**ANÁLISIS DEL RETO**

*Valeria Gutierrez Beltran 1, 202320878 1, v.gutierrezb 1*

*Andrea Aroca, 202320457, c.arocad*

*Juan David Calderon Gonzalez, 202320117, jd.calderong12*

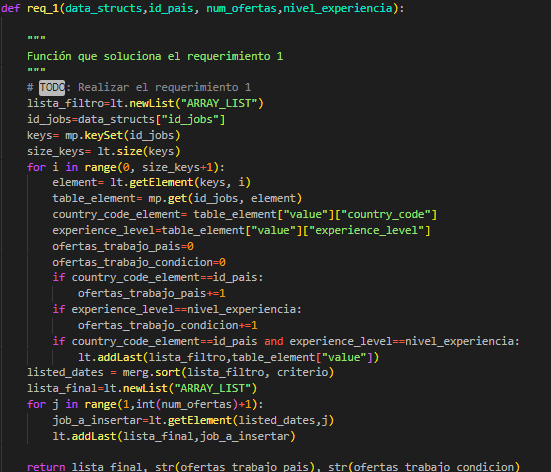
# **Requerimiento <<1>>**

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

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento

La función req\_1 se encarga de procesar ofertas de trabajo, filtrándolas según criterios específicos como el país (id\_pais), el número de ofertas a mostrar (num\_ofertas) y el nivel de experiencia (nivel\_experiencia). Inicialmente, se crea una lista tipo array (lista\_filtro) donde se almacenarán las ofertas que cumplan con estos criterios. Se accede a la tabla de hash id\_jobs desde data\_structs, la cual contiene los detalles de cada oferta. La función itera sobre las ofertas de trabajo: extrae las claves de id\_jobs, calcula su tamaño y utiliza un bucle para examinar cada una. En este bucle, se verifica si el país y el nivel de experiencia de cada oferta coinciden con los parámetros dados. Se identifican las ofertas que cumplen con ambos criterios y se agregan a lista\_filtro.La lista de ofertas seleccionadas se ordena según los criterios necesarios. Posteriormente, se crea otra lista (lista\_final) para contener el número exacto de ofertas requerido (num\_ofertas), seleccionando las primeras ofertas de la lista ordenada. Finalmente, la función retorna lista\_final, junto con los conteos de ofertas por país y por nivel de experiencia



| **Entrada** | Estructura de datos del modelo, id del país insertado por el usuario, número de ofertas que el usuario desea consultar, y el nivel de experticia. |
| --- | --- |
| **Salidas** | Lista tipo arraylist, con las ofertas filtradas por el id del país y el nivel de experticia. Dos strings que contienen el número de ofertas de trabajo del país y ofertas de trabajo por experiencia. |
| **Implementado (Sí/No)** | Si se implementó. Equipo |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Paso 1 | O(mp.keySet(id\_jobs))=O(n) |
| Paso 2 | O(size\_keys + 1)=O(n) |
| Paso 3 | O(merg.sort)= O(nlogn) |
| ***Paso 4*** | O(num\_ofertas bucle)= O(1) |
| ***Total*** | O(nlogn) |

## **Pruebas Realizadas**

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

| **Entrada** | **Tiempo (s)** |
| --- | --- |
| 10-por | 364254.20 |
| 20-por | 700589.32 |
| 30-por | 1085890.24 |
| 40-por | 1534827.37 |
| 50-por | 2389634.60 |
| 60-por | 2786331.29 |
| 70-por | 3298847.7 |
| 80-por | 3992938.2 |
| 90-por | 4583949.45 |
| large | 5938473.65 |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.fg

### **Graficas**

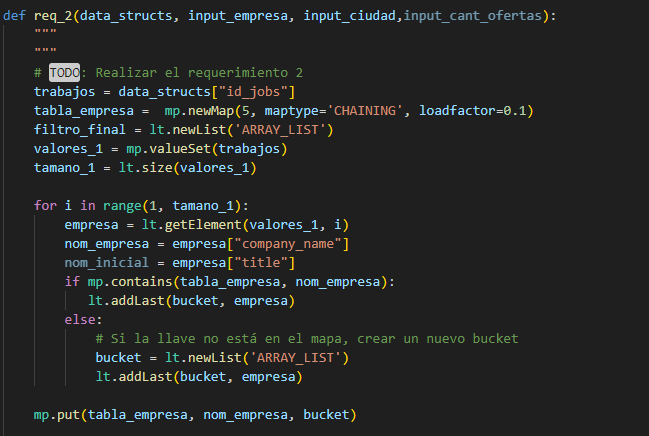
Las gráficas con la representación de las pruebas realizadas.

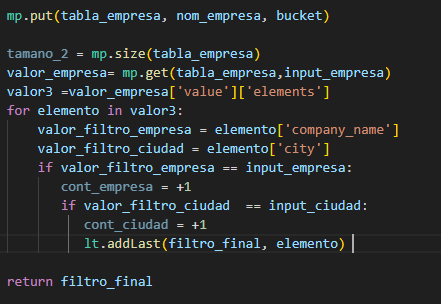
## **Análisis**

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

# **Requerimiento 2**

## **Descripción**





La función `req\_2` comienza inicializando las estructuras de datos necesarias, incluyendo una tabla hash llamada `tabla\_empresa` para organizar los trabajos por empresa, y una lista llamada `filtro\_final` que contendrá los trabajos filtrados al final del proceso.

Luego, se recorren todos los trabajos disponibles y se asignan a la tabla hash `tabla\_empresa`, donde cada llave en la tabla se asocia con una empresa específica. Se utiliza una técnica de encadenamiento para manejar colisiones, lo que significa que cada entrada en la tabla hash apunta a una lista que contiene todos los trabajos asociados con una empresa determinada. Esto organiza eficientemente los trabajos por empresa, facilitando el acceso rápido a los trabajos asociados con una empresa particular.

Después de organizar los trabajos por empresa, la función busca los trabajos asociados con la empresa especificada (`input\_empresa`). Si no hay trabajos asociados con la empresa proporcionada, el proceso de filtrado termina aquí y se devuelve una lista vacía, ya que no hay trabajos que cumplen con los criterios especificados.

Si se encuentran trabajos asociados con la empresa especificada, la función procede a verificar cada trabajo para determinar si se encuentra en la ciudad especificada (`input\_ciudad`) y si cumple con la cantidad mínima de ofertas requerida (`input\_cant\_ofertas`). Para cada trabajo que cumple con estos criterios, se agrega a la lista de trabajos filtrados `filtro\_final`.

