Bocu, Bogotá Cultural

Deivid Farid Ardila Herrera, Diego Alejandro Arevalo Arias, Angel David Beltran García, Cristofer Damian Camilo Ordonez Osa.

No. de equipo de trabajo: {*B*}

I. INTRODUCCIÓN

El presente documento define las características iniciales del proyecto a desarrollar, describiendo la problemática, los tipos de usuario, los requerimientos funcionales, su interfaz de usuario preliminar, los entornos de operación y desarrollo, las pruebas de complejidad para las estructuras de datos lineales, y el prototipo preliminar. Así mismo, se establecen los roles y tareas que tendrá cada integrante del equipo durante la elaboración del producto de software.

El problema adjunto al proyecto reside en la dificultad de los pequeños y jóvenes artistas para hacerse visibles en el ciberespacio y -principalmente- en Bogotá, lo cual propicia la decadencia cultural, artística y social de la ciudad capitalina, que ha tenido su génesis en diversos dilemas políticos y sociales.

Por otro lado, se tomarán en cuenta, esencialmente, tres tipos de usuario: invitado, que solo podrá navegar en la página principal donde encontrará diversas actividades organizadas; usuario registrado, que tendrá una cuenta propia personalizable en la que podrá guardar sus preferencias; y expositor, que será capaz de crear eventos o actividades, especificando la información correspondiente.

En primera instancia, el producto de software tendrá las siguientes funcionalidades: inicio y cierre de sesión; creación de una cuenta; y acceso a la página principal, tendencias, perfil y configuración. Las funcionalidades dependen del tipo de usuario y se muestran en la interfaz preliminar definida en el punto V.

El proyecto será elaborado principalmente en Android Studio, un entorno de desarrollo para aplicaciones móviles; de la misma forma, se usará como lenguaje base Java, dada la comodidad que le aporta a los integrantes del equipo.

Esta primera versión contempla todas las estructuras de datos secuenciales, tales como colas, pilas, estructuras ordenadas y desordenadas.

El prototipo en todas sus versiones se encontrará en un repositorio de GitHub, con el fin de tener un óptimo control de las versiones del proyecto.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

La evidente y creciente interconexión digital ha facilitado profundamente la difusión casi irrestricta e instantánea de la información; actualmente, es posible enterarse de lo que sucede en cualquier parte del mundo, los datos viajan de persona a persona, aparentando no tener ningún tipo de traba. Sin embargo, esto no implica que los pequeños y medianos artistas de toda índole puedan tener visibilidad en el amplio y saturado universo del internet, solo aquellos con el suficiente poder, influencia y capital tienen una alta probabilidad de conseguir un lugar privilegiado en el ciberespacio, sin importar la utilidad y la idoneidad del contenido que aportan.

Una cantidad significativamente grande de información cultural provechosa se pierde constantemente. La publicidad solo es para aquellos con el dinero suficiente para pagarla, es un negocio que degrada la libre circulación del arte.

El presente proyecto busca dar a conocer a los artistas emergentes que no tienen una proyección alentadora en el ciberespacio, mediante la presentación de los eventos de los cuales son partícipes y/o organizadores.

Por otro lado, Bogotá -ciudad en la que se enmarca la problemática- fue catalogada en los 80's por José Maria Vergara [1] como "la Atenas de Sudamérica", dada su extensa oferta cultural y su destacado grupo de intelectuales, entre los que se encontraba Gabriel García Márquez, galardonado con el premio Nobel de Literatura. No obstante, con el pasar de los años se presentaron diversos problemas políticos y otras circunstancias sociales que deterioraron el producto cultural de la zona, haciendo que el arte pasará a un tercer plano.

De esta forma, el actual proyecto intenta también mejorar el decreciente panorama cultural de Bogotá mediante la publicitación de sus eventos artísticos y sociales, así como de sus jóvenes artistas.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

A continuación se encuentran los diversos roles que puede tener un usuario en la aplicación.

Invitado: será capaz de ingresar a la app y navegar en la página principal donde encontrará diversos eventos y distintas funcionalidades, como filtrar y consultar las diversas actividades de la ciudad. También podrá crear una cuenta de tipo individual o como expositor en el apartado *cuenta*.

