# ANÁLISIS DEL RETO

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

## Requerimiento <<1>>

### Descripción

Este requerimiento se encarga de retornar la cantidad de ofertas en un rango de fechas. Primero se obtienen los nodos del árbol. Finalmente, se obtienen los elementos en estas listas.

| | |
|---|---|
| **Entrada** | Fecha inicial, fecha final |
| **Salidas** | Ofertas que están en el rango de fechas |
| **Implementado (Sí/No)** | Si. Implementado por Juan Goyeneche y Mariana González. |

### Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Paso 1 (get) | O(log n) |
| Paso 1 (iteración filtrar lista) | O(n) |
| Paso 3 (addFirst) | O(1) |
| ***TOTAL*** | ***O (log n)*** |

### Pruebas Realizadas

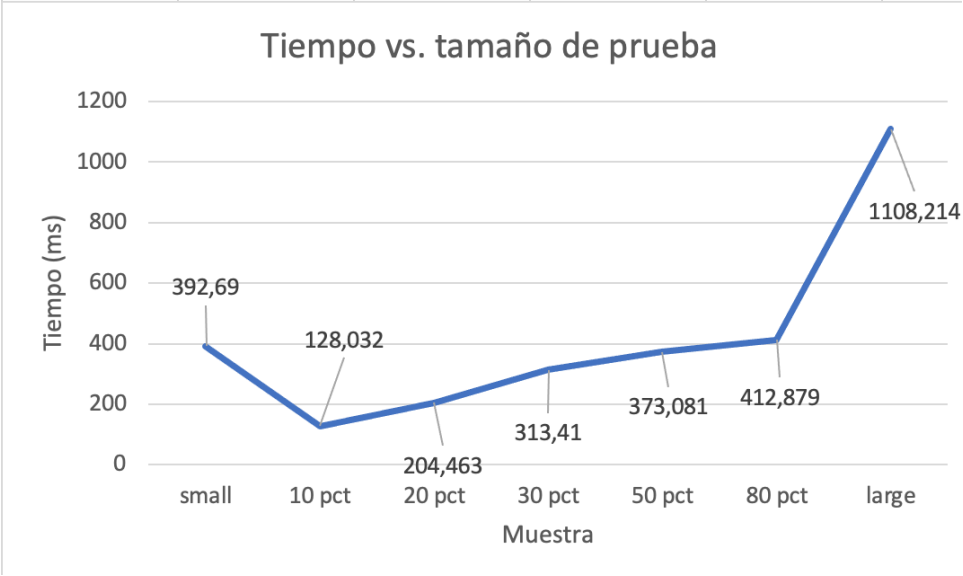
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron la fecha inicial (2022-11-22) y la fecha final (2022-11-22)

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

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 392.690 |
| 10 pct | 128.032 |
| 20 pct | 204.463 |
| 30 pct | 313.410 |
| 50 pct | 373.081 |
| 80 pct | 412.879 |
| large | 1108.214 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Este tiene que obtener los nodos de un árbol (RBT) que estén en un rango de fechas dado, por ello tiene una complejidad de O(k). Después, se itera sobre la lista y se obtienen las ofertas en las fechas dadas y debido a el orden del mapa no se hace un sort para las fechas. Finalmente las ofertas se agregan a una lista y se imprimen.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(log n) ya que a medida que crece la muestra, el tiempo crece de manera proporcional (log n), manteniendo una relación con el aumento del tiempo y el aumentode los datos

# Requerimiento <<3>>

## Descripción

Este requerimiento se encarga de retornar el número de ofertas que hay en un país por su nivel de xp y la información de estas ofertas. Primero se obtiene el nodo con el código del país. Después, se obtiene la lista del nivel de experiencia deseado o en caso de que el input de experiencia sea indiferente se obtienen las listas de todos los niveles de xp. Finalmente, se itera sobre la lista o listas y se ordenan cronológicamente. Al final se retornan las ofertas deseadas y su cantidad.

| | |
|---|---|
| **Entrada** | Numero de ofertas, país, nivel de xp |
| **Salidas** | Total de ofertas ofrecidas en el pais para el nivel de xp, las N ofertas y su respectiva Info |
| **Implementado (Sí/No)** | Si, Mariana Gonzalez |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener el elemento (get) | O(log n) |
| (Iteración en lista) | O(n) |
| Obtener el tamaño (size) | O(1) |
| Addlast (addLast) | O(1) |
| ***Hacer el sort*** | O(n log n) (promedio) |
| ***TOTAL*** | ***O(n log n)*** |