Finalmente, una vez que se han filtrado todos los trabajos que cumplen con los criterios especificados, la función devuelve la lista de trabajos filtrados `filtro\_final`. Esta lista contiene todos los trabajos que satisfacen los criterios de empresa, ciudad y cantidad mínima de ofertas, proporcionando así una salida útil para el análisis posterior o para su presentación al usuario final.

| **Entrada** | estructura de datos, input empresa, input ciudad y cantidad ofertas a listar |
| --- | --- |
| **Salidas** | lista de trabajos filtrados |
| **Implementado (Sí/No)** | Si. Implementado por Andrea Aroca |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Inicialización de estructuras de datos | O(n) |
| Recorrido de los trabajos y almacenamiento en la tabla hash | O(n) |
| Obtención de los trabajos de la empresa especificada | O(n) |
| Filtrado de trabajos por ciudad | O(n) |
| Retorno de la lista de trabajos filtrados: | O(n) |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

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

| Procesadores | Intel Core I7 |
| --- | --- |
| Memoria RAM | 8 GB |
| Sistema Operativo | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
| --- | --- |
| small | 396785.30 |
| 5 pct | 801543.57 |
| 10 pct | 1245907.21 |
| 20 pct | 1648882.77 |
| 30 pct | 2075936.80 |
| 50 pct | 2589657.24 |
| 80 pct | 3389755.98 |
| large | 4357822.35 |

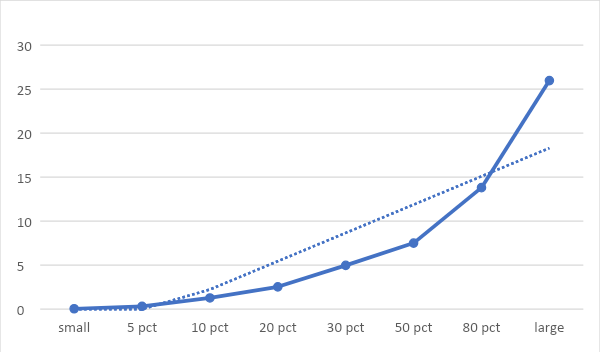
### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| small | Dato1 | 396785.30 |
| 5 pct | Dato2 | 801543.57 |
| 10 pct | Dato3 | 1245907.21 |
| 20 pct | Dato4 | 1648882.77 |
| 30 pct | Dato5 | 2075936.80 |
| 50 pct | Dato6 | 2589657.24 |
| 80 pct | Dato7 | 3389755.98 |
| large | Dato8 | 4357822.35 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.



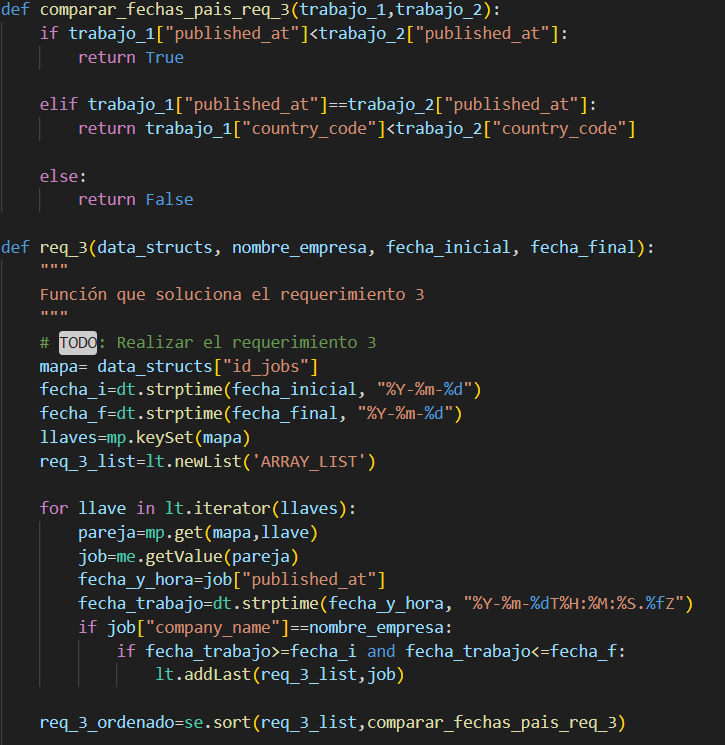
## **Análisis**

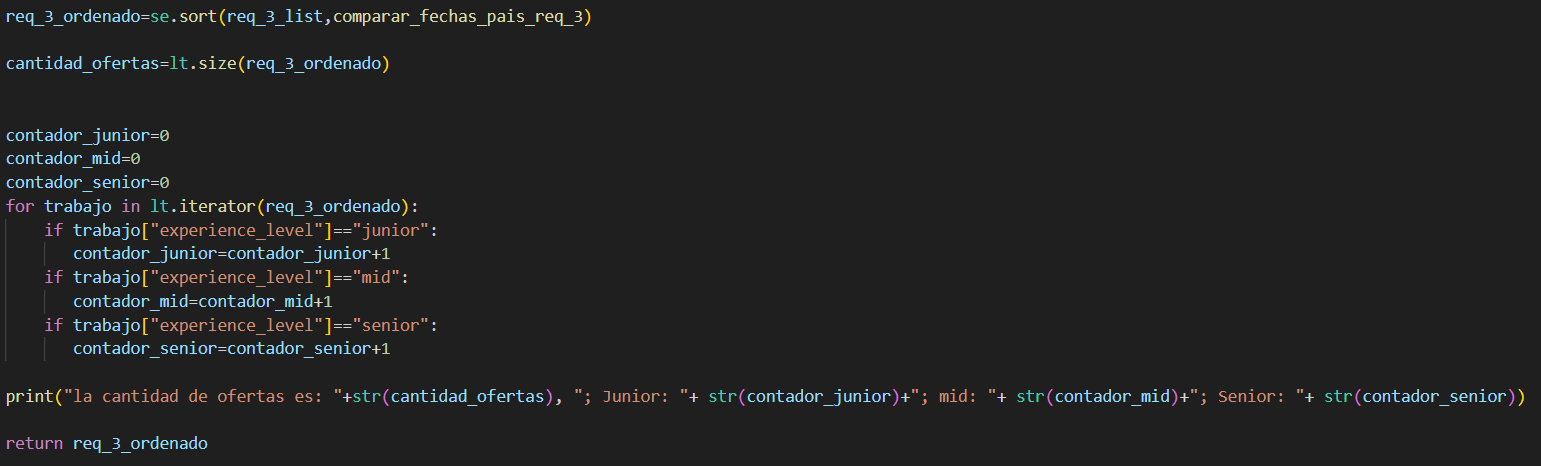
El análisis muestra que el algoritmo tiene un comportamiento diferente según el tamaño de los datos y el porcentaje de completitud, siendo más sensible al tamaño de los datos para entradas más grandes. Esto podría indicar que el algoritmo tiene una complejidad temporal que aumenta de manera significativa con el tamaño de los datos, lo que puede requerir optimizaciones para manejar eficientemente conjuntos de datos más grandes.