Usuario registrado: será capaz de ingresar a la app y en el apartado de *cuenta* editar su información; podrá añadir preferencias en cuestión de los temas de las actividades, tendrá acceso a una página principal personalizada de acuerdo a sus intereses, y le será posible buscar, filtrar y consultar las diversas actividades de la ciudad, además podrá guardar sus actividades favoritas en una sección específica.

Expositor: podrá ingresar a la aplicación, editar su cuenta, navegar en la página principal, y tanto filtrar como consultar las diversas actividades de la ciudad. Además, será capaz de crear actividades en las cuales tendrá que mencionar el nombre, tipo, costo, ubicación, fecha, horario, etc.; con el fin de que los demás usuarios de la app puedan conocer sobre estas.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

1. Inicio de sesión

Descripción: al entrar al apartado *cuenta*, se mostrará la opción de iniciar sesión. El usuario podrá acceder si ya posee una cuenta en la aplicación.

Acción iniciadora: ingresar con el botón “Iniciar sesión”

Comportamiento esperado: el sistema deberá cambiar a la ventana que muestre la información de inicio de sesión y validar los datos que ingrese el usuario para acceder a la aplicación con la cuenta abierta.

Requerimientos funcionales:

- *Consulta de datos:* se debe buscar el nombre de usuario y validar la contraseña en la estructura de datos.
- *Validación de información:* si el usuario existe, se valida la contraseña. Si la contraseña coincide con la almacenada, se ingresa a la página de inicio; de lo contrario, se mostrará una ventana que informe que el usuario o la contraseña son incorrectos.

2. Crear cuenta

Descripción: si el usuario desea crear una cuenta, deberá ingresar la información pertinente para acceder a la aplicación.

Acción iniciadora: ingresar con el botón “Crear cuenta”.

Comportamiento esperado: el sistema deberá cambiar a la ventana que muestre la opción de ingresar como usuario o como artista; luego, mostrará la información para la creación de la cuenta y así validar los datos que ingrese el usuario para crear la cuenta, almacenarla e ingresar a la aplicación con la cuenta abierta.

Requerimientos funcionales:

- *Validación de datos:*
    - *Usuario:* para crear una cuenta como usuario, se deberá ingresar: nombre, fecha de nacimiento, correo, localidad de residencia, contraseña, confirmación de contraseña e intereses.
    - *Artista:* para crear una cuenta como artista, se deberá ingresar: nombre del usuario u organización, correo, contraseña, confirmación de contraseña y tipo de eventos.

  Al ingresar la información anterior y dar clic en el botón “Aceptar”, el sistema debe corroborar los datos con las siguientes excepciones:

    - Se deben llenar todos los espacios.
    - La contraseña debe tener como mínimo 8 caracteres
    - El correo electrónico debe tener una dirección válida.
    - El espacio de “Contraseña” y “Confirmar contraseña” deben contener la misma información.
- *Creación de datos:* al crear una cuenta, se crea un objeto de la clase usuario para manipular la información mediante estructuras de datos.
- *Almacenamiento:* se toma la información validada anteriormente para ingresarla a una base de datos creada localmente en el dispositivo.

3. Ingreso como invitado

Descripción: si el usuario no desea crear una cuenta, deberá ingresar como invitado para entrar a la aplicación.

Acción iniciadora: ingresar con el botón “Ingresar como invitado”

Comportamiento esperado: el sistema deberá cambiar a la ventana de inicio. Sin embargo, el usuario tendrá funcionalidades acortadas en comparación con un usuario registrado.

Requerimientos funcionales:

- *Validación de datos:* el sistema deberá mostrar sólo la pantalla de inicio y descubrir, debido a las funciones limitadas mencionadas anteriormente.

4. Página de Inicio

Descripción: al acceder al inicio, el sistema deberá mostrar los eventos más cercanos a la fecha (si el usuario lo prefiere), junto con la información relevante.

Acción iniciadora: al entrar a la aplicación o al presionar el botón “Inicio”, se deberá entrar a la Activity correspondiente.

Comportamiento esperado: el sistema deberá cargar al layout la lista con los eventos más cercanos a la fecha establecida (si el usuario lo prefiere) en el teléfono. El usuario podrá acceder a más información presionando cualquier evento.