## Pruebas Realizadas

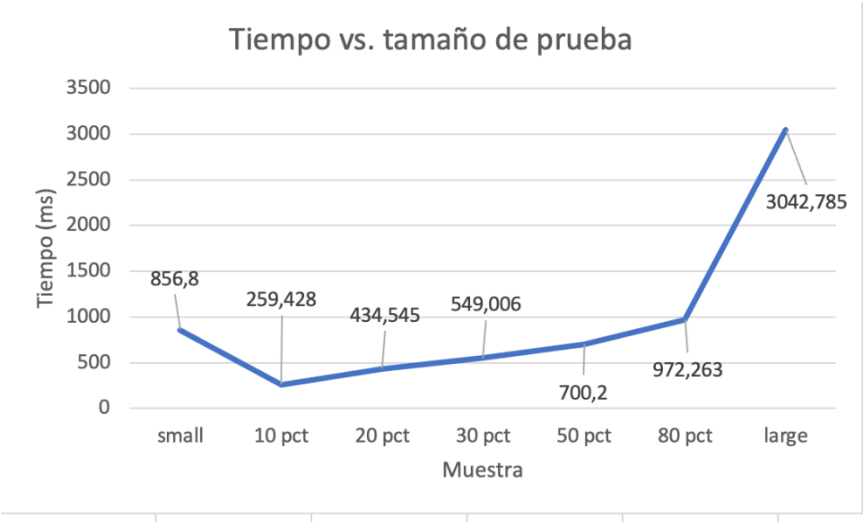
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada 5, US, indiferente.

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

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 856.800 |
| 10 pct | 259.428 |
| 20 pct | 434.545 |
| 30 pct | 549.066 |
| 50 pct | 700.200 |
| 80 pct | 972.263 |
| large | 3042.785 |

## Gráficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Este requerimiento tiene que obtener un nodo del árbol que tenga el pais y el nivel de experiencia deseado para después hacer un sort, por ello tiene una complejidad de O(n log n). Primero, este tiene que obtener el nodo de un árbol (RBT) que sea el país dado, por ello tiene una complejidad de O(log n). Esto debido a lo que se hace después de obtener los trabajos en ese país, es buscar de todos los elementos de esa lista que tenga el mismo nivel de xp, después agregarlos a una lista y finalmente, organizar esa lista y agregar a una sublista. Debido a esto, en el peor de los casos es necesario recorrer toda la lista, seguido a esto agregar a una nueva lista y finalmente, organizar esta lista, agregar a una sublista y devolver el resultado.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que no tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n), debido tal vez, a que la lista no se encuentra tan desordenada, debido al orden que tiene el árbol.

# Requerimiento <<4>>

## Descripción

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

| | |
|---|---|
| **Entrada** | Numero de ofertas, ciudad, tipo de trabajo (remoto, oficina, parcialmente remoto) |
| **Salidas** | Total de ofertas ofrecidas en la ciudad para la ubicación deseada, las N ofertas y su respectiva Info |
| **Implementado (Sí/No)** | Si, Juan Goyeneche |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| Pasos | Complejidad |
|---|---|
| Obtener el elemento (get) | O(log n) |
| (Iteración en lista) | O(n) |
| Obtener el tamaño (size) | O(1) |
| Addlast (addLast) | O(1) |
| ***Hacer el sort*** | O(n log n) (promedio) |
| ***TOTAL*** | ***O(n log n)*** |

## Pruebas Realizadas

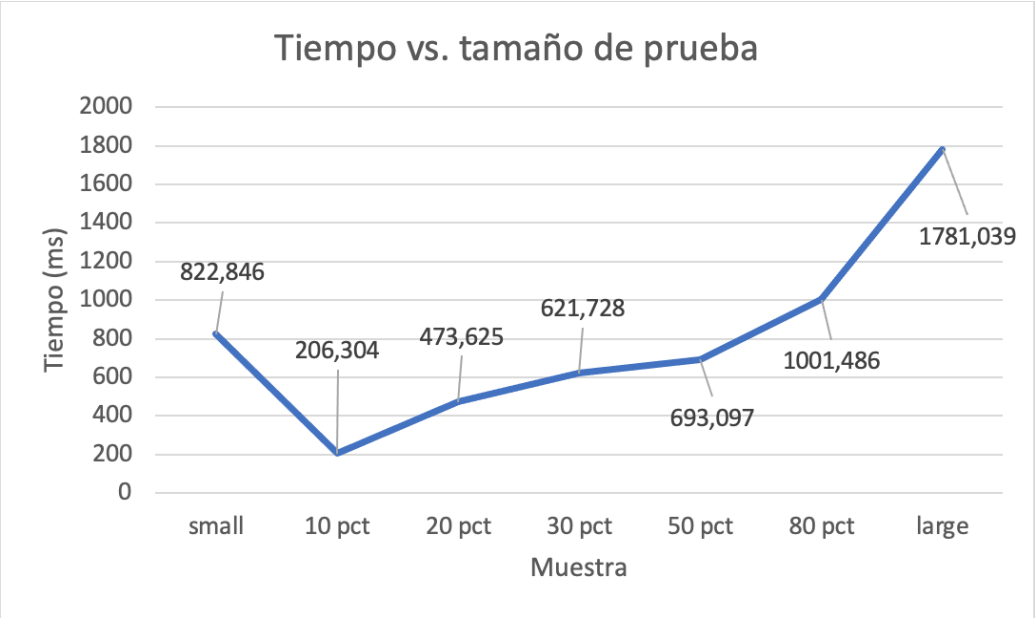
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada 5, remote, berlin.