# **Requerimiento <<3>>**

## **Descripción**

Breve descripción de como abordaron la implementación del requerimiento





En el requerimiento 3, usando el nombre de una empresa y un rango de fechas dado por el usuario, recorre el hash map de trabajos y encuentra los trabajos que cumplen tanto con el nombre de la empresa que los publico y que estén en el rango de fechas indicado por el usuario (convirtiendo estas fechas de strings dentro del mapa a datetime). Si un trabajo cumple con ambas condiciones (nombre de empresa y está dentro del rango de fechas), la oferta es añadida a una nueva lista (ARRAY LIST) y el arreglo se ordena usando selección Sort de manera cronológica las ofertas de trabajo que quedaron en dicho arreglo (si dos fechas son iguales el orden es decidido alfabéticamente por el país de la oferta). Adicionalmente, la función también imprime la cantidad de ofertas y el número de ofertas Junior, Mid y Senior que cumplen con los requisitos solicitados por el usuario.

| **Entrada** | Estructura de datos del modelo  Nombre de la empresa.  La fecha inicial del periodo a consultar (con formato "%Y-%m-%d").  La fecha final del periodo a consultar (con formato "%Y-%m-%d"). |
| --- | --- |
| **Salidas** | • Número total de ofertas.  • Número total de ofertas con experticia junior.  • Número total de ofertas con experticia mid.  • Número total de ofertas con experticia senior.  • El listado de ofertas de la empresa ordenados cronológicamente por fecha y país (v.gr. Para dos ofertas con la misma fecha, el orden lo decide el país de forma alfabética). Donde para cada uno de los elementos resultantes contendrá la siguiente información:  o Fecha de la oferta.  o Título de la oferta.  o Nivel de experticia requerido  o Ciudad de la empresa de la oferta  o País de la empresa de la oferta  o Tamaño de la empresa de la oferta  o Tipo de lugar de trabajo de la oferta.  o Disponible a contratar ucranianos (Verdadero o Falso). |
| **Implementado (Sí/No)** | Si, implementado por Juan David Calderon |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| convertir fechas de str a datetime | O(1) |
| sacar llaves del mapa | O(1) |
| crear nuevo arreglo vacio | O(1) |
| recorrer mapa buscando trabajos dentro del rango de tiempo dado y que cumplan con la empresa escogida por el usuario | O(n) |
| ordenar arreglos cronologicamente | O(n2) |
| encontrar tamaño del arreglo ordenado | O(1) |
| Encontrar y contar las ofertas dependiendo de su nivel de experticia | O(1) |
| ***TOTAL*** | ***O(n2)*** |

## **Pruebas Realizadas/Tablas de datos**

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).

Parámetros:

Nombre Empresa= Comarch

fecha inicial=2023-03-01

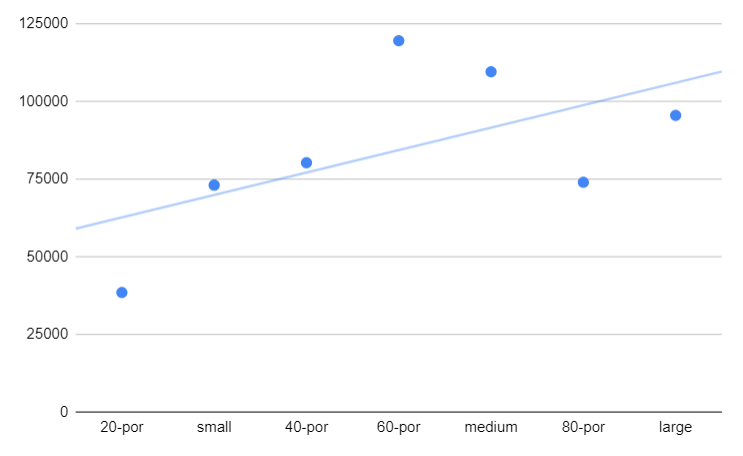
fecha final=2023-04-01

|  | **Máquina** |
| --- | --- |
| **Procesadores** | AMD Ryzen 5 2500U |
| **Memoria RAM (GB)** | 8GB |
| **Sistema Operativo** | Widows 10 |

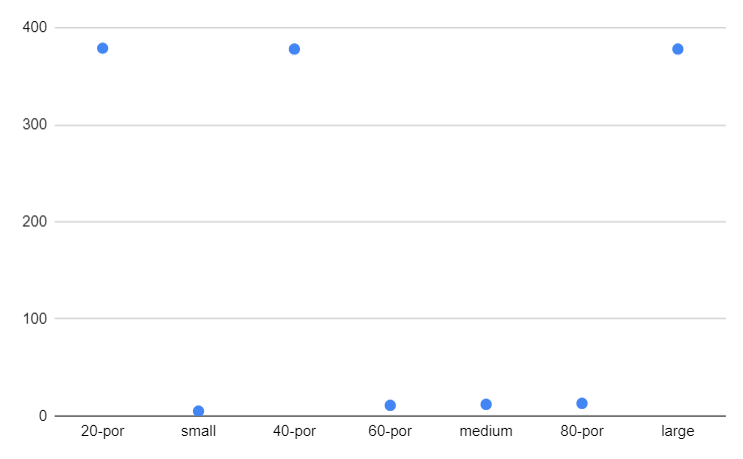
| **Entrada** | **Tiempo (ms)** | **Memoria (kb)** |
| --- | --- | --- |
| 20 | 38483 | 379.490234375 |
| 40 | 80278 | 378.431640625 |
| 60 | 119574 | 11.875 |
| 80 | 73974 | 13.0625 |
| small | 73045 | 5.015625 |
| medium | 109548 | 12.875 |
| large | 95497 | 378.47265625 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.



