Task It

Adrian Ramirez, Mateo Bustos, Santiago Avila

No. grupo del curso: {*2*}

No. de Equipo de Trabajo: {*5*}

I. INTRODUCCIÓN

En este documento, presentaremos una descripción detallada de cómo nuestro equipo implementó en el diseño previamente desarrollado distintas estructuras no lineales, tales como colas prioritarias y AVL`s para mejorar el proyecto, logrando que este sea más eficiente y completo.

En la entrega previa del proyecto se implementaron diferentes funcionalidades por medio de estructuras unidimensionales, siendo este nuestro punto de partida. En esta entrega nos enfocamos en la introducción de árboles y sus usos, pues su implementación puede traer ventajas y/o desventajas gracias a las características que estas ofrecen para algunas funcionalidades del programa.

A lo largo de este documento, explicaremos en detalle el proceso de implementación de los árboles y las colas prioritarias, así como los beneficios que aportaron a nuestra solución. También se presentarán comparaciones con la implementación de estructuras unidimensionales realizada anteriormente, destacando las mejoras obtenidas en términos de eficiencia. Por último, se espera que el progreso resultante mejore de manera significativa el análisis asintótico, de acuerdo al resultado teórico de notación Big O de complejidad.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

Tenemos claro que la repartición de tareas en un equipo de trabajo es de las cosas que más problemas puede traer, ya sea por falta de comunicación asertiva, por inequidad en la repartición de las mismas, o muchas otros factores que pueden afectar de cierta manera dicha repartición, y que al ser tan determinante en la realización exitosa de un proyecto pueda llevar a malos resultados si no se tiene sumo cuidado con este tema. Es por esta razón que con este proyecto buscamos que esta asignación de tareas deje de depender de ese tipo de factores y se haga de una manera efectiva que permita aumentar la eficiencia de cualquier equipo de trabajo.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

Los usuarios esperados de nuestro producto de software serían cualquier grupo de trabajo que necesite clasificar y repartir diferentes tareas para una mayor eficiencia, ya sean grupos de trabajo colegiales, universitarios, laborales, entre otros.

Habría un único rol, el de integrante del grupo, pues no habrían privilegios de acceso ni de seguridad, ya que todos podrían en todo momento visualizar, agregar y finalizar las tareas que ellos vean convenientes.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

- ***Ingresar nueva tarea***

- ***Descripción:***

En esta sección el integrante del grupo podrá ingresar una nueva tarea a la cola prioritaria de tareas pendientes. En esta función el integrante podrá definir a que subdivisión específica será encolada la tarea.

- ***Acciones iniciadoras y comportamiento esperado*:**

El integrante deberá suministrar una descripción de la tarea, la prioridad que tiene la misma y especificar la subdivisión a la que está pertenecera. Si la descripción es válida, el programa le asignará un ID único a la tarea y la guardará max heap de tareas sin asignar relacionada con la subdivisión escogida (cada subdivisión cuenta con su max heap única de tareas) . En caso de que el integrante no proporcione una descripción, el programa no permitirá el ingreso de la tarea.

- ***Requerimientos funcionales*:**

    - Se podrán ver todas las subdivisiones registradas actualmente.
    - El usuario podrá describir la tarea de la manera en que lo crea conveniente. En caso de que la descripción de la tarea esté vacía, no se permitirá el ingreso de la tarea.
    - El usuario podrá elegir libremente la subdivisión en la que está interesado.
    - Todas las tareas válidas que se ingresen tendrán asignada una identificación única además de su propia prioridad.

- ***Notificar tarea finalizada***

- ***Descripción:***

Lo que hace la funcionalidad “Notificar tarea finalizada” es que poniendo la ID de la tarea que se realizó, reporte la tarea como finalizada y lo lleve al inicio.

- ***Acciones iniciadas y comportamiento esperado*:**

Se espera que el usuario digite el ID de la tarea que realizó y que el programa haga comprobación, primero de que lo que ingresó el usuario sea un número, y luego que verifique si la presunta ID que ingresó el usuario exista en los archivos guardados, y si es así de a la tarea como finalizada.

- ***Requerimientos funcionales:***

Para que el usuario pueda desempeñar correctamente la tarea:

- El software debe tener las comprobaciones anteriormente mencionadas, para que el programa no se quiebre luego de recibir un ID incorrecto o inexistente.
- Ser capaz de buscar en la lista doblemente enlazada en la que se guardan las subdivisiones, la tarea, y que cuando la encuentre, la saque del árbol AVL "Tareas en progreso" y lo pase a la pila de "Tareas finalizadas".
- Debe habilitar correctamente el resto de funcionalidades para el correcto funcionamiento del programa.