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

| Muestra | Tiempo (ms) |
|---|---|
| small | 822.846 |
| 10 pct | 206.304 |
| 20 pct | 473.625 |
| 30 pct | 621.728 |
| 50 pct | 693.097 |
| 80 pct | 1001.486 |
| large | 1781.039 |

## Gráficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Este requerimiento tiene que obtener un nodo del árbol que tenga la ciudad y la ubicación deseada para después hacer un sort, por ello tiene una complejidad de O(n log n). Primero, este tiene que obtener el nodo de un árbol (RBT) que sea la ciudad dada, por ello tiene una complejidad de O(log n). Esto debido a lo que se hace después de obtener los trabajos en esa ciudad, es buscar de todos los elementos de esa lista que tenga la misma ubicación, después agregarlos a una lista y finalmente, organizar esa lista y agregar a una sublista. Debido a esto, en el peor de los casos es necesario recorrer toda la lista, seguido a esto agregar a una nueva lista y finalmente, organizar esta lista, agregar a una sublista y devolver el resultado.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n log n), manteniendo una relación con el aumento del tiempo y el aumento de los datos. A diferencia del anterior requerimiento, esta si mantiene una relación con (n log n) puede ser debido a que el orden es por fecha y al obtener un elemento del árbol por una ubicación especifica el orden se ve afectado, por otro lado, en el req anterior no usábamos una xp especifica y por ello todos los elementos del nodo deseado eran validos bajando así la complejidad.

# Requerimiento <<6>>

Este requerimiento se encarga de clasificar las N ciudades con mayor cantidad de ofertas de trabajo en un país entre un rango de fechas y nivel de experticia de la oferta. Lo primero que hace es encontrar los elementos que cumplen con las condiciones, después busca los elementos en la otra estructura (infos) que tienen el mismo id de los que cumplían con las condiciones y seguido a esto, en la otra función, crea nuevos diccionarios con los datos deseados. De no existir, retorna Ningún resultado encontrado y termina el programa.

## Descripción

Breve descripción de como abordaron la implementación del requerimiento

| | |
|---|---|
| **Entrada** | Rango de fechas, rango de salario y numero de ciudades |
| **Salidas** | Numero de ofertas entre los rangos dados, numero de ciudades que tienen ofertas en los rangos dados, Las N ciudades ordenadas alfabéticamente y para la ciudad con mayores ofertas en los rangos, una lista con cada oferta como elemento. |
| **Implementado (Sí/No)** | Si. Implementado por Juan Goyeneche y Mariana Gonzalez |

## Análisis de complejidad

| **Pasos** | **Complejidad** |
|---|---|
| Obtener el elemento (get) | O(log n) |
| Obtener el tamaño (size) | O(1) |
| Iterar por las listas obtenidas | O(n) |
| Addlast (addLast) | O(1) |
| Iterar por la nueva lista | O(n) |
| Crear los diccionarios | O(n) |
| addLast | O(1) |
| ***Hacer el sort*** | O(n log n) (promedio) |
| ***TOTAL*** | ***O(n log n)*** |

