt.size(lista_final),lista_fi
nal
```

Descripción

En este requerimiento se filtran las ofertas de trabajo que publicó una empresa en un periodo de tiempo específico y además retorna los contadores para cada uno de los niveles de experticia. Primero,se obtiene la lista de llaves del mapa “specific_jobs” y se le asigna a la variable “catalogo”,se genera un arreglo de nombre “lista” donde serán almacenadas todas las ofertas de trabajo que cumplan con los filtros, se convierten las fechas al formato deseado usando el módulo datetime y se inicializan las variables para los 3 contadores. Posteriormente, se Itera con lt.iterator sobre cada llave compuesta del mapa, y se comprueba si la empresa que entró como parámetro haga parte de dicha llave, de ser asi, obtenemos el valor respectivo de la llave para iterar sobre él y observar si la oferta fue publicada en el

rango de fechas deseado. Por último, cada oferta que es filtrada se agrega a la lista a medida que se actualizan los contadores.

Entrada	El catálogo en el apartado de “Jobs” (en model), la empresa de la consulta, la fecha inicial a consultar y la fecha final a consultar.
Salidas	Las ofertas cronológicamente organizadas y filtradas publicadas por una empresa en un rango de fechas,además de los contadores para cada uno de los niveles de experticia
Implementado (Sí/No)	Si se implementó y lo realizó Alisson Moreno

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 crear un nuevo arreglo, contadores y strptime()	O(1)
Paso 2 (iterar sobre las llaves del mapa *data_structs*)	O(M)
Paso 3 (función get)	O(N/M)
Paso 4 (función getValue)	O(1)
Paso 5 (iterar sobre la lista del SC)	O(N)
Paso 6 (función AddLast)	O(1)
Paso 7 (ordenamiento con merge)	O(NlogN)
Paso 8 (funciones para sacar sublista)	O(1)
TOTAL	***O(NlogN)***

Pruebas Realizadas

Input:

empresa: Tpay, fecha1: 2022-01-01,fecha2: 2023-01-01

Procesadores	**AMD Ryzen 7 5300U with Radeon Vega Mobile Gfx 2.30 GHz**
Memoria RAM	**16,0 GB**
Sistema Operativo	**Windows 11**

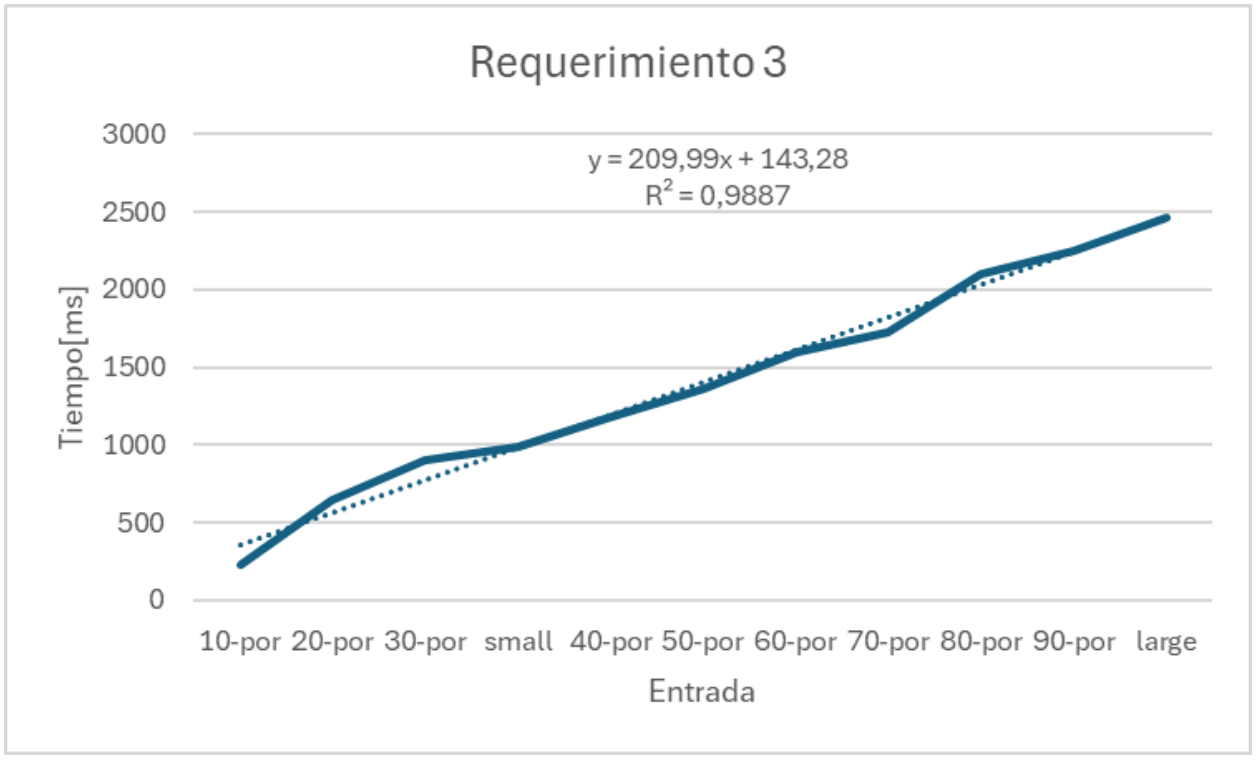
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	224.49
20-por	642.90
30-por	901.87
small	988.33
40-por	1179.74
50-por	1360.23
60-por	1599.52
70-por	1727.75
80-por	2101.87
90-por	2249.24
large	2459.47

Gráficas

Las gráficas con la representación de las pruebas realizadas



Análisis

Al observar el comportamiento del tiempo de ejecución frente a la muestra de datos cargados se puede afirmar que es casi enteramente lineal, y su complejidad depende de forma proporcionalmente directa de la cantidad de datos que se cargan, de modo que a medida que estos aumentan también aumenta el tiempo de ejecución. La complejidad temporal del requerimiento en el peor caso está dada por el algoritmo de ordenamiento merge sort y es de NlogN, el cual resulta ser el más bajo en términos de tiempo comparado con otros algoritmos como shell sort.

Requerimiento <<4>>

Descripción

Para explicar que se hizo en el requerimiento 4 primero hay que entender que en la carga de datos se cargan cinco índices: “employments_types”, “jobs”, “specific_jobs”, “skills” y “multi locations”. Para este requerimiento solo se usó el índice de “specific jobs” donde cada llave tiene la siguiente estructura “pais_ciudad_empresa_lvl experiencia fecha” donde fecha solo contiene “YYYY-MM-DD”. En este requerimiento se solicita buscar ofertas dadas en un país y un rango de fechas entregado por el usuario. Entonces en primera instancia se itera por las llaves dentro del índice “specific jobs” como se ve a continuación:

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


    for key in lt.iterator(mp.keySet(data_structs)):
```

En este caso ‘data structs’ ya es una instancia del catálog[‘specific jobs’] por lo que iteramos por las llaves usando mp.keySet(). A continuación el código:

```
def req_4(data_structs, pais, fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4


    #-----------------------------------------
    conteo_empresas = 0
    #-----------------------------------------
```

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

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

        fechaOf = datetime.strptime(key[-10:] ,"%Y-%m-%d")
        #if key[-10:] == "2023-06-28":
        #    print("stop")
        if pais in key:
            if fechaOf.year >= fecha_inicial[0] and fechaOf.year <=
fecha_final[0]:
                if fechaOf.month >= fecha_inicial[1] and fechaOf.month <=
fecha_final[1]:
                    if (fechaOf.month*30 + fechaOf.day) >=
(fecha_inicial[1]*30 + fecha_inicial[2]) and (fechaOf.month*30 +
fechaOf.day) <= (fecha_final[1]*30 + fecha_final[2]):
                        lt.addLast(filtered_keys, key)

    if lt.size(filtered_keys) == 0:
        return False, False, False, False, False, False, False, False
    filtered_offers= lt.newList("ARRAY_LIST")

    empresas = mp.newMap(numelements= 1000, maptype="PROBING",
loadfactor=0.5)
    ciudades = mp.newMap(numelements= 700, maptype= "PROBING",
loadfactor=0.5)
```