Requerimientos funcionales:

- *Creación:* en Android Studio se necesitará de un adaptador para crear una lista con los datos de los eventos.
- *Consulta de datos:* se debe recuperar la información de los eventos en la base de datos para mostrarlos en pantalla.
- *Ordenamiento:* los eventos que estén más cerca a la fecha del dispositivo deben aparecer al principio de la lista solo si el usuario lo prefiere, en otro caso, la lista aparecerá desordenada.

5. Acceso a categorías de eventos

Descripción: para buscar algún evento en específico, se puede simplificar el proceso entrando a alguna categoría de eventos.

Acción iniciadora: entrar a la sección “Descubrir” en la aplicación.

Comportamiento esperado: se va a mostrar una lista de categorías. Si el usuario presiona alguna de las categorías, se va a efectuar el mismo proceso que realiza la página de inicio

para mostrar los eventos, sin embargo, en este caso sólo van a aparecer los eventos que concuerden con la categoría elegida.

Requerimientos funcionales:

- *Creación:* en Android Studio se necesitará de un adaptador para crear una lista con los datos de los eventos.
- *Consulta de datos:* se debe recuperar la información de los eventos en la base de datos para mostrarlos en pantalla. En este caso, se debe especificar la búsqueda de eventos que concuerden con la categoría de evento que se quiere tener.
- *Ordenamiento:* los eventos que estén más cerca a la fecha del dispositivo deben aparecer al principio de la lista (si el usuario lo decide). Si no, se va a mostrar de forma desordenada.

6. Acceso a eventos

Descripción: para acceder a mayor información acerca del evento en pantalla, se puede presionar en cualquier sitio del recuadro.

Acción iniciadora: presionar el recuadro del evento.

Comportamiento esperado: se muestra una nueva Activity con toda la información relacionada al evento.

Requerimientos funcionales:

- *Consulta de datos:* se debe recuperar la información del evento en la base de datos para mostrarlos en pantalla.

7. Página de cuenta

Descripción: el usuario o artista puede revisar la información suministrada a la aplicación mediante la sección de cuenta

Acción iniciadora: acceder mediante el botón “Cuenta”.

Comportamiento esperado: se debe iniciar una Activity que muestre la información almacenada del usuario.

Requerimientos funcionales:

- *Actualización:* si el usuario desea cambiar algún dato de la cuenta, se deben tener en cuenta las excepciones descritas en el apartado “Crear cuenta”. Al validar la información, el sistema debe tomar los datos ingresados y reemplazarlos en la base de datos existente.
- *Consulta de datos:* para mostrar la información relevante de la cuenta, se debe acceder a la base de datos mediante el correo o nombre del usuario. La información se traslada a la clase Usuario para mostrar la información mediante POO.
- *Almacenamiento:* para la actualización de datos, se maneja la base de datos ubicada localmente en el dispositivo.

8. Página de eventos

Descripción: el artista posee una sección especial para crear, modificar o eliminar eventos.

Acción iniciadora: presionar el botón “Eventos”

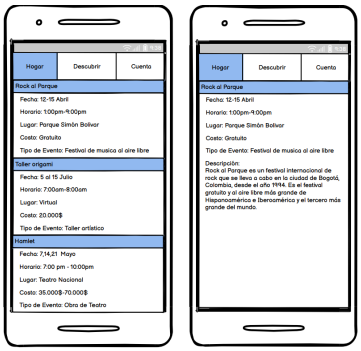
Comportamiento esperado: se debe mostrar una lista con los eventos que ha creado el artista. Por defecto debe estar vacía.

Requerimientos funcionales:

- *Creación:* en Android Studio se necesitará de un adaptador para crear una lista que contenga los datos de los eventos, con un filtro que indique el artista que lo creó. Para crear un nuevo evento, se oprime el botón “Crear”, que llevará a una nueva Activity, la cual pedirá: nombre del evento, descripción, categoría del evento, localidad dónde se va a realizar el evento, dirección específica, fecha, hora y costo.
- *Validación de datos:* para crear un evento, se deben considerar las siguientes excepciones: se deben llenar todos los campos, el nombre del evento debe ser de máximo 50 caracteres, y la descripción debe ser de máximo 500 caracteres. Al corroborar lo anterior, el sistema toma la información para almacenarla en la base de datos.
- *Consulta de datos:* se debe recuperar la información de los eventos creados por el artista en la base de datos para crear un objeto de la clase Evento y así mostrarlo en pantalla.
- *Almacenamiento:* para la actualización o eliminación de eventos, se maneja la base de datos ubicada localmente en el dispositivo.
- *Actualización:* si el artista desea cambiar la información del evento, se debe consultar dicho evento en la base de datos para mostrarlo en pantalla. Luego de hacer los cambios, se deben corroborar las excepciones para acceder a la base de datos y modificar la información.
- *Ordenamiento:* los eventos creados por el artista se muestran del más reciente al más antiguo, sólo si el artista así lo prefiere; en caso contrario, los eventos creados no se muestran de forma organizada.