## Pruebas Realizadas

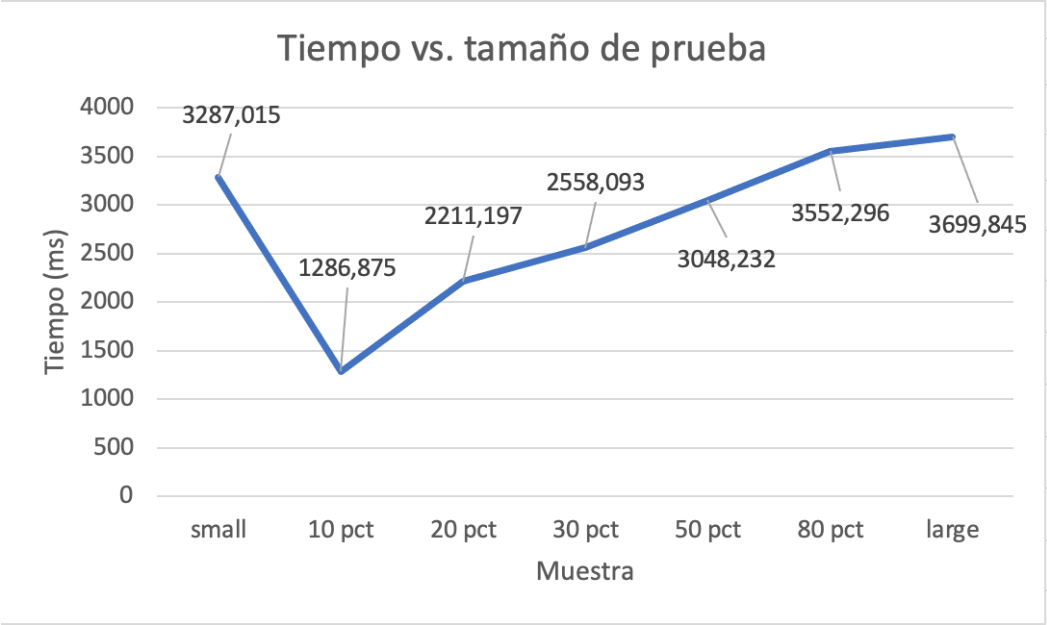
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el 5, 2022-11-22, 2023-01-01, 100, 5000.

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

| Muestra | Tiempo (ms) |
|---|---|
| small | 3287.015 |
| 10 pct | 1286.875 |
| 20 pct | 2211.197 |
| 30 pct | 2558.093 |
| 50 pct | 3084.232 |
| 80 pct | 3552.296 |
| large | 3699.845 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Este requerimiento tiene que obtener unas fechas elemento de una árbol y a su ves, un rango de salarios de otro árbol. Seguido a esto, agrega los elementos que cumpla con las condiciones a una lista y los elementos de estas son comparados por ID para obtener los elementos correspondientes y crear así la lista final por ciudad. Finalmente, al tener la lista final se hace un sort, por ello tiene una complejidad de O(n log n). Con respecto a la estructura de datos, se hace uso de 2 estructuras, Una es un árbol en los cuales sus nodos son mapas que tienen como llave una fecha y como valor una lista de ofertas en esa fecha. La segunda estructura es un árbol en los cuales sus nodos son el promedio de salarios de una oferta y esta tiene como valor una lista de ofertas que tienen como promedio ese salario.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n log n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n log n), manteniendo una relación con el aumento del tiempo y el aumento de los datos

# Requerimiento <<7>>

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

## Descripción

Este requerimiento se encarga de mostrar la gráfica de la propiedad elegida del país y el año ingresados por parámetro. Lo primero que se hace es filtrar el mapa y el arbol RBT para obtener una lista con las ofertas validas. Se hace un if por cada una de las tres propiedades, para filtrar la ciudad, el nivel de experiencia o la habilidad. Se crea un diccionario en el cual las llaves son los datos de la propiedad y los valores son el número de veces que se repiten en la lista de ofertas validas.

| | |
|---|---|
| **Entrada** | Año relevante, código del país, propiedad de conteo (experticia, ubicación, habilidad) |
| **Salidas** | Número de ofertas publicadas durante ese año, número de ofertas utilizadas para hacer el gráfico, valor mínimo y máximo de la gráfica, gráfica, listado de ofertas laborales y su información requerida. |
| **Implementado (Sí/No)** | Si. Implementado por Juan Goyeneche y Mariana González |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| mp.get() | O(1) |
| me.getValue() | O(1) |
| om.get() | O(log n) |
| nodoRBT.getValue() | O(log n) |
| For experiencia | O(n) |
| For filtro ciudades | O(n) |
| For diccionario | O(n) |
| For lista ides | O(n) |
| me.get() skills | O(1) |
| me.getValue() skills | O(1) |
| ***TOTAL*** | ***O(log n)*** |