```
    for key in lt.iterator(filtered_keys):
        key_splited = key.split(";")
        if len(key_splited) > 5:
            print("habemus un problema")

        mp.put(empresas, key_splited[2], 0)

        existscity = mp.contains(ciudades, key_splited[1])
        if not existscity:
            mp.put(ciudades, key_splited[1],lt.newList("ARRAY_LIST"))



        entry = mp.get(data_structs, key)
        entry_ciudad = mp.get(ciudades, key_splited[1])

        lst = me.getValue(entry)
        lst_ciudad = me.getValue(entry_ciudad)

        for offer in lt.iterator(lst):
            lt.addLast(filtered_offers, offer)
            lt.addLast(lst_ciudad, offer)

    max_city = None
    min_city = None
    maxc = 0
```

```
    minc = None
    cant_ciudades = 0


    for city in lt.iterator(mp.keySet(ciudades)):
        entry = mp.get(ciudades, city)
        #print(city)
        if entry != None:
            cant_ciudades +=1
            cityOf = me.getValue(entry)


            if minc == None:
                minc = lt.size(cityOf)
                min_city = city


            size = lt.size(cityOf)


            if size > maxc:
                maxc = size
                max_city = city
            if size <= minc:
                minc = size
                min_city = city


    sort_crit = get_sort_crit("req4")


    sort(filtered_offers, sort_crit)
```

```
    #for offer in lt.iterator(filtered_offers):
    #    print(offer["published_at"][:10], offer["company_name"])



    cant_empresas = lt.size(mp.keySet(empresas))

    cant_ofertas = lt.size(filtered_offers)



    return cant_empresas, cant_ciudades, cant_ofertas,
max_city,lt.size(me.getValue(mp.get(ciudades,max_city))), min_city,
lt.size(me.getValue(mp.get(ciudades,min_city))), filtered_offers
```

Como se vio anteriormente, al iterar por las llaves de 'specific_jobs' y poder filtrar las ofertas que nos interesan en este caso se pregunta si el país que dio es usuario está en la llave, y luego se hace lo mismo con la fecha. Luego de completar el ciclo, una lista de tipo ARRAY_LIST contiene las llaves filtradas. A continuación procedemos a iterar por las llaves filtradas, se crean mapas nuevos, uno de ciudades y uno de empresas. El mapa de empresas simplemente sirve para llevar un conteo general de las empresas que hubo en total durante la consulta ya que por ciudad se repiten empresas presentes en otras ciudades. Una vez creado estos índices iniciales. Para inicializar una llave de una ciudad hacemos uso de mp.contains() para saber si la llave existe en el mapa, en el caso de que no exista, pone la llave en la tabla de hash cuyo valor va a ser un una lista de tipo ARRAY_LIST. Luego para poder manipular esa lista que se acabó de crear se usa me.getValue(mp.get(ciudades, llave_ciudad)) y se adiciona la oferta a una lista 'filtered_offers' ya que eventualmente se tendrá que hacer display de las primeras y últimas cinco ofertas consultadas ordenadas por fecha. Además se adiciona a la lista de la 'ciudad' como tal la oferta también. Una vez las ofertas filtradas ya están guardadas dentro de la tabla de hash, itera por las llaves mp.keySet(ciudades) y con cada llave accede al diccionario. Esto para poder encontrar la mayor ciudad y la menor ciudad que lo pide el requerimiento. A continuación se comienzan a hacer una serie de comparaciones que permitan encontrar la ciudad máxima y la mínima. Finalmente se sortea la lista de ofertas filtradas con un criterio de ordenamiento que ordena por fecha cronológicamente. Una vez todo esto se completo se retornan los siguientes datos:

```
cant_empresas, cant_ciudades, cant_ofertas,max_city ,ofertas_max,
min_city,ofertas_min, filtered_offers
```

Entrada	Data_structs, país, fecha inicial y fecha final. Donde Data_structs corresponde con el model en la llave “specific_jobs”.
Salidas	`cant_empresas, cant_ciudades, cant_ofertas,max_city ,ofertas_max, min_city,ofertas_min, filtered_offers`
Implementado (Sí/No)	Sí se implementó, lo implemento Juan José Cortés Villamil

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad (M = tamaño del mapa, N= número de tuplas)**
Iterar por el keySet de “specific_jobs”	O(N)
condicionales para hacer efectiva la filtración por llaves	O(1)
lt.addLast(Filteres_keys, key)	O(1)
Condicional para verificar si se filtraron ofertas o no	O(1)
mp.newMap() (creación de un mapa de empresas y de ciudades)	O(1)
Iterar por las llaves filtradas	O(N)
mp.contains(ciudades, ciudad) (“PROBING”)	O(N)
mp.put(ciudades, ciudad, lt.newList())	O(N) en el peor caso
mp.get()	O(N) en el peor caso
iterar sobre las ofertas dentro de la lista que es el valor de la llave filtrada	O(N) en el peor caso, aunque con datos reales muy difícilmente va a pasar esto. En la práctica hay un promedio de 3 ofertas por llave, con excepciones.
lt.addLast()	O(1)
Iterar por las ciudades (keySet) que contienen las ofertas filtradas por ciudad	O(N)
me.get()	O(M) en el peor caso, pero la función de has de DISClib hace una distribución uniforme de las llaves.
me.getValue()	O(1)
condicionales para encontrar la ciudad con mayor y menor cantidad de ofertas filtradas.	O(1)
sort(filteres_offers, sort_crit)	N log(N)
lt.size()	O(1)
return	O(1)

Manipular las primeras y últimas cinco ofertas	O(1)
TOTAL	***#O(N (N/M) log(N)) en el peor caso*** ***O(N log(N)) en el mejor caso*** ***O(N) teniendo en cuenta el mejor caso de mergeSort()***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Input:

país = PL, fecha_inicial = 2022-04-12, fecha_final = 2023-12-24.

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

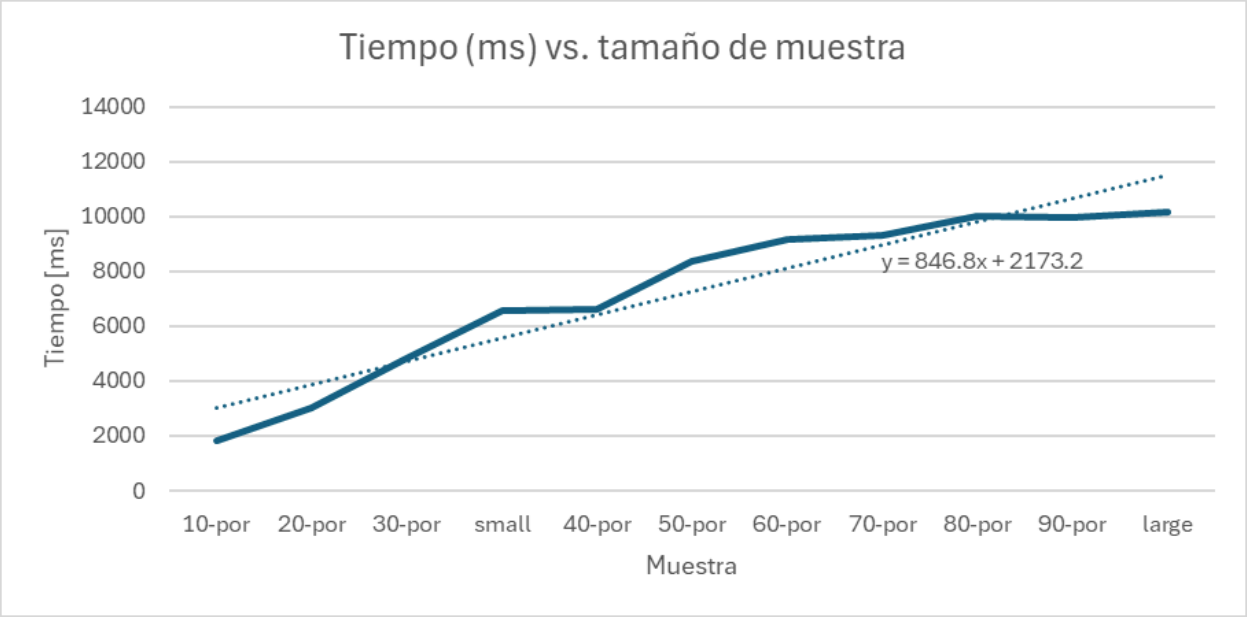
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	1803.2860999982804
20-por	3026.979699999094
30-por	4838.372800001875
small	6556.8265999984
40-por	6608.530200000852
50-por	8360.752900000662
60-por	9143.190199999139
70-por	9319.925199998543
80-por	9991.642799999565
90-por	9969.9104999993
large	10174.42620000057

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Teniendo en cuenta el análisis de complejidad, estamos ante el mejor caso O(N) pues el crecimiento es lineal. Puedo decir esto porque en las pruebas realizadas se evidencia cómo en promedio cuando el tamaño de la muestra aumenta un 10%, el tiempo de ejecución aumenta alrededor de 1000 ms. Es interesante que a pesar de tener una serie de ciclos 'for' anidados no se multiplican las complejidades porque en el caso de los datos que se manejan, no se va a dar un caso en el que haya N ofertas en cada llave filtrada, por lo que iterar por 3 o 5 ofertas realmente no hace gran diferencia en la complejidad del algoritmo.

Requerimiento <<5>>

```
def req_5(data_structs, city, first_date, last_date):