Grafica Tiempo



Grafica Memoria

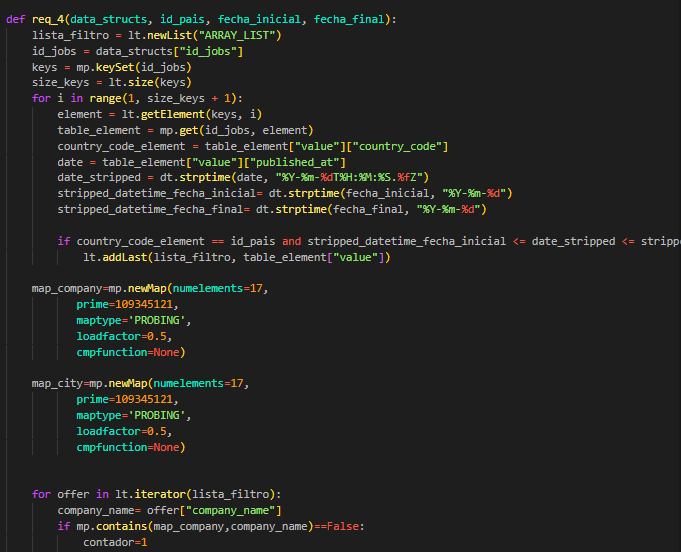
## **Análisis**

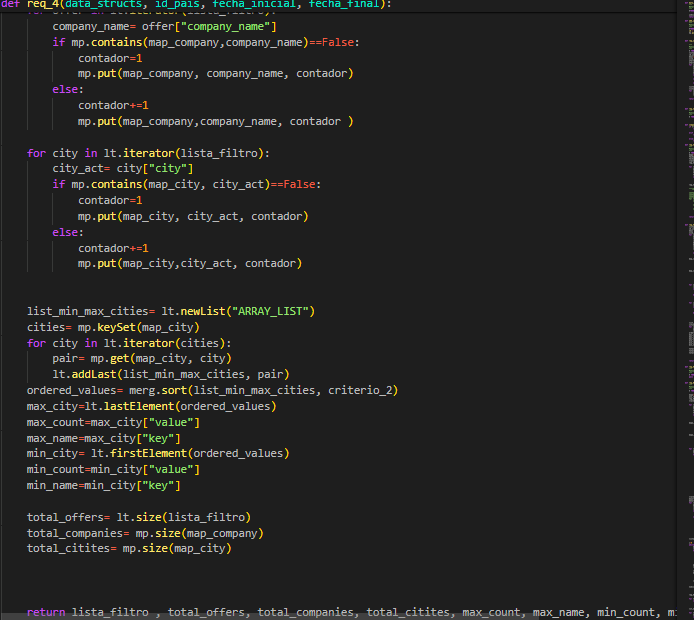
Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

El incremento del tiempo dependiendo de la cantidad de datos, aunque disperso, muestra en general y a gran escala un incremento, en donde se puede asumir que extrapolando los datos, entres más datos hallan, más tiempo en promedio se demorara el requerimiento en cargar. Esto se debe a la complejidad n2 de la función, la cual, no es visible con el rango datos tomados (estos muestran y sugieren una posible tendencia lineal).

La memoria cambia drásticamente en distintas cargas de datos; sin embargo, estas mantienen valores constantes en los bajos 10 kb y altos 370 kb de memoria. Esto propone que para los datos con poco uso de memoria, requirieron hacer menos rehashing, dando mapas más en términos de espacio más pequeños y mejor distribuidos que aquellos con un alto uso de memoria, lo cual propone que se extendió mucho el espacio alocado para los mapas.

**Requerimiento 4**

****

****

## **Descripción**

La función req\_4 filtra y analiza ofertas de trabajo dentro de un rango de fechas y un país específico, utilizando los datos de data\_structs. Inicialmente, se extraen y filtran las ofertas de trabajo por país y fecha, almacenando las coincidencias en lista\_filtro. Posteriormente, se procesan estas ofertas para analizar la distribución de las compañías y ciudades en las ofertas filtradas. Para cada compañía y ciudad presentes en las ofertas filtradas, se realiza un conteo, almacenando los resultados en los mapas map\_company y map\_city. Luego, se analiza la información de las ciudades para identificar aquella con el mayor y menor número de ofertas, resultando en max\_name y min\_name junto con sus respectivos conteos. Además, se calcula el total de ofertas, compañías y ciudades involucradas en las ofertas filtradas. Finalmente, la función retorna una lista de las ofertas filtradas junto con estadísticas relevantes como el total de ofertas, compañías, ciudades, y detalles sobre la ciudad con mayor y menor número de ofertas.

| Entrada |  |
| --- | --- |
| Estructura de datos del modelo | data\_structs |
| País | id\_pais |
| Fecha inicial del periodo a consultar (con formato "%Y-%m-%d") | fecha\_inicial |
| Fecha final del periodo a consultar (con formato "%Y-%m-%d") | fecha\_final |

| Salidas |  |
| --- | --- |
| Número total de ofertas | total\_offers |
| Número total de empresas | total\_companies |
| Número total de ciudades | total\_citites |
| Ciudad con mayor número de ofertas | max\_name y max\_count |
| Ciudad con menor número de ofertas | min\_name y min\_count |
| El resto de las salidas solicitadas no están directamente disponibles en la función dada. |  |

| Implementado (Sí/No) |  |
| --- | --- |
|  | Sí, implementado |

Analisis de complejidad

| Pasos | Complejidad |
| --- | --- |
| Creación de lista y mapa | O(1) |
| Iteración sobre las claves de id\_jobs | O(n) |
| Iteración sobre lista\_filtro | O(n) |
| Iteración para crear list\_min\_max\_cities | O(n) |
| mergesort | O(n log n) |
|  |  |
|  |  |
| TOTAL | O(n log n) |

## **Pruebas Realizadas/Tablas de datos**

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).