## Pruebas Realizadas

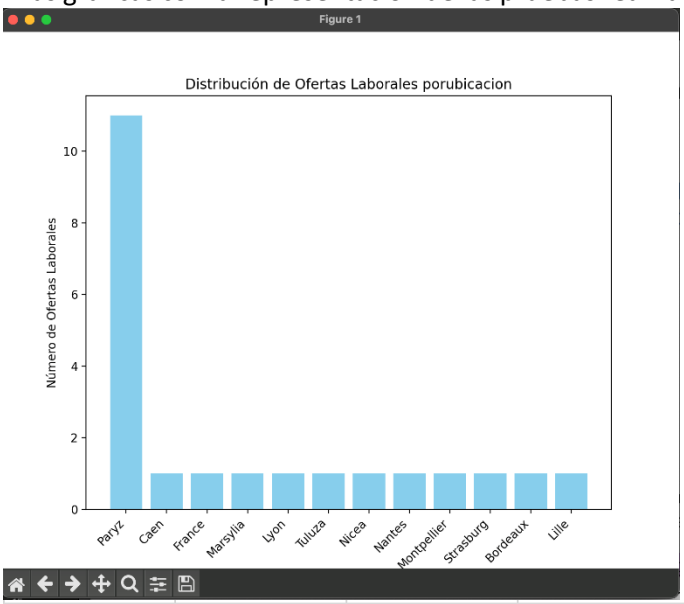
Las pruebas realizadas se realizaron en una maquina con estas especificaciones. Los datos de entrada fueron FR, 2023, ubicación

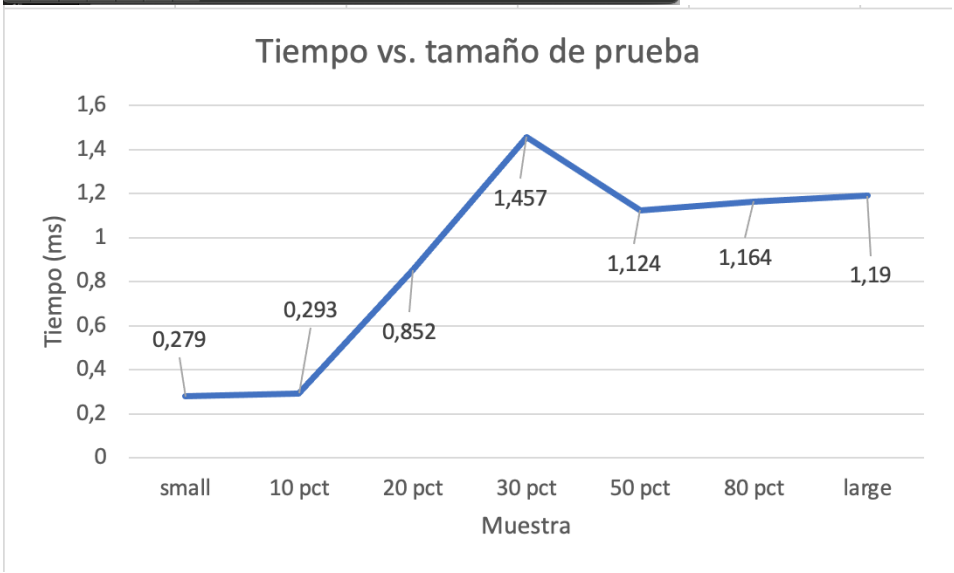
| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | MacOS Sonoma |

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 0.279 |
| 10 pct | 0.293 |
| 20 pct | 0.852 |
| 30 pct | 1.457 |
| 50 pct | 1.124 |
| 80 pct | 1.164 |
| large | 1.190 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

La estructura de datos para este requerimiento es un mapa que tiene como llaves los países y valor un árbol RBT, el cual a su vez tiene como llave los años y como valor un ARRAY_LIST que contiene las ofertas de trabajo que son válidas según los parámetros dados. Primero se realizan los gets necesarios con complejidad O(1) y O(logn) para obtener la lista de la cual se basará la gráfica.

Cuando la propiedad deseada es experticia se crea el diccionario con cada una de las llaves (‘senior’, ‘mid’, ‘junior’) y se realiza un for para que según la experticia de cada oferta se agregue uno al valor de la llave necesaria.

Cuando la propiedad requerida es ubicación, se realiza un for para tener una lista sin las ubicaciones repetidas y por lo tanto no estén repetidas en el diccionario, se realiza otro for en el cual se crea el diccionario y por cada oferta de esa ciudad se agrega uno al valor.

Finalmente, cuando la propiedad deseada es habilidad, se filtra la lista para que los ides no estén repetidos; una vez con esta lista se genera un for para obtener la información de cada id del archivo skills , haciendo un get() y un getVlue(), ambos con complejidades O(1). Después se crea el diccionario y por cada oferta con ese ID se genera una llave y se le agrega uno a su valor.

Por último, se utiliza la librería matplotLib para generar los gráficos.

La complejidad final del requerimiento es O(log n) debido a que los get() de tablas de hash son

O(1), mientras que los get() y getValue() de los arboles RBT son O(log n) siendo los de mayor complejidad.

# Requerimiento <<8>>

Este requerimiento se encarga de clasificar las N ciudades con mayor cantidad de ofertas de trabajo en un país entre un rango de fechas y nivel de experticia de la oferta. Lo primero que hace es encontrar los elementos que cumplen con las condiciones, después busca los elementos en la otra estructura (infos) que tienen el mismo id de los que cumplían con las condiciones y seguido a esto, en la otra función, crea nuevos diccionarios con los datos deseados. De no existir, retorna Ningún resultado encontrado y termina el programa.