    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    offers = lt.newList("ARRAY_LIST")
    final = lt.newList("ARRAY_LIST")
```

```
primer_limite = first_date
primer_limite0= datetime.strptime(primer_limite, "%Y-%m-%d")


ultimo_limite = last_date
ultimo_limite0 = datetime.strptime(ultimo_limite, "%Y-%m-%d")


for i in lt.iterator(mp.keySet(data_structs)):
    if city in i:
        slide = i.split(";")
        slide = slide[-1]
        fecha_dt = datetime.strptime( slide, "%Y-%m-%d")
        if primer_limite0<= fecha_dt and fecha_dt<= ultimo_limite0:
            filterr = mp.get(data_structs, i)
            adds = me.getValue(filterr)
            for offer in lt.iterator(adds):


                lt.addLast(offers, offer)



if lt.size(offers) !=0:
    sort_crit = get_sort_crit("req4")


    sort(offers, sort_crit)
```

```
    if lt.size(offers) >= 10:
        sb1 = lt.subList(offers, 1, 5)
        for zc in lt.iterator(sb1):
            lt.addLast(final, zc)



        sb2 = lt.subList(offers, (lt.size(offers)-5), 5)
        for zt in lt.iterator(sb2):
            lt.addLast(final, zt )




    elif lt.size(offers)<10 and lt.size(offers)>0:
        final = offers
    elif lt.size(offers)==0:
        final = None


    size = lt.size(offers)


    return final, size
```

Descripción

Se inicializa un nuevo arreglo con el nombre de “lista” donde se almacenarán las ofertas que pasen los filtros, después se pasan todos las fechas a formato datetime.datetime (tanto los que dio el usuario como los que dio el arreglo) porque se encuentran en formato str. Iteramos con lt. iterator sobre el arreglo que pasa por parámetro y aplicamos los filtros de ciudad y de rango de fechas, si pasan ambos filtros se utilizan las funciones get y getValue para solo obtener la oferta y poder guardarla en el arreglo. Como estamos trabajando en separate chaining también tenemos que ingresar a la lista del valor de la llave para asegurarnos que se guarden todas las ofertas se encuentren o no en un bucket.

Así como en el requerimiento 2, estamos con la condición de saber cuántas ofertas se guardaron para saber si sacar una sublista o no y en los dos requerimientos pasamos el diez desde el model porque después el size adquiere un diferente valor si lo sacamos en controller.

Entrada	El catálogo en el apartado de “Jobs” (en model), la ciudad de la consulta, la fecha inicial a consultar y la fecha final a consultar.
Salidas	Las ofertas cronológicamente organizadas y filtradas por compañía y ciudad.
Implementado (Sí/No)	Si se implementó y lo realizo el estudiante Lucas Valbuena Leon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (crear dos nuevos arrays)	O(1)
Paso 2 (iterar sobre el mapa *data_structs*)	O(N)
Paso 3 (función get)	O(N/M)
Paso 4 (función getValue)	O(1)
Paso 5 (iterar sobre la lista del SC)	O(N/M)
Paso 6 (función AddLast)	O(1)
Paso 7 (función sort con merge)	O (N Log N)
Paso 8 (funciones para sacar sublista)	O(1)
TOTAL	***O(N)***

Pruebas Realizadas

Input:

Fecha inicial: 2023-01-12, Fecha final: 2023-01-23, Ciudad: Warszawa.

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

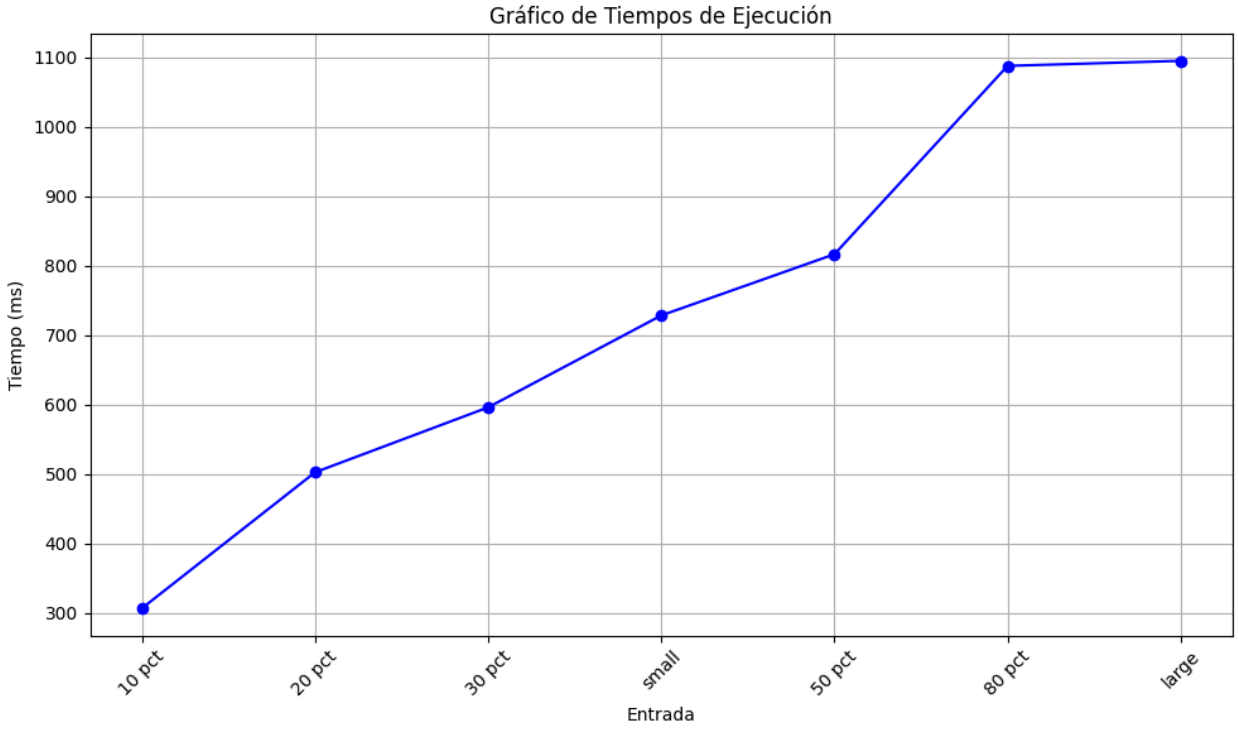
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10 pct	306.992
20 pct	502.618
30 pct	596.145
small	728.400
50 pct	816.446
80 pct	1087.481
large	1094.743

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento muestra un análisis de tendencia lineal muy parecido al requerimiento 2 porque están implementados de manera casi idéntica, con la única diferencia de que este requerimiento necesitaba ser organizado cronológicamente a través de merge sort y una función de comparación. La complejidad a nivel espacial es similar a la temporal, la única que generaría un poco más de complejidad, pero no significativa, sería sacar las sublistas al final y eso sería en el peor caso.

Requerimiento <<6>>

```
def req_6(data_structs,n,año,experticia):


    """
    Función que soluciona el requerimiento 6
    """
    catalogo= mp.keySet(data_structs["specific_jobs"])
    lista= lt.newList("ARRAY_LIST")
    empresas = mp.newMap(numelements= 1000, maptype="PROBING",
loadfactor=0.5)
    ciudades = mp.newMap(numelements= 700, maptype= "PROBING",
loadfactor=0.5)


    for key in lt.iterator(catalogo):
        keysplit= key.split(";")
        fecha=keysplit[4]
        añoF,mesF,diaF= fecha.split("-")
        añoF= int(añoF)
        if experticia != str("indiferente"):
            if año == añoF and experticia == keysplit[3]:
                parejaE=mp.get(data_structs["specific_jobs"],key)
                valorE=me.getValue(parejaE)
                for e in lt.iterator(valorE):
                    lt.addLast(lista, e)
        else:
            if año == añoF:
                parejaE=mp.get(data_structs["specific_jobs"],key)
```

```
                valorE=me.getValue(parejaE)
                for e in lt.iterator(valorE):
                    lt.addLast(lista, e)


    lista_salario=encontrar_salario(lista,data_structs)
    add_mapas(lista_salario,ciudades,empresas)
    mayor_menor_ciudad= max_min(ciudades)
    mayor_menor_empresa= max_min(empresas)
    lista_P=lista_ciudades(ciudades)
    sorted_list=merg.sort(lista_P,sort_crit_req6)


    if lt.size(sorted_list) >= n:
        sublista= lt.subList(sorted_list,1,n)
    else:
        sublista=sorted_list



    return
sublista,mayor_menor_ciudad,mayor_menor_empresa,lt.size(lista_P)


def lista_ciudades(ciudades):


    lista_ciudades=mp.keySet(ciudades)
    lista_nueva= lt.newList("ARRAY_LIST")
    mapa_empresa_ciudad= mp.newMap(numelements= 1000, maptype="PROBING",
loadfactor=0.5)
    for ciudad in lt.iterator(lista_ciudades):
```