- **Solicitar tarea**

- ***Descripción:***

Cuando el usuario requiera una nueva tarea, podrá pedirla en esta sección, únicamente tendrá que seleccionar la subdivisión deseada y se le otorgará la descripción y la ID de su nueva tarea.

- ***Acciones iniciadoras y comportamiento esperado*:**

Se espera que el usuario seleccione la subdivisión deseada en un combobox y presione el botón de "Pedir Tarea", luego el programa buscará en los archivos guardados la subdivisión en la cola prioritaria "Tareas" y le suministre al usuario la descripción y la ID de la más antigua de ellas, y que haría un extract max a la cola prioritaria (max heap).

- ***Requerimientos funcionales*:**

Los requerimientos asociados a esta funcionalidad son:

- El programa debe verificar que existan tareas sin asignar en la subdivisión deseada para posteriormente notificar al usuario.
- Debe otorgar correctamente las subdivisiones que hay actualmente en los archivos guardados.

- ***Agregar nueva subdivisión***

- ***Descripción:***

El programa permitirá crear nuevas subdivisiones al interior del grupo donde cada una contará con sus respectivas listas, árboles y colas prioritarias de tareas (Sin asignar, en proceso, finalizado).

- ***Acciones iniciadoras y comportamiento esperado*:**

Secuencia de acciones del usuario y respuestas esperadas del programa para esta funcionalidad.

En esta función el usuario únicamente escribirá el nombre de la subdivisión que quiere crear. Si la descripción no está vacía, el programa procederá a instanciar una nueva subdivisión con sus diferentes listas,árboles y colas de tareas mencionadas anteriormente.

- ***Requerimientos funcionales*:**

- Permitir al usuario ingresar el nombre de la subdivisión y hacer la verificación de que dicho nombre no esté vacío.
- Si el nombre es valido, se instancian una nueva subdivisión la cual contará contará con los siguientes atributos:
  - Nombre de la subdivisión.
  - Cola prioritaria para guardar tareas sin asignar.
  - Árbol AVL para guardar tareas en progreso.
  - Pila para guardar tareas finalizadas.
- La subdivisión se guardará en árbol AVL junto con el resto de subdivisiones.

- ***Visualizar estado de tareas***

- ***Descripción:***

Permite visualizar en tablas los estados de cada una de las tareas de una subdivisión determinada.

- ***Acciones iniciadoras y comportamiento esperado*:**

El usuario podrá escoger cualquiera de las subdivisiones que se encuentren registradas en el programa actualmente y, al presionar el botón "consultar" la información de las tareas será cargada.

- ***Requerimientos funcionales*:**

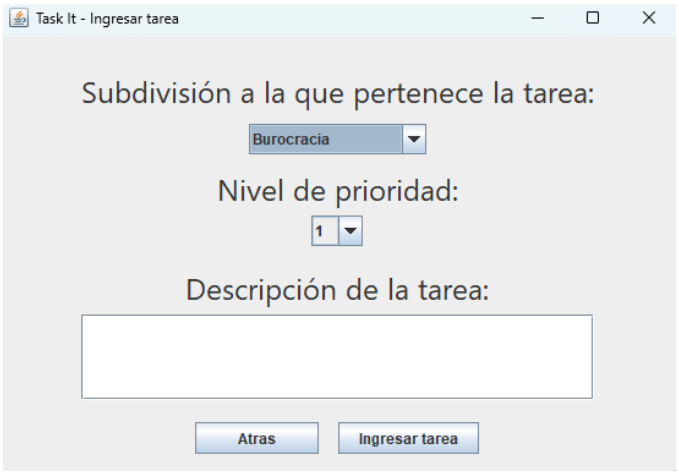
- Se proveerá una lista con todas las subdivisiones que se encuentren registradas al momento de usar la función.
- Al momento de confirmar la subdivisión en la que se está interesado por medio del botón "consultar".
- Se mostrarán 3 tablas donde en cada una se guardarán las tareas asociadas con cada uno de los estados definidos en el programa (Sin asignar, En proceso, Finalizado).