Parámetros:

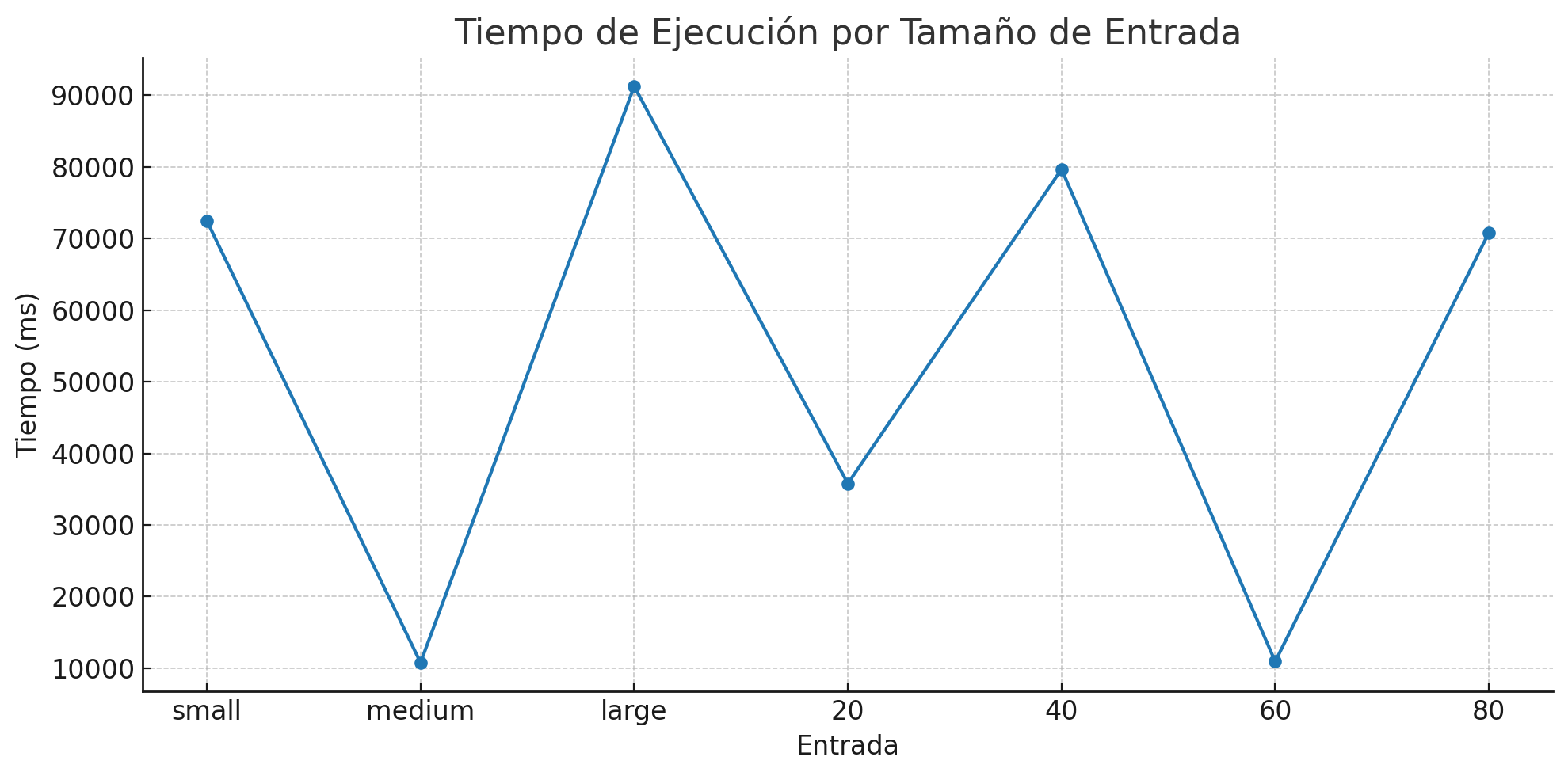
Pais=PL

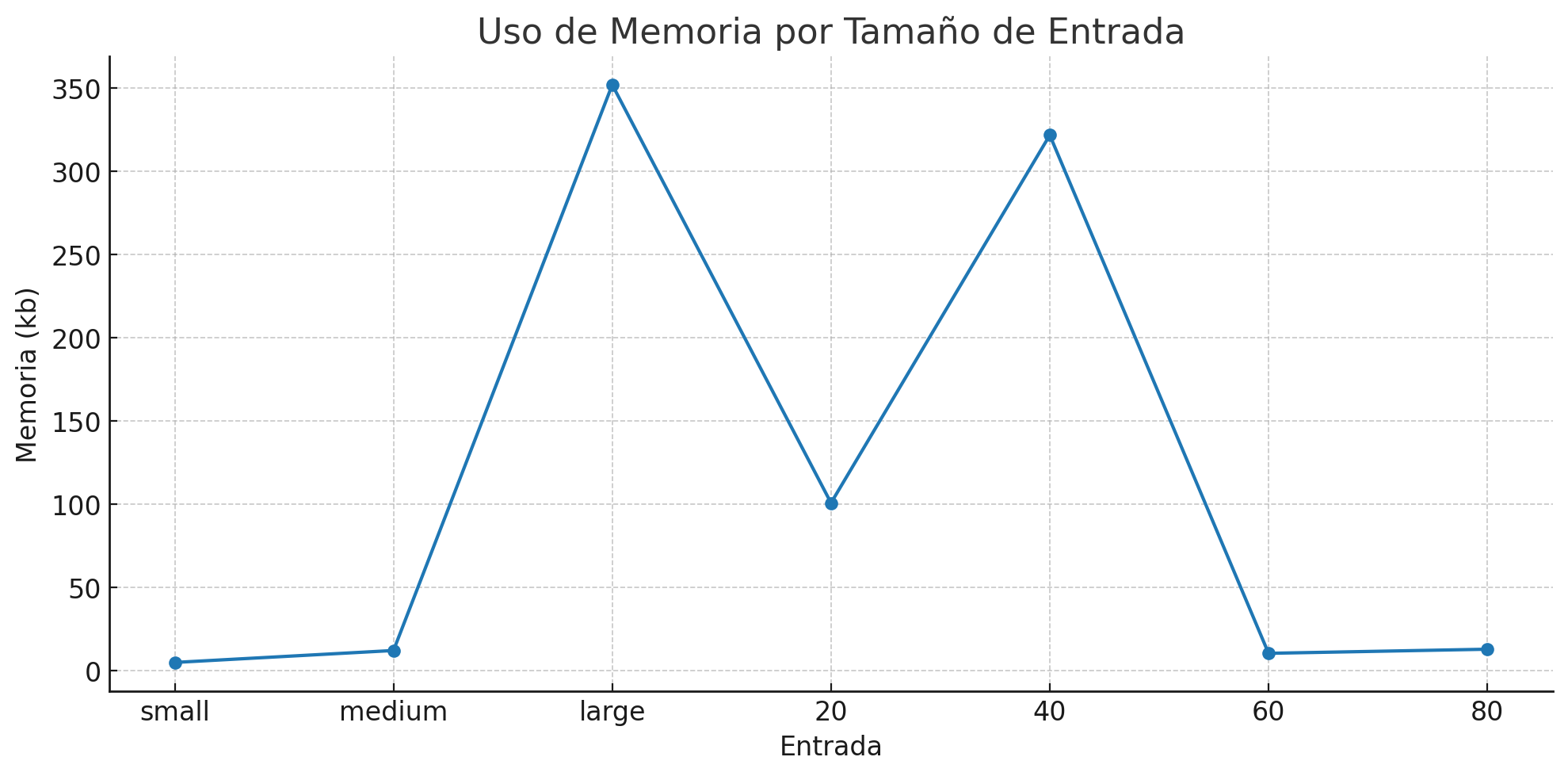
fecha inicial=2023-08-31

fecha final=2023-09-01

|  | **Máquina** |
| --- | --- |
| **Procesadores** | INtel I3 Dual Core |
| **Memoria RAM (GB)** | 6gb |
| **Sistema Operativo** | Widows 10 |

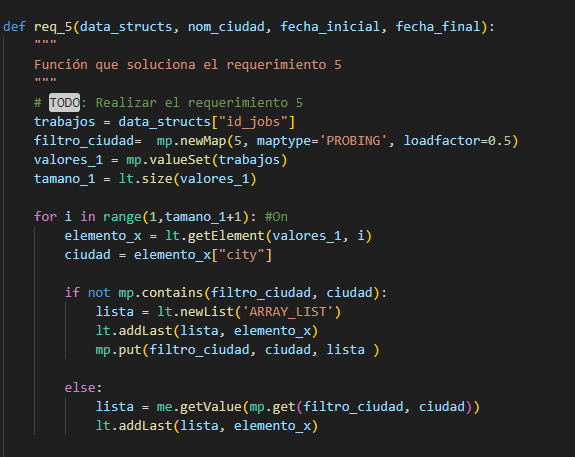
| **Entrada** | **Tiempo (ms)** | **Memoria (kb)** |
| --- | --- | --- |
| 20 | 35786 | 100.8427734375 |
| 40 | 79667 | 321.6487834214 |
| 60 | 10945 | 10.568 |
| 80 | 70811 | 13.0328 |
| small | 72445 | 5.093891 |
| medium | 10789 | 12.260 |
| large | 91234 | 351.87835625 |



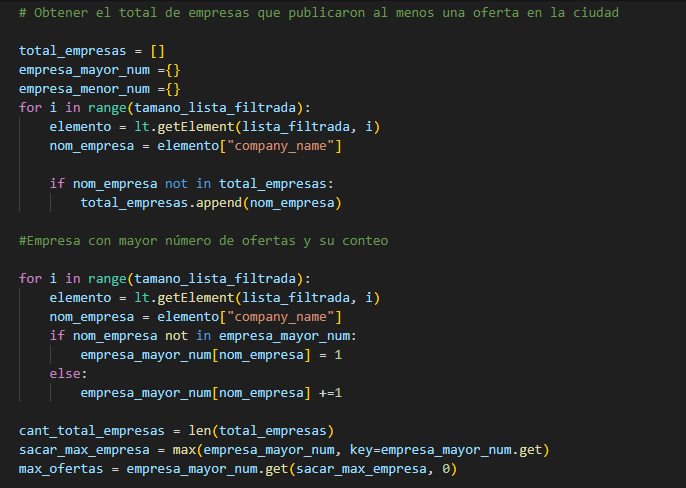


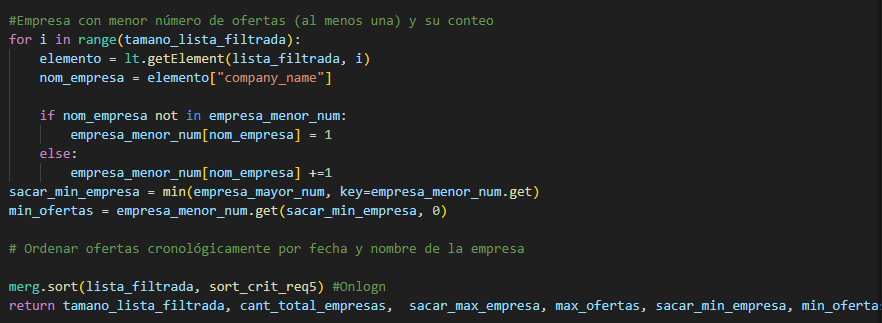
# **Requerimiento 5**

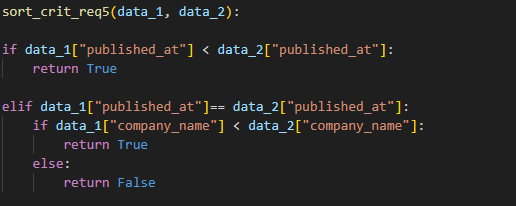
## **Descripción**











El código proporcionado consta de dos funciones principales: `sort\_crit\_req5` y `req\_5`.

La función `sort\_crit\_req5` es una función de comparación utilizada para ordenar elementos. Toma dos diccionarios `data\_1` y `data\_2`, cada uno representando un conjunto de datos con claves "published\_at" y "company\_name". Esta función compara primero las fechas de publicación. Si son iguales, compara los nombres de las empresas. Devuelve `True` si `data\_1` debería colocarse antes que `data\_2` según los criterios establecidos y `False` en caso contrario.

La función `req\_5`, toma una estructura de datos (`data\_structs`), un nombre de ciudad (`nom\_ciudad`), y dos fechas (`fecha\_inicial` y `fecha\_final`). La función filtra los trabajos por ciudad al extraerlas de una tabla de hash y crea un nuevo mapa donde coloca las “ciudades” como llaves para facilitar su búsqueda, y en los valores de estas el resto de información correspondiente a las ofertas que tienen esa ciudad y luego una vez compara la ciudad que ingresa el usuario con la llave del mapa, agrega el valor correspondiente a un arreglo donde hace un filtro con las fechas, y luego realiza varios cálculos sobre estos datos filtrados, como contar el número total de empresas que publican ofertas de trabajo en la ciudad especificada durante el período de tiempo dado, encontrar la empresa con el mayor y menor número de ofertas, y finalmente ordena las ofertas filtradas cronológicamente por fecha y nombre de la empresa.