```
        maximo=0
        minimo=float("inf")
        mejor= None
        peor=None
        pareja=mp.get(ciudades,ciudad)
        valor= me.getValue(pareja)
        llave=me.getKey(pareja)
        salario=0
        for e in lt.iterator(valor):
            dict={}
            size=lt.size(valor)
            empresa= e["company_name"]
            salario += e["salario"]
            pais= e["country_code"]
            if e["salary_to"]:
                salary_to= float(e["salary_to"])
                if salary_to>maximo:
                    maximo= salary_to
                    mejor=e["title"]
            if e["salary_from"]:
                salary_from= float(e["salary_from"])
                if salary_from < minimo:
                    minimo= salary_from
                    peor= e["title"]
            if mp.contains(mapa_empresa_ciudad,empresa) == False:
                valorT=lt.newList("ARRAY_LIST")
```

```
                lt.addLast(valorT,e)
                mp.put(mapa_empresa_ciudad,empresa,valorT)
            else:
                parejaE=mp.get(mapa_empresa_ciudad,empresa)
                valorE=me.getValue(parejaE)
                lt.addLast(valorE,e)



        mejor_empresa,peor_empresa,conteo_mejor,conteo_menor,num_empresas=
max_min(mapa_empresa_ciudad)


        promedio= salario/lt.size(valor)
        dict["salario"]=promedio
        dict["total_ofertas"]=size
        dict["ciudad"]=llave
        dict["pais"]=pais
        dict["mejor_oferta"]=mejor
        dict["peor_oferta"]=peor
        dict["Empresa_con_mas_ofertas"]=mejor_empresa
        dict["conteos"]=conteo_mejor
        dict["total_empresas"]=num_empresas
        lt.addLast(lista_nueva,dict)


    return lista_nueva



def add_mapas(lista,ciudades,empresas):
```

```
    for element in lt.iterator(lista):
        ciudad=element["city"]
        if mp.contains(ciudades,ciudad) == False:
            valorL=lt.newList("ARRAY_LIST")
            lt.addLast(valorL,element)
            mp.put(ciudades,ciudad,valorL)
        else:
            pareja=mp.get(ciudades,ciudad)
            valorC=me.getValue(pareja)
            lt.addLast(valorC,element)

        empresa=element["company_name"]
        if mp.contains(empresas,empresa) == False:
            valorT=lt.newList("ARRAY_LIST")
            lt.addLast(valorT,element)
            mp.put(empresas,empresa,valorT)
        else:
            parejaE=mp.get(empresas,empresa)
            valorE=me.getValue(parejaE)
            lt.addLast(valorE,element)


def max_min(mapa):
    maximo=0
    VariableMX= None
    VariableMN=None
    minimo=float("inf")
```

```
    llaves=mp.keySet(mapa)
    sizeTotal=lt.size(llaves)
    for llave in lt.iterator(llaves):
        pareja= mp.get(mapa,llave)
        valorP=me.getValue(pareja)
        size=lt.size(valorP)
        if size > maximo:
            maximo=size
            VariableMX=str(llave)

        if size < minimo:
            minimo=size
            VariableMN=str(llave)

    return VariableMX,VariableMN,maximo,minimo,sizeTotal


def encontrar_salario(lista,data_structs):
    for element in lt.iterator(lista):
        Global=0
        id= element["id"]
        mapa= data_structs["employments_types"]
        pareja=mp.get(mapa,id)
        info= me.getValue(pareja)
        for j in lt.iterator(info):
            salario_from= j["salary_from"]
            salario_to=j["salary_to"]
```

```
            if j["salary_from"] and j["salary_to"]:
                Global += (float(j["salary_from"]) +
float(j["salary_to"]))/2


            elif j["salary_from"]:
                Global += float(j["salary_from"])
            elif j["salary_to"]:
                Global += float(j["salary_to"])
            else:
                Global += 0.0


        promedio = Global/lt.size(info)
        element["salario"] = promedio
        element["salary_from"]=salario_from
        element["salary_to"]=salario_to


    return lista
```

Descripción

En este requerimiento se filtran las ofertas de trabajo teniendo en cuenta un año y un nivel de experticia específico y retorna; la mejor y la peor empresa con el respectivo conteo de ofertas y de igual forma para empresas, el número de ofertas filtradas solicitadas por el usuario en formato de una lista de ciudades que contiene información sobre el salario promedio,número de ofertas, pais, empresa con mas ofertas con su conteo y número de empresas. En primera instancia ,se obtiene la lista de llaves del mapa “specific_jobs” y se le asigna a la variable “catálogo”,se genera un arreglo de nombre “lista” donde serán almacenadas todas las ofertas de trabajo que cumplan con los filtros y se crean 2 mapas, uno para empresas y otro para ciudades de tipo “PROBING”. Luego, se Itera con lt.iterator sobre cada llave compuesta del mapa, y se obtiene el año para cada uno, luego comprobamos que tipo de experticia nos dieron como parámetro, pues si es indiferente es importante tomar todos los niveles en cuenta.Hecho, esto se agregan todas las ofertas filtradas a la lista creada anteriormente, sin embargo es importante hacer uso de otras funciones auxiliares que nos permitirán encontrar el salario promedio para cada una de lsa ofertas, añadir información a los mapas que creamos y encontrar las ciudades y empresas con

mayor número de ofertas publicadas. Al final la lista resultante de una de estas funciones auxiliares lista_ciudades() es ordenada con merge sort.

Entrada	El catálogo en el apartado de “specific_Jobs” (en model), el número de ofertas a imprimir, el año y el nivel de experticia.
Salidas	Lista de ciudades ordenadas desde la ciudad con mayor cantidad de ofertas publicadas filtradas por el año y nivel de experticia,el total de ofertas publicadas y la ciudad y empresa con mayor y menor cantidad de ofertas
Implementado (Sí/No)	Si se implementó y lo realizó Alisson Moreno

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 crear un nuevo arreglo y mapas	O(1)
Paso 2 (iterar sobre las llaves del mapa *data_structs*)	O(M)
Paso 3 (función get)	O(N/M)
Paso 4 (función getValue)	O(1)
Paso 5 (iterar sobre la lista del SC)	O(N)
Paso 6 (función AddLast)	O(1)
Paso 7 (ordenamiento con merge)	O(NlogN)
Paso 8 (Función encontrar_salario())	O(N)
Paso 9 (Función add_mapas())	O(N)
Paso 10 (Funcion max_min())	O(M)
Paso 11 (Funcion lista_ciudades())	O(N)
Paso 12 (funciones para sacar sublista)	O(1)
TOTAL	***O(NlogN)***

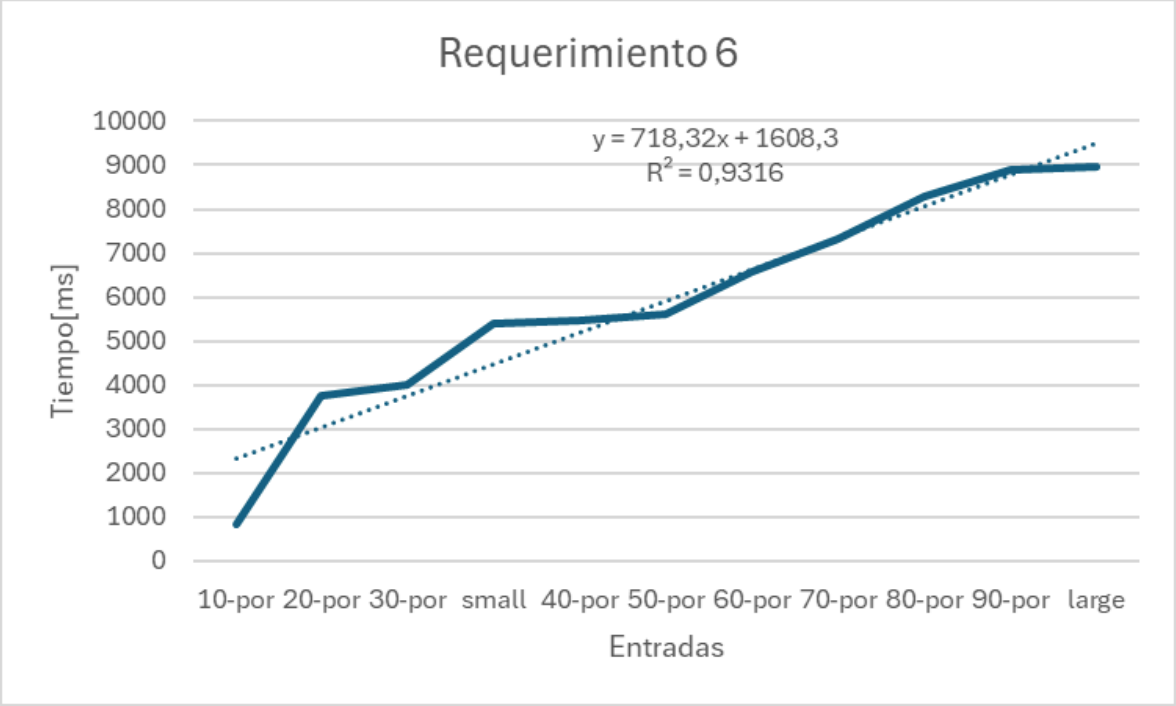
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	824.92
20-por	3747.90
30-por	4001.68
small	5398.32
40-por	5462.95
50-por	5619.5
60-por	6591.98
70-por	7339.08
80-por	8284.03
90-por	8877.69
large	8952.61

Gráficas

Las gráficas con la representación de las pruebas realizadas



Análisis

Este requerimiento muestra una naturaleza lineal ya que en principio tiene la misma lógica que la mayoría de los requerimientos, a excepción de acceder al archivo employment_types, que se usó para poder obtener la información de los salarios de acuerdo al id de las ofertas previamente filtradas. Pese a

este cambio, y la adición de varias funciones auxiliares la orden de crecimiento temporal también está dado por el merge sort, que en esta ocasión no ordena cronológicamente sino por el número de ofertas por cada ciudad.

Requerimiento <<7>>