V. AVANCE EN LA IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

- **Pestaña-Inicio**

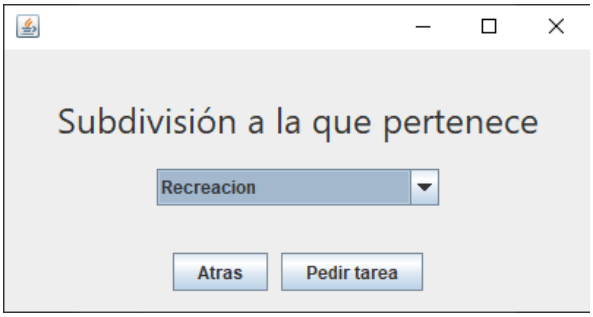
Pestaña principal del programa, la cual da acceso a las diferentes funcionalidades del mismo.

- **Pestaña-Ingresar nueva tarea**



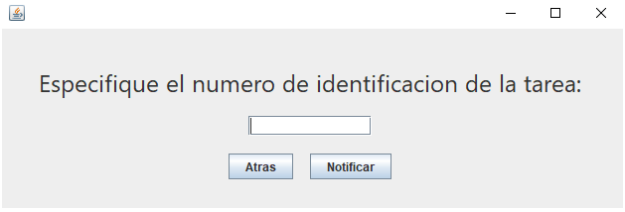
Pestaña que permite al usuario agregar una nueva tarea, almacenando y asignando a su respectiva subdivisión teniendo en cuenta el nivel de prioridad especificado por el usuario..

- **Pestaña-Solicitar Tarea**



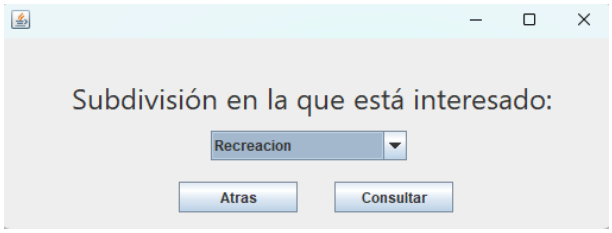
Pestaña en que el usuario puede solicitar que se le asigne una tarea, de acuerdo a la subdivisión a la que se pertenezca y al tiempo que esta tarea lleva agregada y pendiente.

- **Pestaña-Notificar tarea finalizada**



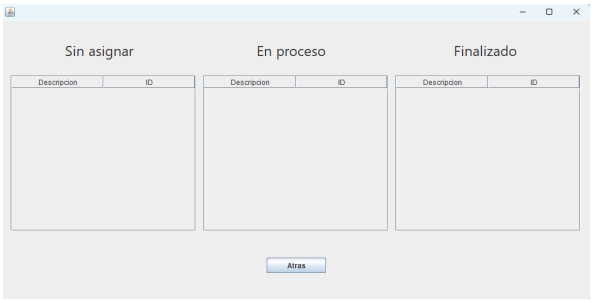
Esta pestaña permite notificar la culminación de una tarea, permite ingresar el ID único para cada tarea para notificar que esta ha sido terminada

- **Pestaña-Ver estado tarea**



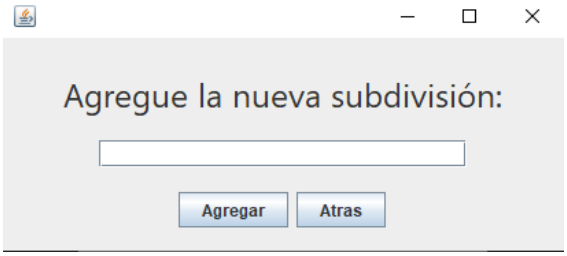
Pestaña para indicar la subdivisión la cual queremos conocer el estado de sus tareas.

- **Pestaña-Tablas con los estados de las tareas**



Pestaña donde se mostrarán en tablas el estado de las tareas de una subdivisión específica.

- **Pestaña-Agregar subdivisiones**



Pestaña que permite agregar una nueva subdivisión de trabajo dentro del equipo, al cual se le pueden asignar sus propias tareas.

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El software se ha desarrollado en Java con IDE NetBeans en sistemas operativos de Windows.
El proyecto puede ser ejecutable en cualquier plataforma que soporte java, con un hardware básico.

VII. DISEÑO, IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Para el prototipo del programa se usaron 5 diferentes tipos de estructuras de datos abstractas para el buen funcionamiento del programa.