La función devuelve varios valores: el tamaño de la lista filtrada, el número total de empresas que publicaron ofertas en la ciudad, el nombre de la empresa con el mayor número de ofertas y su cantidad, el nombre de la empresa con el menor número de ofertas y su cantidad, y la lista filtrada de ofertas ordenadas.

| **Entrada** | Estructura de datos, input ciudad, input fecha inicial, input fecha final |
| --- | --- |
| **Salidas** | Tamaño lista, cantidad total empresas, maximo de empresas, minimo de empresas, minimo de ofertas y la lista filtrada |
| **Implementado (Sí/No)** | Si. Implementado por Andrea Aroca |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Creación de la estructura de datos de filtro por ciudad | O(n) |
| Filtrado de trabajos por ciudad | O(n) |
| Filtrado de trabajos por fecha | O(n) |
| Cálculo del total de empresas que publicaron al menos una oferta | O(n) |
| Determinación de la empresa con el mayor número de ofertas | O(n) |
| Determinación de la empresa con el menor número de ofertas | O(n) |
| Ordenamiento cronológico | 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 ID 1.

| Procesadores | Intel Core I7 |
| --- | --- |
| Memoria RAM | 8 GB |
| Sistema Operativo | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
| --- | --- |
| small | 43723.98 |
| 5 pct | 52664.89 |
| 10 pct | 55238.02 |
| 20 pct | 57624.08 |
| 30 pct | 60035.57 |
| 50 pct | 67922.75 |
| 80 pct | 71449.55 |
| large | 88645.01 |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| small | Dato1 | 43723.98 |
| 5 pct | Dato2 | 52664.89 |
| 10 pct | Dato3 | 55238.02 |
| 20 pct | Dato4 | 57624.08 |
| 30 pct | Dato5 | 60035.57 |
| 50 pct | Dato6 | 67922.75 |
| 80 pct | Dato7 | 71449.55 |
| large | Dato8 | 88645.01 |

## **Análisis**

Este análisis sugiere que el algoritmo muestra un comportamiento similar para ambos conjuntos de datos en términos de sensibilidad al porcentaje de completitud, pero los tiempos de ejecución son considerablemente menores en este caso. Esto podría deberse a optimizaciones en el algoritmo o diferencias en las características de los datos que permiten un procesamiento más eficiente. La búsqueda con tablas de hash facilita un análisis detallado y eficiente de los datos proporcionados, lo que permite una mejor comprensión del comportamiento del algoritmo en diferentes situaciones.

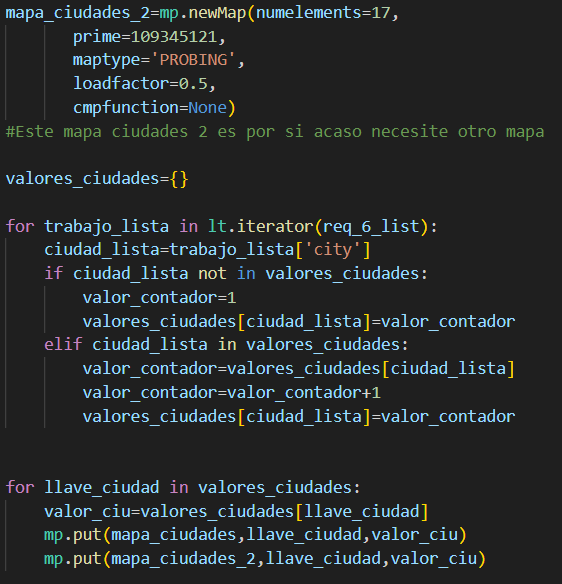
# 

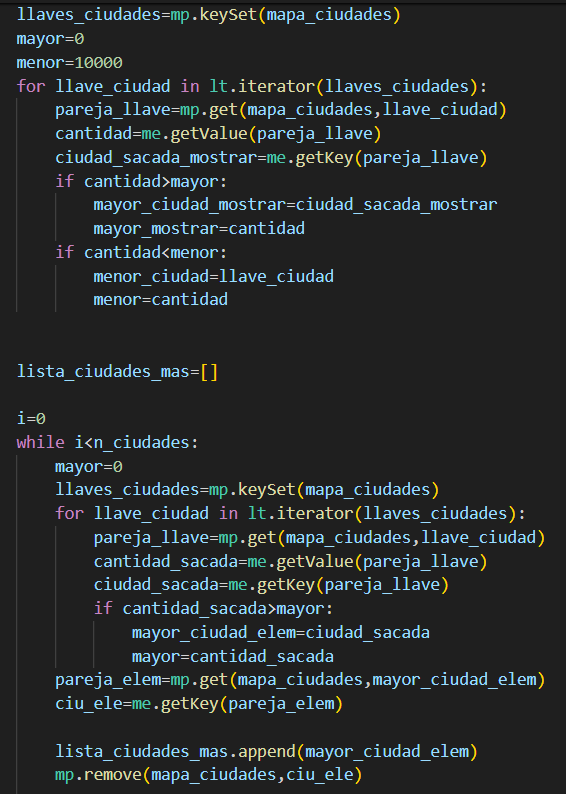
# 

# **Requerimiento 6**

## **Descripción**











Este requerimiento toma, a partir de un año, un N número de ciudades y una experticia (todo seleccionado por el usuario), para filtrar los detalles trabajos que se encuentran en las primeras N ciudades con más ofertas de trabajos dado los requisitos del usuario, en donde contando a partir de un arreglo creado para guardar los trabajos que cumplen con el año y la experticia, se crea un mapa para guardar las ciudades y cuantos trabajos hay en esas ciudades, y finalmente, se guardan y se ordenan las primeras N ciudades con más trabajos que cumplen con los requisitos en otro arreglo distinto. El requerimiento da como resultado una lista de los trabajos en estas N ciudades y que cumplen con el resto de los requisitos de búsqueda. Tambien informa al usuario sobre el total de ciudades que cumplen con estos requisitos de búsqueda, el total de empresas en las n ciudades que cumplen los requisitos, el total de ofertas publicadas que cumplen con los requisitos en las N ciudades y el nombre de la ciudad con mayor y menor ofertas dados los criterios de búsqueda y su respectivo conteo de ofertas.