## Descripción

Breve descripción de como abordaron la implementación del requerimiento

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, numero de ciudades, codigo del pais, nivel de experiencia, fecha inicial y fecha final. |
| **Salidas** | Una nueva tad lista organizada y cada elemento de la tad lista contiene el nombre de la ciudad, total ofertas de ciudad, promedio_salario_ciudad, n_empresas_publicaron_una_oferta_ciudad, n_empresa_mayor_ofertas, conteo, mejor_oferta, peor_oferta |
| **Implementado (Sí/No)** | Si. Implementado por Juan Goyeneche y Mariana Gonzalez |

## Análisis de complejidad

| **Pasos** | **Complejidad** |
|---|---|
| Iterar sobre la lista proveida (depende del req) | 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 3, ., junior, 2022-11-11, 2023-06-06.

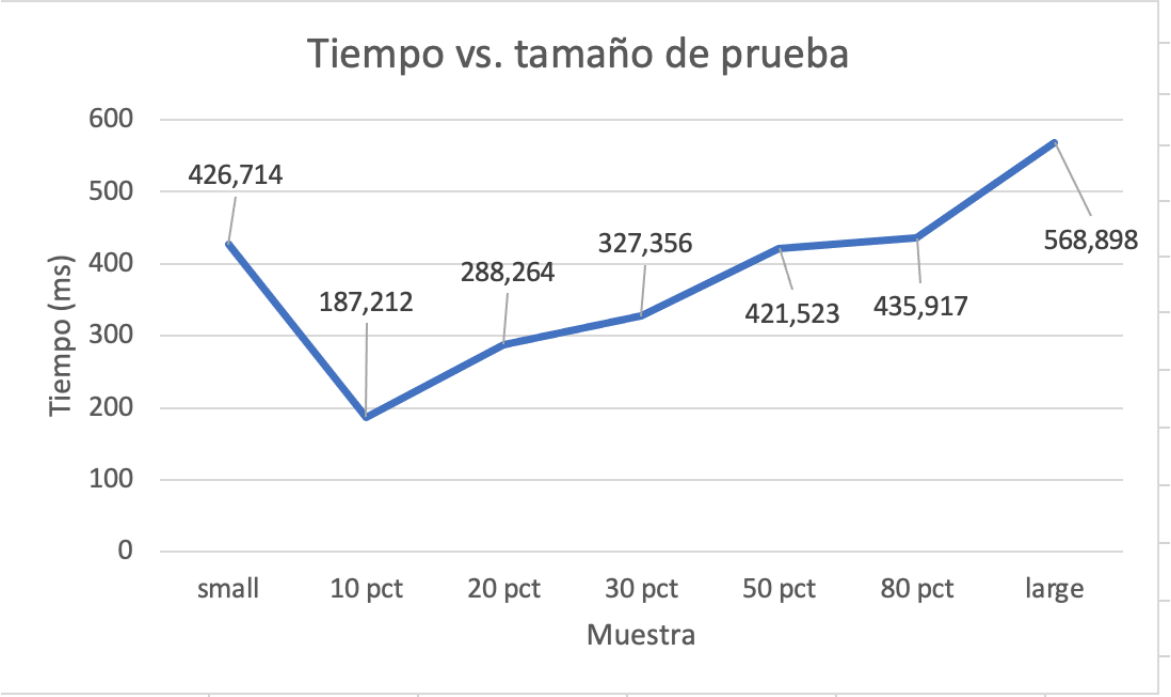
| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 10 |

| Muestra | Tiempo (ms) |
|---|---|
| small | 426.714 |
| 10 pct | 187.212 |
| 20 pct | 288.264 |
| 30 pct | 327.356 |
| 50 pct | 421.523 |
| 80 pct | 435.917 |
| large | 568.898 |

*El análisis del tiempo se hizo con el req 3*

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Este requerimiento tiene que obtener los elementos de un array_list y mostrar su ubicación en el mapa, por ello tiene una complejidad de O(n). Esto se debe a que se itera sobre la lista propuesta y después con folium se muestran las ofertas en el mapa.

Viendo el comportamiento de la gráfica, podemos darnos cuenta de que si tiene un compartimiento similar a O(n) ya que a medida que crece la muestra, el tiempo crece de manera similar (n), manteniendo una relación con el aumento del tiempo y el aumento de los datos. (esto ya que la prueba fue hecha para la misma función con los mismos parámetros)

# Carga de datos:

Este requerimiento se encarga de clasificar las N ciudades con mayor cantidad de ofertas de trabajo en un país entre un rango de fechas y nivel de experticia de la oferta. Lo primero que hace es encontrar los elementos que cumplen con las condiciones, después busca los elementos en la otra estructura (infos) que tienen el mismo id de los que cumplían con las condiciones y seguido a esto, en la otra función, crea nuevos diccionarios con los datos deseados. De no existir, retorna Ningún resultado encontrado y termina el programa.