Los objetos de la clase subdivisión, clase principal de la implementación, tienen tres estructuras como atributos, los tres estados posibles de las tareas. Las tareas sin asignar serán guardadas en un árbol max Heap llamado "Tareas", el cual utiliza un array dinámico en su implementación. De esta forma se consigue que las tareas se organicen y distribuyan según la prioridad asignada a cada una, de forma que se realicen primero las tareas de máxima prioridad.

Las "Tareas en progreso", están guardadas en un árbol AVL, de forma que la complejidad Big O sea logarítmica al momento de eliminar e insertar tareas dentro de esta estructura. Además, el AVL también es implementado para almacenar las subdivisiones, de forma que su búsqueda, en caso de que hayan demasiadas subdivisiones, también sea logarítmica. Y por último las "Tareas finalizadas" se van insertando en una pila.

Adicionalmente, para visualizar el estado de las tareas en tablas se hizo uso de Arrays estáticos para ingresar las filas dentro de las tablas creadas en la pestaña correspondiente.

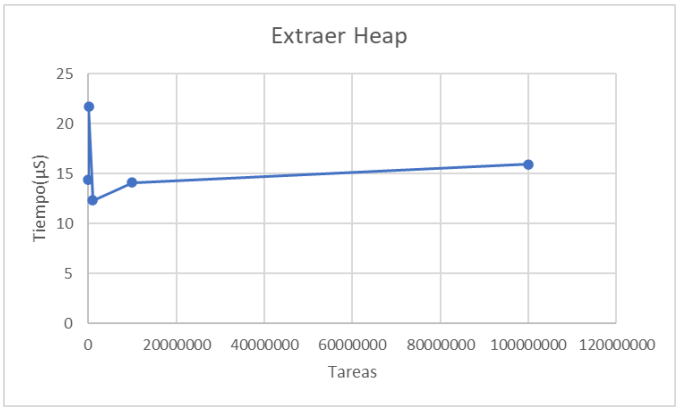
VIII. PRUEBAS DEL PROTOTIPO DE SOFTWARE

- Asignar tarea:

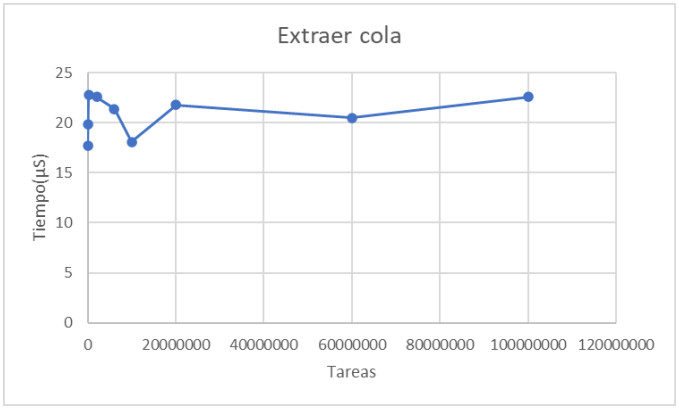
En esta funcionalidad se realiza dequeue en la cola y PushBack en la lista

- Análisis Extraer en el Heap(Lista prioritaria)
  Se espera una complejidad O(logn)

Extraer Heap	
Tareas	Tiempo(µS)
10000	14.4
100000	21.7
1000000	12.3
10000000	14.1
100000000	15.9



A continuación adjuntamos la gráfica resultado de desencolar en una cola unidimensional con el objetivo de comparar

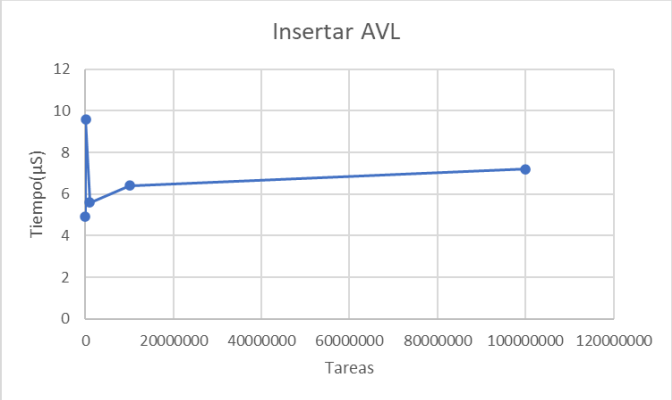


Como se observa son gráficas muy similares, lo más diferencial es que se presenta una tendencia de pendiente muy baja en el Heap, lo cual es coherente con la teoría para complejidad O(logn).