```
def req_7(data_structs, country_number, month, year):

    """

    Función que soluciona el requerimiento 7

    """

    # TODO: Realizar el requerimiento 7

###############################################################################
###########################################

###############################################################################
###########################################

################### Incializar lo que vamos a utilizar
#############################################################

###############################################################################
###########################################

###############################################################################
###########################################

    offers = lt.newList("ARRAY_LIST")

    dicc_country = {}

    conteo_countries= {}

    dicc_final = {}

    lista_final = lt.newList("ARRAY_LIST")

    dicc_experience = {}

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

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

##########################################################################
##########################################

################################### Filtrar las ofertas por mes y año
##########################################################

##########################################################################
##########################################

##########################################################################
##########################################

    for skills_it in
lt.iterator(mp.keySet(data_structs["specific_jobs"])):

        slide = skills_it.split(";")

        slide = slide[-1]

        fecha_dt = datetime.strptime( slide, "%Y-%m-%d")

        mes = fecha_dt.month

        anio = fecha_dt.year

        if int(mes)== int(month) and int(year) == int(anio):

            filterr = mp.get(data_structs["specific_jobs"], skills_it)


            adds = me.getValue(filterr)

            for offer in lt.iterator(adds):


                lt.addLast(offers, offer)

    #Ya quedo esta parte, NO TOCAR

##########################################################################
##########################################

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

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

##########################################################################
##########################################

################################ Contar cuantos paises aparecen y cuantas
veces ###################################

##########################################################################
##########################################

##########################################################################
##########################################

    #for a in offers["elements"]:

        #for i in a["elements"]:

    for i in lt.iterator(offers):

            if i["country_code"] not in dicc_country:

                lista = []

                lista.append(i)

                dicc_country[i["country_code"]] = lista

            else:

                dicc_country[i["country_code"]].append(i)

            if i["country_code"]not in conteo_countries:

                conteo_countries[i["country_code"]]=1

            else:

                conteo_countries[i["country_code"]]+=1
##########################################################################
##########################################

##########################################################################
##########################################

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

```
############################ Sacar solo el numero de paises que nos piden
#########################################

#########################################################################
#########################################

#########################################################################
#########################################

    if len(conteo_countries)>0:

        for i in range(int(country_number)):

            if len(conteo_countries)>0:

                maximo = max(conteo_countries.values())

                for i in conteo_countries.items():

                    if i[1]==maximo:

                        nombre_maximo = i[0]

                dicc_final[nombre_maximo]=maximo

                conteo_countries.pop(nombre_maximo)
#########################################################################
#########################################

#########################################################################
#########################################

#########################################################################
#########################################

#########################################################################
#########################################

#########################################################################
#########################################

    for i in dicc_country.items():

        if i[0] in dicc_final:
```

```
            for j in i[1]:
                lt.addLast(lista_final, j)
    # si esta guardando en lista final
##########################################################################
##########################################
##########################################################################
##########################################
##########################################################################
##########################################
##########################################################################
##########################################


    for i in lt.iterator(lista_final):
        #for j in lt.iterator(mp.keySet(data_structs["skills"])):
##########################################################################
##########################################
##########################################################################
##########################################
##########################################################################
##########################################
##########################################################################
##########################################
            if "junior" == i["experience_level"]:
                if i["experience_level"] not in dicc_experience:
                    dicc_sec = {}
                    dicc_experience["junior"] = dicc_sec
                else:
                    dicc_sec = dicc_experience["junior"]
                """
```

```
                if i["id"] in j:
                    filterr_sk = mp.get(data_structs["skills"], j)
                    adds_sk = me.getValue(filterr_sk)
                    for b in adds_sk["elements"]:
                            #for b in a["name"]:
                """
                skill = me.getValue(mp.get(data_structs["skills"], i["id"]
))
                for sk_j in lt.iterator(skill):
                    #print(sk_j)
                    if sk_j["name"] not in dicc_sec:
                        dicc_sec[sk_j["name"]  ] = 1
                    else:
                        dicc_sec[sk_j["name"]  ] += 1
                dicc_experience["junior"] = dicc_sec




################################################################################
##############################################

################################################################################
##############################################

################################################################################
##############################################

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

```
            if "mid" == i["experience_level"]:
                if i["experience_level"] not in dicc_experience:
                    dicc_sec1 = {}
                    dicc_experience["mid"] = dicc_sec1
                else:
                    dicc_sec1 = dicc_experience["mid"]
                """
                if i["id"] in j:
                    filterr_sk1 = mp.get(data_structs["skills"], j)
                    adds_sk1 = me.getValue(filterr_sk1)
                    for b in adds_sk1["elements"]:
                            #for b in a["name"]:
                """
                skill = me.getValue(mp.get(data_structs["skills"], i["id"]
))
                for sk_jr in lt.iterator(skill):
                    #print(sk_jr)
                    if sk_jr["name"] not in dicc_sec1:
                        dicc_sec1[sk_jr["name"]  ] = 1
                    else:
                        dicc_sec1[sk_jr["name"] ] += 1
                dicc_experience["mid"] = dicc_sec1

##########################################################################
##############################################

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

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

##########################################################################
##########################################
            if "senior" == i["experience_level"]:
                if i["experience_level"] not in dicc_experience:
                    dicc_sec2 = {}
                    dicc_experience["senior"] = dicc_sec2
                else:
                    dicc_sec2 = dicc_experience["senior"]


                """
                if i["id"] in j:
                    filterr_sk2 = mp.get(data_structs["skills"], j)
                    adds_sk2 = me.getValue(filterr_sk2)
                    for b in adds_sk2["elements"]:
                            #for b in a["name"]:
                """
                skill = me.getValue(mp.get(data_structs["skills"], i["id"]
))
                for sk_jh in lt.iterator(skill):
                    #print(sk_jh)
                    if sk_jh["name"] not in dicc_sec2:
                        dicc_sec2[sk_jh["name"]  ] = 1
                    else:
                        dicc_sec2[sk_jh["name"]  ] += 1
```

```
                dicc_experience["mid"] = dicc_sec2


##########################################################################
##########################################
##########################################################################
##########################################
##########################################################################
##########################################
##########################################################################
##########################################