## Descripción

Breve descripción de como abordaron la implementación del requerimiento

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, numero de ciudades, codigo del pais, nivel de experiencia, fecha inicial y fecha final. |
| **Salidas** | Una nueva tad lista organizada y cada elemento de la tad lista contiene el nombre de la ciudad, total ofertas de ciudad, promedio_salario_ciudad, n_empresas_publicaron_una_oferta_ciudad, n_empresa_mayor_ofertas, conteo, mejor_oferta, peor_oferta |
| **Implementado (Sí/No)** | Si. Implementado por Juan Goyeneche y Mariana Gonzalez |

## Análisis de complejidad

| **Pasos** | **Complejidad** |
|---|---|
| Buscar si el elemento existe (isEmpty) | O(1) |
| Obtener el elemento (getElement) | O(n) |
| Obtener el tamaño (size) | O(1) |
| (Conversión formato de fecha) | O(n) |
| (Iteración filtro de fecha y ciudad) | O(n) |
| Addlast (addLast) | O(n) |
| Iterar por la nueva lista | O(n) |
| Crear los diccionarios | O(n) |
| addLast | O(n) |
| ***Hacer el sort*** | O(n log n) (promedio) |
| ***TOTAL*** | ***O(n log n)*** |

## Pruebas Realizadas

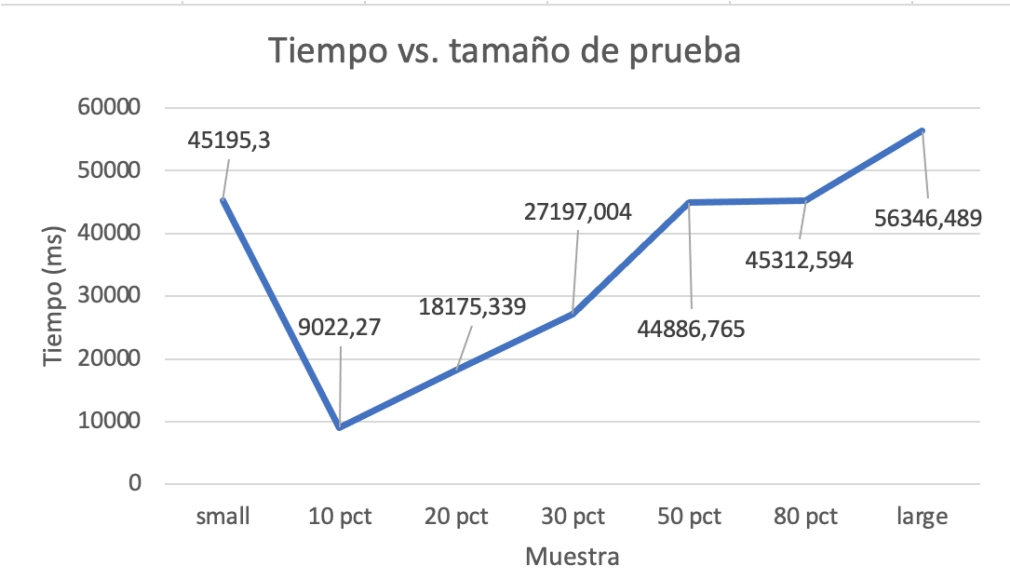
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el 3, ., junior, 2022-11-11, 2023-06-06.

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

| Muestra | Tiempo (ms) |
|---|---|
| small | 45195.300 |
| 10 pct | 9022.270 |
| 20 pct | 18175.339 |
| 30 pct | 27197.004 |
| 50 pct | 44886.765 |
| 80 pct | 45312.594 |
| large | 56346.489 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Ahora bien, para el requerimiento 7 se buscaba tener una complejidad cercana a O(1), por esto para la carga de datos se creó un mapa (tabla de hash) que tenía como llaves cada uno de los países y como valor un mapa RBT de cada país; para el cual las llaves eran los años en los cuales el país tenía ofertas y el valor era una lista con todas las ofertas que cumplían ambos filtros (país y año) (ver representación gráfica en la imagen 1). Se puede decir que la complejidad de la carga de datos del requerimiento 7 es O(logn) ya que los get () y getValue() de los árboles RBT tienen esta complejidad en caso promedio sabiendo que paraje llave-valor se quiere extraer del árbol.