- Análisis insertar en el AVL
  Se espera una complejidad O(logn)

Insertar AVL	
Tareas	Tiempo(µS)
10000	4.9
100000	9.6
1000000	5.6

10000000	6.4
100000000	7.2

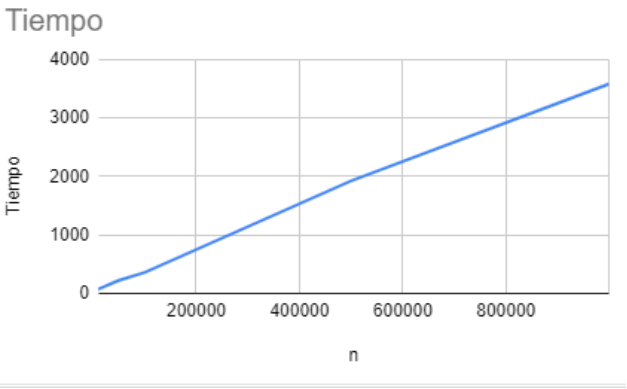


A continuación adjuntamos la gráfica resultado de insertarla en una lista doblemente enlazada, la cual presenta una complejidad constante.

Nuevamente las gráficas son muy similares gracias a la similitud entre complejidad constante y logarítmica.

- Ingresar nueva tarea:
  Implementación principal, cola en una lista simplemente enlazada.
  Complejidad esperada O(n).

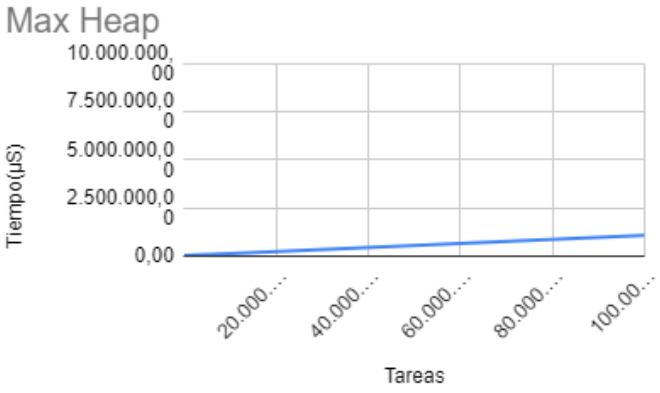
Cola Lista Simplemente Enlazada	
Tareas	Tiempo(µS)
5000	8
10000	76
50000	226.1
100000	363.7
500000	1924.8
1000000	3580.5



Podemos observar que la gráfica tiende a comportarse linealmente por lo que la complejidad esperada de O(n) se cumple.

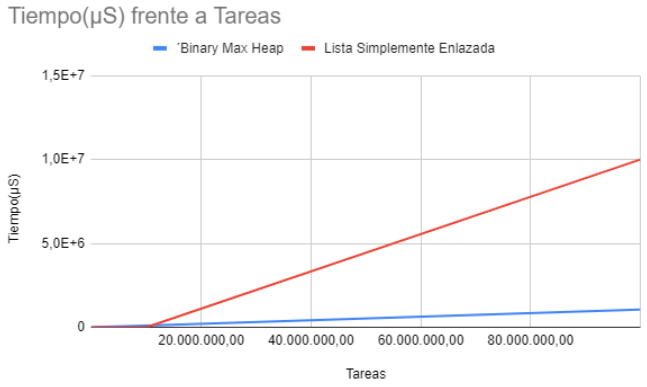
Implementación de ingresar tarea en un Max Heap.
Complejidad esperada O(tree height).

Binary Max Heap	
Tareas	Tiempo(µS)
10.000,00	74,50
100.000,00	2.401,20
1.000.000,00	13.600,20
10.000.000,00	89.371,40
100.000.000,00	1.056.036,20



En la gráfica se puede observar como la gráfica tiende a ser constante ya que es de acuerdo al valor de la altura del árbol con diferentes cantidades de datos, haciendo que la teoría se cumpla en la práctica.

También se comparó con la anterior implementación con estructuras lineales:



Como nos podemos dar cuenta, el Max Heap es mucho más veloz que la implementación de la lista simplemente enlazada para agregar las tareas.