    return lista_final, dicc_final, dicc_experience
```

Descripción

Primero tomamos el arreglo original y lo iteramos con lt.iterator para filtrar dado un año y un mes. Después hacemos un diccionario que contenga las ofertas X país, a ese diccionario lo pasamos a otro diccionario con solo los N países que pide imprimir el usuario. Unimos toda la información en un arreglo con un filtro de entrada, el país se debe encontrar en el diccionario de N países y si es así agrega todas las ofertas de trabajo que tenga al arreglo, como llave se encuentra el nombre del país y como valor todas las ofertas de trabajo. Después de sacar el arreglo final buscamos los tres niveles de experticia solicitados en el mapa de skills iterando muy poco se organizan las skills en otro diccionario, su llave será el nivel de experticia y el valor será un diccionario compuesto por las habilidades y el número de veces que se repiten.

Entrada	El catálogo en el apartado de “Jobs” (en model), el año de la consulta, el mes de la consulta y el número de países a imprimir.
Salidas	Número de empresas, ofertas y skills por nivel de experticia dado un año y un mes de consulta.
Implementado (Sí/No)	Si se implementó y lo realizo el estudiante Lucas Valbuena Leon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (crear dos nuevos arrays e inicializar todo)	O(1)

Paso 2 (iterar sobre el mapa *data_structs["specific_jobs"]*)	O(N)
Paso 3 (función get)	O(N/M)
Paso 4 (función getValue)	O(1)
Paso 5 (iterar sobre *offers*)	O(N)
Paso 6 (función append)	O(1)
Paso 7 (función addLast de lista_final)	O(Z) Z siendo el número de países a imprimir
Paso 8 (iterar "Lista_final")	O(N)
Paso 9 (iterar "Skills" dentro de lista final)	O(1)
TOTAL	***O(N)***

Pruebas Realizadas

Input:

Año : 2023, mes: 2, número de países a imprimir: 4

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

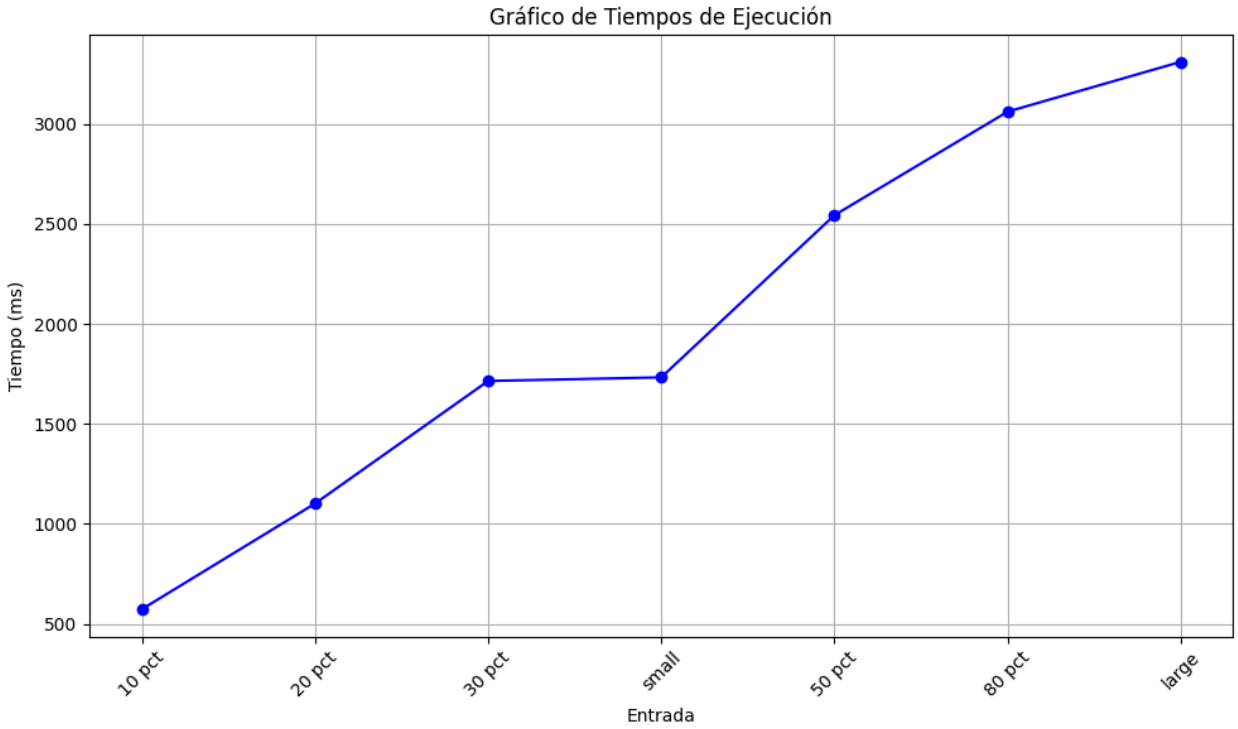
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10 pct	572.686
20 pct	1101.245
30 pct	1714.738
small	1732.719
50 pct	2543.993
80 pct	3061.865
large	3311.444

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El análisis de tendencia de este reto muestra que se comporta como una función lineal, acorde a su complejidad temporal O(N), comportándose igual a la mayoría de los requerimientos. Es una complejidad muy efectiva dado el número de veces que se estima que recorre cada una de las estructuras, tanto nativas como no nativas, y puede imprimir los resultados en muy poco tiempo. Su complejidad espacial es costosa por todas las estructuras auxiliares relacionadas con el reto.

Requerimiento <<8>>

Descripción

```
def req_8(catalog,  lvl_exp, divisa, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    filtered_keys_ET = lt.newList("ARRAY_LIST")
```

```
#for key_ET in lt.iterator(mp.keySet(catalog["employments_types"])):
#    if divisa in key_ET:
#        if lvl_exp != "indiferente":
#            if lvl_exp in key_ET:
#                lt.addLast(filtered_keys_ET, key_ET)
#        else:
#            lt.addLast(filtered_keys_ET, key_ET)

filtered_keys_JB = lt.newList("ARRAY_LIST")

for key_JB in lt.iterator(mp.keySet(catalog["specific_jobs"])):
    key_splited = key_JB.split(";")
    fechaOf = datetime.strptime(key_JB[-10:] ,"%Y-%m-%d")