| **Entrada** | Estructuras de datos del modelo  • Nombre de la ciudad.  • La fecha inicial del periodo a consultar (con formato"%Y-%m-%d").  • La fecha final del periodo a consultar (con formato "%Y-%m-%d"). |
| --- | --- |
| **Salidas** | • El total de ciudades que cumplen con las condiciones de la consulta (valor menor o igual a N)  • El total de empresas que cumplen con las condiciones de la consulta  • El total de ofertas publicadas que cumplen con las condiciones de la consulta  • Nombre de la ciudad con mayor cantidad de ofertas de empleos y su conteo  • Nombre de la ciudad con menor cantidad de ofertas de empleos y su conteo  Listado de trabajos en las N ciudades |
| **Implementado (Sí/No)** | Si. Implementado por Juan David Calderon |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Buscar los trabajos que cumplen con el año de publicación y experticia y agregarlos a un arreglo nuevo | O(n) |
| contar cuantas ofertas de trabajo hay en una ciudad | O(n) |
| almacenar la ciudad como llave y el número de ofertas en la ciudad como valor | O(n) |
| Encontrar la ciudad con más y menos ofertas de trabajo | O(n) |
| sacar de todas las ciudades encontradas las n requeridas de la lista y añadirlas a una lista nativa de Python | O(n2) |
| A partir de las n ciudades encontradas la lista nativa de Python agregarlas un arreglo | O(n) |
| encontrar la cantidad de empresas que ofrecen trabajos dentro de esas n ciudades | O(n) |
| ***TOTAL*** | ***O(n2)*** |

## **Pruebas Realizadas/Tablas de datos**

parametros de prueba:

año=2023

N ciudades=5

experticia= mid

|  | **Máquina** |
| --- | --- |
| **Procesadores** | AMD Ryzen 5 2500U |
| **Memoria RAM (GB)** | 8GB |
| **Sistema Operativo** | Widows 10 |

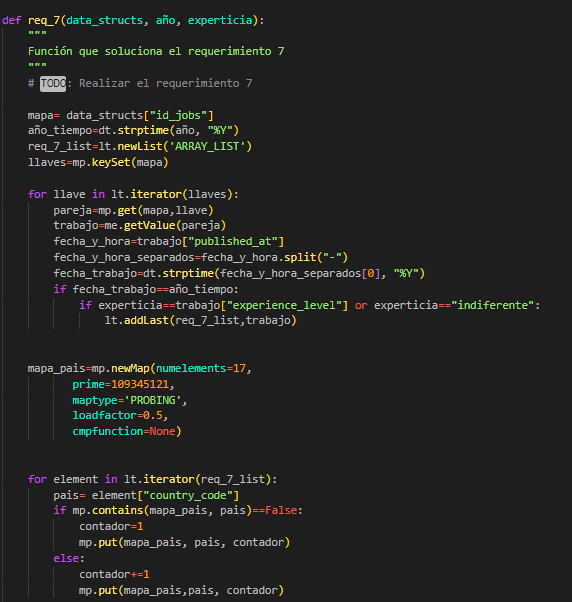
| **Entrada** | **Tiempo (ms)** | **Memoria (kb)** |
| --- | --- | --- |
| 20 | 24253 | 464 |
| 40 |  |  |
| 60 |  |  |
| 80 |  |  |
| small | 44503 | 488 |
| medium |  |  |
| large |  |  |

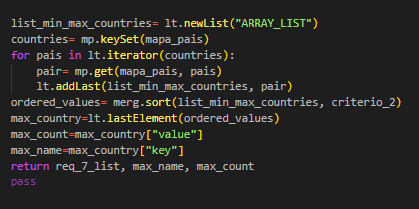
### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Requerimiento 7

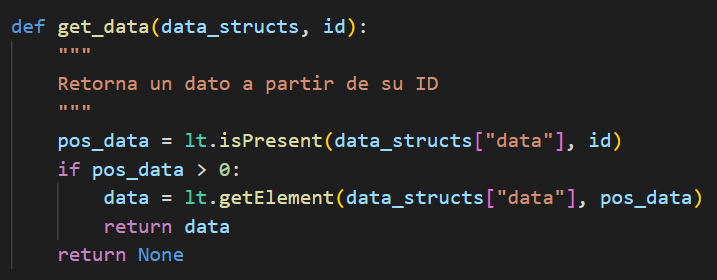




La función req\_7 se encarga de filtrar ofertas de trabajo de acuerdo a un año y nivel de experticia dados. Para ello, recorre las ofertas contenidas en data\_structs, comparando la fecha de publicación y el nivel de experticia con los parámetros proporcionados. Si coinciden, la oferta se agrega a una nueva lista. Además, realiza un conteo de ofertas por país, utilizando un mapa para llevar la cuenta. Finaliza devolviendo la lista de ofertas filtradas junto con el país que tiene el mayor número de ofertas y la cantidad de estas, permitiendo así identificar rápidamente las áreas con mayor demanda en el mercado laboral para el año y experiencia especificados.

# **Requerimiento Ejemplo**

## **Descripción**



Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

| **Entrada** | Estructuras de datos del modelo, ID. |
| --- | --- |
| **Salidas** | El elemento con el ID dado, si no existe se retorna None |
| **Implementado (Sí/No)** | Si. Implementado por Juan Andrés Ariza |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
| --- | --- |
| Buscar si el elemento existe (isPresent) | O(n) |
| Obtener el elemento (getElement) | O(1) |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

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

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

| **Entrada** | **Tiempo (ms)** |
| --- | --- |
| small | 0.05 |
| 5 pct | 0.33 |
| 10 pct | 1.28 |
| 20 pct | 2.54 |
| 30 pct | 4.98 |
| 50 pct | 7.51 |
| 80 pct | 13.81 |
| large | 25.97 |

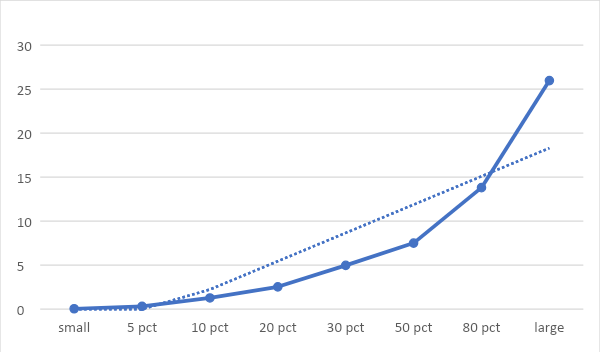
### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
| --- | --- | --- |
| small | Dato1 | 0.05 |
| 5 pct | Dato2 | 0.33 |
| 10 pct | Dato3 | 1.28 |
| 20 pct | Dato4 | 2.54 |
| 30 pct | Dato5 | 4.98 |
| 50 pct | Dato6 | 7.51 |
| 80 pct | Dato7 | 13.81 |
| large | Dato8 | 25.97 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.



## **Análisis**

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.