- **Búsqueda de subdivisiones**

La búsqueda de subdivisiones es una operación que es necesaria en 4 de las 5 funciones que tiene este programa, ya que antes de poder hacer cualquier modificación o consulta sobre cualquiera de los atributos de una subdivisión específica, primero hay que encontrar la subdivisión en la que está interesado el usuario la cual se encuentra guardada dentro de una estructura de datos junto con todas las otras subdivisiones. En este caso, se hará un comparación de la operación de búsqueda entre una lista doblemente enlazada con cola (en esta estructuras se guardaron las subdivisiones durante la entrega anterior) y el AVL (estructuras actuales donde se guardan las subdivisiones).

Resultados AVL

Complejidad esperada: O(log n)

AVL (Búsqueda)	
Tareas	Tiempo(µS)
10000	8000
100000	23500
1000000	9100
10000000	9300
100000000	26300

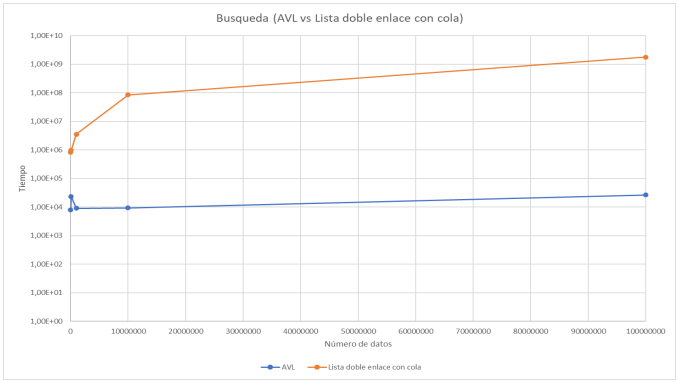
Resultados Lista doblemente enlazada con cola

Complejidad esperada: O(n)

Lista doblemente enlazada con cola (Búsqueda)	
Tareas	Tiempo(µS)
10000	827200
100000	972100
1000000	3506800
10000000	85290900
100000000	1773128200

Gráfica comparativa

En la gráfica presentada el tiempo está en escala logarítmica con el fin poder visualizar mejor la diferencia entre las operaciones de búsquedas de ambas estructuras.



Como se puede ver, la diferencia en tiempos de ejecución para la búsqueda de subdivisiones es mucho más grande en la lista doblemente enlazada con cola que en el AVL, lo cual nos permite confirmar que hacer uso de un AVL para realizar estas búsquedas resulta muy beneficioso para el proyecto

IX. ACCESO AL REPOSITORIO DE SOFTWARE

Link del repositorio del proyecto en github: https://github.com/adrianram21/Task-It

X. ACCESO AL VIDEO DEMOSTRATIVO DE SOFTWARE

Link video demostrativo de proyecto: https://youtu.be/8tPsADLwC6o

XI. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS (Listado)
Adrian Ramirez	Líder	Distribución de trabajo.
	Tecnico	Diseño de interfaz.
		Implementación de estructuras de datos.
Santiago Avila	Coordinador	Programar funciones.
		Redactar documento.
	Programador	
Mateo Bustos	Investigador	Buscar cómo solucionar los nuevos desafíos encontrados.
		Lectura de la bibliografía en busca de información útil.
	Observador	Analizar el trabajo en equipo y ver los aspectos en los que flaquea.
		Comprobación del código base.

XII. DIFICULTADES Y LECCIONES APRENDIDAS

Para esta segunda entrega inicialmente existieron algunas complicaciones relacionadas con la implementación del AVL debido principalmente al uso extensivo de métodos recursivos dentro de este. Por otro lado, aún siguen existiendo diferencias en la forma en la que cada integrante plantea y propone usar las implementaciones de las estructuras dentro del programa, lo cual, aunque implica un constante intercambio de ideas el cual permitió mejorar las soluciones ofrecidas, también hace que el desarrollo del proyecto pueda volverse lento.

Entre las lecciones más importantes que obtuvimos a lo largo de esta segundo entrega, se encuentra el haber entendido de mejor manera la gran utilidad que nos ofrece la recursividad para solucionar distintos problemas de manera fácil y eficiente, como es el caso de la implementación del AVL el cual se beneficia del uso de esta técnica.

XIII. REFERENCIAS BIBLIOGRÁFICAS

[1] J. T. Streib y T. Soma, “Guide to Data Structures: A Concise Introduction Using Java ” . Suiza: Springer International Publishing, 2017.