    if fechaOf.year>= fecha_inicial[0] and fechaOf.year <=
       fecha_final[0]:
        if fechaOf.month >= fecha_inicial[1] and fechaOf.month <=
       fecha_final[1]:
            if (fechaOf.month*30 + fechaOf.day) >=
       (fecha_inicial[1]*30 + fecha_inicial[2]) and (fechaOf.month*30
       + fechaOf.day) <= (fecha_final[1]*30 + fecha_final[2]):
                if lvl_exp == key_splited[3]:
                    lt.addLast(filtered_keys_JB, key_JB)
                elif lvl_exp == "indiferente":
                    lt.addLast(filtered_keys_JB, key_JB)
```

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


c= CurrencyConverter()
if lt.size(filtered_keys_JB) == 0:

    return False, False, False ,False, False, False, False, False,
       False



paises = mp.newMap(numelements=200, maptype= "PROBING",
       loadfactor=0.5)
empresas = mp.newMap(numelements=1000, maptype="PROBING",
       loadfactor=0.5)
ciudades = mp.newMap(numelements= 2000, maptype="PROBING",
       loadfactor=0.5)
ofertas_con_salario = lt.newList("ARRAY_LIST")
ofertas_sin_salario = lt.newList("ARRAY_LIST")




#for key_Jb in lt.iterator(filtered_keys_JB):
for key_Jb in lt.iterator(filtered_keys_JB):
    key_splited = key_Jb.split(";")

    existscountry = mp.contains(paises,key_splited[0])
```

```
    if not existscountry:
        mp.put(paises, key_splited[0], mp.newMap(numelements=1000,
        maptype="PROBING", loadfactor=0.5))
        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "promedio_salarial", 0)
        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "suma_salarios", 0)
        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "cantidad_salarios", 0)

        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "promedio_habilidades", 0)
        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "suma_habilidades", 0)
        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "cantidad_habilidades", 0)

        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "empresas_presentes", mp.newMap(numelements=1000,
        maptype="PROBING", loadfactor= 0.5))

        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "mayor_oferta", (None,None))
        mp.put(me.getValue(mp.get(paises, key_splited[0])),
        "menor_oferta", (None,None))

    existscity = mp.contains(me.getValue(mp.get(paises,
        key_splited[0])), key_splited[1])
    if not existscity:
```

```
            mp.put(me.getValue(mp.get(paises, key_splited[0])),
            key_splited[1], mp.newMap(numelements=1000, maptype="PROBING",
            loadfactor=0.5))

            mp.put(me.getValue(mp.get(me.getValue(mp.get(paises,
            key_splited[0])),key_splited[1])), "con_salario",
            lt.newList("ARRAY_LIST"))

            mp.put(me.getValue(mp.get(me.getValue(mp.get(paises,
            key_splited[0])),key_splited[1])), "sin_salario",
            lt.newList("ARRAY_LIST"))

            mp.put(me.getValue(mp.get(me.getValue(mp.get(paises,
            key_splited[0])),key_splited[1])), "con_salario_fijo",
            lt.newList("ARRAY_LIST"))



        for offer in
            lt.iterator(me.getValue(mp.get(catalog["specific_jobs"],
            key_Jb))):

            employment_type =
            me.getValue(mp.get(catalog["employments_types"], offer["id"]))

            skill = me.getValue(mp.get(catalog["skills"], offer["id"]))




            for i in range(1, lt.size(skill)+1):


                skill_level = lt.getElement(skill, i)

                mp.put(me.getValue(mp.get(paises,
            key_splited[0])),"suma_habilidades",
            me.getValue(mp.get(me.getValue(mp.get(paises,
            key_splited[0])),"suma_habilidades")) +
            int(skill_level["level"]))
```

```
            mp.put(me.getValue(mp.get(paises,
        key_splited[0])),"cantidad_habilidades",
        me.getValue(mp.get(me.getValue(mp.get(paises,
        key_splited[0])),"cantidad_habilidades")) + 1)


        salario_promedio_oferta = 0

        conteo_salario = 0


        for i in range(1, lt.size(employment_type)+1):

            #salario_usd =


            if lt.getElement(employment_type, i)["currency_salary"] ==
        divisa or lt.getElement(employment_type, i)["currency_salary"]
        == "":

                mp.put(empresas, key_splited[2], 0)

                mp.put(me.getValue(mp.get(me.getValue(mp.get(paises,
        key_splited[0])),"empresas_presentes")),key_splited[2], 0)

                mp.put(ciudades, key_splited[1], 0)

                print(mp.keySet(ciudades))

                #print(offer)


                if lt.getElement(employment_type,i)["fijo"] == False :

                    salario_usd =
        convertir_divisas(lt.getElement(employment_type,i)["promedio_s
        alarial"],
        lt.getElement(employment_type,i)["currency_salary"],c)

                    salario_promedio_oferta += salario_usd #Salario
        promedio por oferta

                    conteo_salario += 1
```

```
                mp.put(me.getValue(mp.get(paises,
key_splited[0])),"suma_salarios",
me.getValue(mp.get(me.getValue(mp.get(paises,
key_splited[0])),"suma_salarios")) + salario_usd)

                mp.put(me.getValue(mp.get(paises,
key_splited[0])),"cantidad_salarios",
me.getValue(mp.get(me.getValue(mp.get(paises,
key_splited[0])),"cantidad_salarios")) + 1)

                if i == 1:


lt.addLast(me.getValue(mp.get(me.getValue(mp.get(me.getValue(m
p.get(paises,
key_splited[0])),key_splited[1])),"con_salario")), offer)


            elif lt.getElement(employment_type,i)["fijo"] == True
:

                salario_usd =
convertir_divisas(lt.getElement(employment_type,i)["promedio_s
alarial"],
lt.getElement(employment_type,i)["currency_salary"],c)

                salario_promedio_oferta += salario_usd  #Salario
promedio por oferta

                conteo_salario += 1




                mp.put(me.getValue(mp.get(paises,
key_splited[0])),"suma_salarios",
me.getValue(mp.get(me.getValue(mp.get(paises,
key_splited[0])),"suma_salarios")) + salario_usd)

                mp.put(me.getValue(mp.get(paises,
key_splited[0])),"cantidad_salarios",
me.getValue(mp.get(me.getValue(mp.get(paises,
key_splited[0])),"cantidad_salarios")) + 1)
```

```
                if i == 1:


lt.addLast(me.getValue(mp.get(me.getValue(mp.get(me.getValue(m
p.get(paises,
key_splited[0])),key_splited[1])),"con_salario_fijo")), offer)









            else:
                if i == 1:


lt.addLast(me.getValue(mp.get(me.getValue(mp.get(me.getValue(m
p.get(paises,
key_splited[0])),key_splited[1])),"sin_salario")), offer)


 if conteo_salario != 0:
     salario_promedio_oferta =
salario_promedio_oferta/conteo_salario
 if  me.getValue(mp.get(me.getValue(mp.get(paises,
key_splited[0])),"mayor_oferta"))[0] == None:
     mp.put(me.getValue(mp.get(paises,
key_splited[0])),"mayor_oferta", (salario_promedio_oferta,
offer))
 if  me.getValue(mp.get(me.getValue(mp.get(paises,
key_splited[0])),"menor_oferta"))[0] == None:
     if salario_promedio_oferta !=0:
         mp.put(me.getValue(mp.get(paises,
key_splited[0])),"menor_oferta", (salario_promedio_oferta,
offer))
```

```
            if salario_promedio_oferta >
        me.getValue(mp.get(me.getValue(mp.get(paises,
        key_splited[0])),"mayor_oferta"))[0]:

            mp.put(me.getValue(mp.get(paises,
        key_splited[0])),"mayor_oferta", (salario_promedio_oferta,
        offer))

            if me.getValue(mp.get(me.getValue(mp.get(paises,
        key_splited[0])),"menor_oferta"))[0] != None:

                if salario_promedio_oferta <
        me.getValue(mp.get(me.getValue(mp.get(paises,
        key_splited[0])),"menor_oferta"))[0] and
        salario_promedio_oferta !=0:

                    mp.put(me.getValue(mp.get(paises,
        key_splited[0])),"menor_oferta", (salario_promedio_oferta,
        offer))




#                       A PARTIR DE AQUÍ SE COMIENZA LA MANIPULACIÓN
        DE LOS RESULTADOS DE LA CONSULTA


cant_of_con_salario = 0

cant_of_con_salario_fijo = 0

cant_of_sin_salario = 0


paises_orden = lt.newList("ARRAY_LIST")


for key_country in lt.iterator(mp.keySet(paises)):

    if key_country == None:

        print("ABBIAMO UN PROBLEMA")
```

```
country = me.getValue(mp.get(paises, key_country))
if me.getValue(mp.get( country, "suma_salarios")) != 0 and
    me.getValue(mp.get( country, "cantidad_salarios")) != 0:
    mp.put(country, "promedio_salarial", (me.getValue(mp.get(
    country, "suma_salarios"))/me.getValue(mp.get( country,
    "cantidad_salarios"))))

if me.getValue(mp.get( country, "suma_habilidades")) != 0 and
    me.getValue(mp.get( country, "cantidad_habilidades")) != 0:
    mp.put(country, "promedio_habilidades", (me.getValue(mp.get(
    country, "suma_habilidades"))/me.getValue(mp.get( country,
    "cantidad_habilidades"))))

#print(f"{me.getValue(mp.get( country, 'empresas_presentes'))}
    --------------- Esta es")
mp.put(country, "empresas_presentes" ,
    lt.size(mp.keySet(me.getValue(mp.get( country,
    "empresas_presentes")))))

if mp.isEmpty(country):
    mp.remove(paises, key_country)


lt.addLast(paises_orden , (key_country, me.getValue(mp.get(
    country, "promedio_salarial"))))
borrar = ""
for key_city in lt.iterator(mp.keySet(country)):
    no
    =["suma_salarios","cantidad_salarios","promedio_salarial","can
```

```
        tidad_habilidades","suma_habilidades","promedio_habilidades",
        "mayor_oferta", "menor_oferta", "empresas_presentes"]

        if key_city not in no:

            cant_of_con_salario +=
        lt.size(me.getValue(mp.get(me.getValue(mp.get(country,
        key_city)), "con_salario")))

            cant_of_con_salario_fijo +=
        lt.size(me.getValue(mp.get(me.getValue(mp.get(country,
        key_city)), "con_salario_fijo")))

            cant_of_sin_salario +=
        lt.size(me.getValue(mp.get(me.getValue(mp.get(country,
        key_city)), "sin_salario")))

            if lt.size(me.getValue(mp.get(me.getValue(mp.get(country,
        key_city)), "con_salario"))) == 0 and
        lt.size(me.getValue(mp.get(me.getValue(mp.get(country,
        key_city)), "con_salario_fijo"))) == 0 and
        lt.size(me.getValue(mp.get(me.getValue(mp.get(country,
        key_city)), "sin_salario"))) == 0:

                borrar += "True"

            else:

                borrar += "False"


    if "False" not in borrar:

        mp.remove(paises, key_country)

        lt.removeLast(paises_orden)




total_ofertas = cant_of_sin_salario + cant_of_con_salario +
        cant_of_con_salario_fijo
```

```
    paises_orden = sort(paises_orden, sort_crit_req8)

    #for pais in lt.iterator(paises_orden):

    #    print(pais)

    return lt.size(mp.keySet(empresas)), total_ofertas, mp.size(paises) ,
           lt.size(mp.keySet(ciudades)), cant_of_con_salario,
           cant_of_con_salario_fijo, cant_of_sin_salario, paises_orden,
           paises
```

En primera instancia lo que hace este requerimiento es que va a filtrar las llaves de “specific_jobs” de acuerdo a un rango de fecha que entregó el usuario y también según un nivel de experiencia que este último también quiera consultar. Luego se inicializa el CurrencyConverter() que va a permitir más adelante cambiar el salario promedio de cada oferta a USD. Luego se verifica que hayan llaves filtradas, de no ser así significa que no se encontraron resultados entonces se retorna False para todos los parámetros que recibe Controller. Luego se inicializan mapas de países, ciudades y empresas que servirán para organizar la información más fácilmente como la necesita el usuario. A continuación se comienza a iterar por todas las llaves filtradas y se inicializa la llave de cada país a partir de cada oferta poniendo llaves al ARRAY_LIST que tiene como valor la llave para manipular información general del país como el promedio salarial, etc. Luego se hace lo mismo con cada ciudad. Acto seguido, se hace un for anidado por cada oferta que está en el ARRAY_LIST que tiene como valor la llave “key_Jb” y cada una sería una oferta filtrada per se. Luego de debuggear el programa me di cuenta de que cuando se hace la carga de datos, el apartado de skills se carga en índice por “id” pero resulta que hay entradas del CSV para las que se repite el id. Es por esto que a continuación se hace un tercer for anidado, y que si bien esto puede parecer alarmante para la complejidad no es el caso en vista de que el máximo de skills con mismo ID son dos o tres. Por lo que no va a representar grandes consecuencias en la complejidad. Luego de que se acaba ese tercer for anidado, se vuelve a hacer un “tercer” for anidado para los “employments_types” ya que sucede lo mismo que con skills, hay diferentes salarios para un mismo id, entonces hay que tenerlos en cuenta. En este for se segmentan las ofertas con salario fijo, con rango salarial, o sin salario de acuerdo con su id, y los ciclos anidados que se hicieron anteriormente. Finalmente se hacen una serie de comparaciones para tener la mayor oferta por país y se comienza a manipular la información obtenida de la consulta para retornar lo que necesita el usuario.