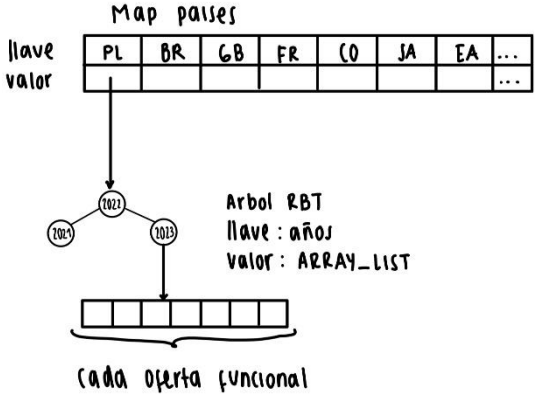
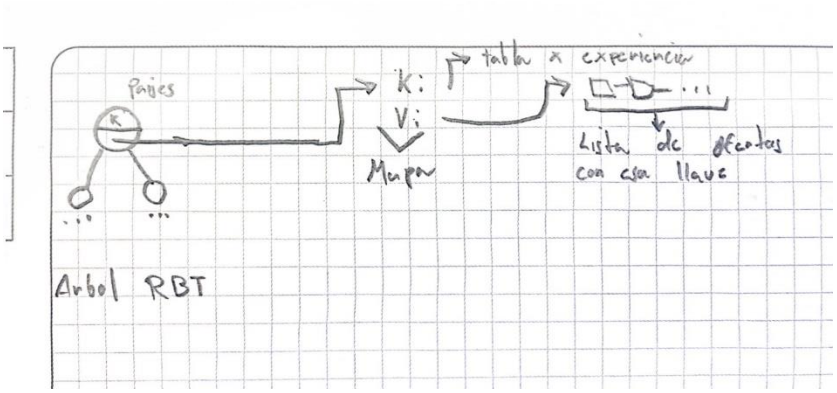



*Imagen 1*: Representación gráfica de la estructura de datos usada para el requerimiento 7

*Imagen 2*:




Representación gráfica de la estructura de datos usada para el requerimiento 3

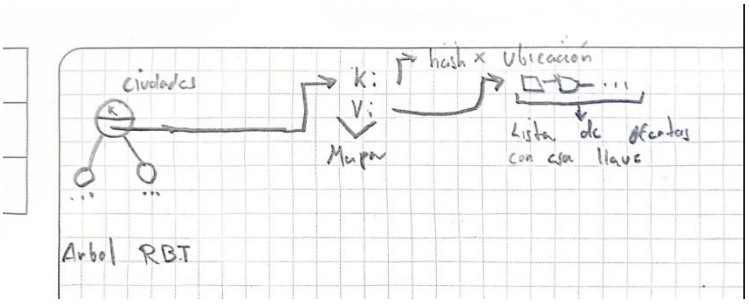
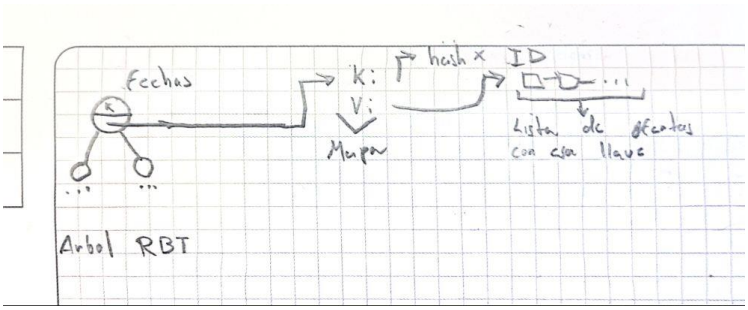



*Imagen 3*: Representación gráfica de la estructura de datos usada para el requerimiento 4



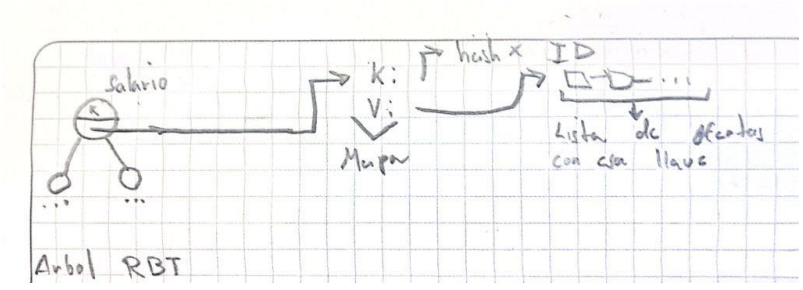




*Imagen 4 y 5*: Representación gráfica de la estructura de datos usada para el requerimiento 6. (la imagen 4 se usó también para el requerimiento 1)