Entrada	`control["model"], lvl_exp, divisa, fecha_inicial, fecha_final`
Salidas	`cant_empresas, total_ofertas, cant_paises , cant_ciudades, cant_of_con_salario,`

	`cant_of_con_salario_fijo, cant_of_sin_salario, paises_orden, paises`
Implementado (Sí/No)	Sí. Implementado por Juan José Cortés Villamil

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad (M = tamaño de la tabla, N = número de tuplas)**
Iterar por las llaves del catálogo en “specific_jobs” (for)	O(N)
condicionales para filtrar por rango de fechas y nivel de experiencia (if)	O(1)
añadir la llave filtrada a una lista de llaves filtradas (lt.addLast())	O(1)
Instanciar el currency_converter (CurrencyConverter())	O(1)
Condicional para verificar que la búsqueda fue efectiva (encontró algo) (lt.size())	O(1)
Creación de nuevos mapas de países, empresas y ciudades para construir la estructura de datos que necesita el usuario. (mp.newMap())	O(1)
Iterar por las llaves filtradas	O(N)
Verificar si la llave del país existe (mp.contains())	O(N)
En el caso de que no exista la llave del país, entonces se añaden nuevas etiquetas dentro del mapa (que eventualmente contendrá las ciudades) que es el valor de la llave país. E.g. promedio_salarial, suma_salarios, etc. (mp.put())	O(N)
me.getValue(mp.get())	O(N)
Verificar si la llave de la ciudad existe (mp.contains())	O(N)
En el caso de que no exista la llave de la ciudad, entonces se inicializa la llave con un mapa como valor, donde cada ciudad va a tener tres llaves, “con_salario”, “sin_salario” y “con_salario_fijo”. (mp.put())	O(N)
Iterar por cada oferta que tiene el ARRAY_LIST que es valor de la llave “key_Jb” que fue filtrada.	O(M) en el peor de los casos
Obtener un array_list que contiene los employments types y un array_list que contiene las skills según el id de la oferta de la iteración. (array porque en la práctica hay más de un skill o employment type que corresponden a una misma ID.) me.getValue(mp.get())	O(N)

Iterar por las habilidades y con los tipos de empleo que coinciden con el ID de la oferta en la iteración	O(2) o O(3)
Ahora se filtra por divisa	O(1)
Convertir los salarios de la divisa de interés del usuario a USD (convertir_divisas())	O(1) en el mejor de los casos
Sumar a “suma_salarios” el salario en USD y sumarle 1 a “cantidad_salarios” para eventualmente poder obtener el promedio salarial por país. Se hace esto para las ofertas con salario fijo y con rango salarial. (mp.put())	O(N)
Comparaciones para encontrar mayor y menor oferta con base en su promedio salarial. me.getValue(mp.get()) y mp.put()	O(N)
iterar por el keySet() del mapa “paises”	Alrededor de O(195) hay 195 países en el mundo
Calcular el promedio salarial y de habilidades (mp.getValue(mp.get()) y mp.put()	O(N)
si el país está vacío significa que clasificó a la consulta por experticia y rango de fechas pero no por la divisa (mp.isEmpty())	O(1)
Si está vacío hay que quitar el país (mp.remove)	O(N)
Iterar por el keySet() del mapa que es el valor de la llave correspondiente al país en el que está la iteración	O(N)
sortear un ArrayList que tiene en cada posición tuplas de la forma (cod_país, promedio_salarial)	O(N log N)
TOTAL	***O(N log N) en el peor caso*** ***O(N) en el mejor caso***

Pruebas Realizadas

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

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

Tablas de datos

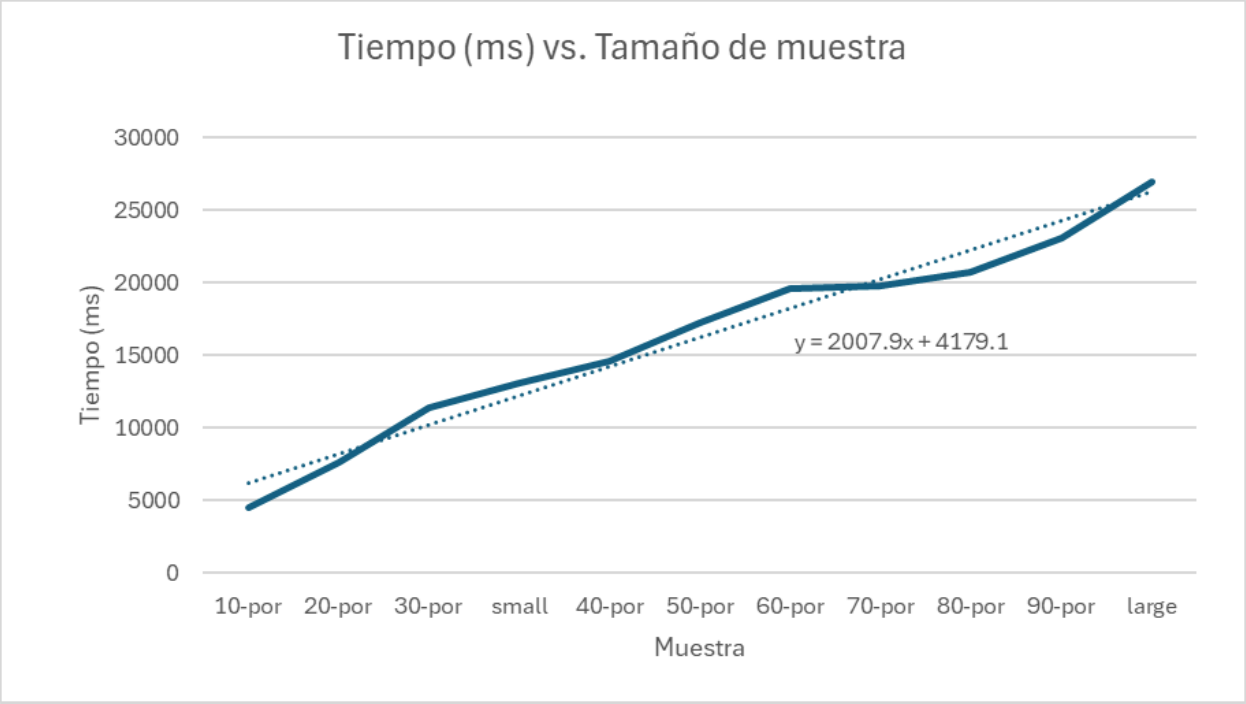
Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	4460.20380000025
20-por	7575.7699000015855

30-por	11410.865400001407
small	13051.004399999976
40-por	14605.77499999851
50-por	17253.965200003237
60-por	19598.701300002635
70-por	19757.394099995494
80-por	20756.133600000292
90-por	23082.011800002307
large	26941.29870000109

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Teniendo en cuenta el análisis de complejidad estamos ante el mejor caso pues el algoritmo tiene un comportamiento lineal O(N). Nuevamente pasa lo que pasaba con el requerimiento 4, se tienen una serie de ciclos for anidados que no repercuten significativamente porque si se itera por las skills o los employments son tres, máximo cuatro iteraciones más. Cosa que a gran escala es O(1). En un principio, cuando instancie el CurrencyConverter() fue más que nada porque si lo creaba en la función convertirDivisa() me di cuenta de que se demoraba muchísimo tiempo más, teorizó yo que al ser una librería, el llamar por cada iteración el CurrencyConverter() significaba más complejidad para traer una serie de recursos que consumía tiempo. En el caso de mp.put() o mp.get() que en el peor de los casos es

O(N), no se da este caso, probablemente porque la tabla es lo suficientemente grande y porque además si se diera, el tiempo de ejercicio sería mucho mayor.

.