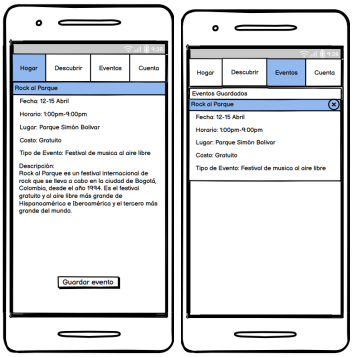
V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

Página principal y vista eventos:



Cuenta y Descubrir Eventos:

Espacio para guardar eventos:



Espacio para crear y editar un evento:

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

Para la creación del software se va a utilizar el entorno de desarrollo integrado Android Studio, ya que provee las herramientas necesarias para crear aplicaciones. Los lenguajes de programación en los que se basa Android Studio son Java y Kotlin, sin embargo, se va a utilizar el primero por la comodidad que ofrece al grupo de trabajo.

El entorno seleccionado para poner en operación el proyecto es el sistema operativo Android. Los dispositivos a utilizar para emular la aplicación son: Huawei Y9 Prime 2019, Samsung Galaxy A52 y Samsung Galaxy A51.

Para colaborar de forma simultánea a lo largo del proyecto, se va a utilizar Github, que facilita la creación y manipulación de repositorios en la nube. La web utiliza el sistema de control de versiones Git, para manejar el código con mayor facilidad entre dispositivos.

VII. PROTOTIPO DE SOFTWARE INICIAL

El link del repositorio es el siguiente:

https://github.com/faridardila/DataStructuresProjectGroupB_Entrega1

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

Clases:

1. Usuario:
    - Descripción: representa un usuario dentro de la app, ya sea: registrado o expositor.
    - Atributos Usuario registrado: nombre, fecha de nacimiento, correo, localidad de residencia, contraseña e intereses.
    - Atributos Expositor: nombre del usuario u organización, correo, contraseña y tipo de eventos.
    - Funcionalidades: actualizar información del usuario.
2. Evento:
    - Descripción: representa un evento creado por un expositor.
    - Atributos: nombre del evento, descripción categoría del evento, localidad dónde se va a realizar el evento, dirección específica, fecha, hora y costo.
    - Funcionalidades: actualizar información de los eventos.

Estructuras de datos:

1. Listas enlazadas (administración de usuarios):
    - Implementación: búsqueda y comparación de datos.
    - Contribución: evitar la creación de usuarios repetidos y permitir las funcionalidades de *Usuario Registrado o Expositor.*
2. Listas desordenadas con arreglos dinámico (página de inicio y página de eventos):
    - Implementación: mostrar en el recyclerView los eventos generados por los *expositores*.
    - Contribución: permitir visibilizar los eventos a todos los usuarios.
3. Pilas con listas enlazadas (historial de eventos):

- Implementación: crea una lista que muestra los eventos desde el último visto hasta el primero.
- Contribución: permite al usuario revisar y acceder a los eventos vistos anteriormente.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Para todas las implementaciones de estructuras lineales se hicieron pruebas de rendimiento que tenían el objetivo de medir el tiempo promedio de procesamiento por dato que requería ejecutar una funcionalidad específica, tomando en cuenta diferentes magnitudes de datos: 10 mil, 100 mil, 1 millón, 10 millones y 100 millones. Dicho tiempo fue medido en nanosegundos.

Como se verá más adelante, algunas pruebas carecen de resultados para ciertas cantidades de datos; esto se debe a que se presentaron errores de falta de memoria, principalmente *Java heap space* y *GC overhead limit exceeded*, y -en otros casos- el tiempo de espera para conseguir resultados superó el día.

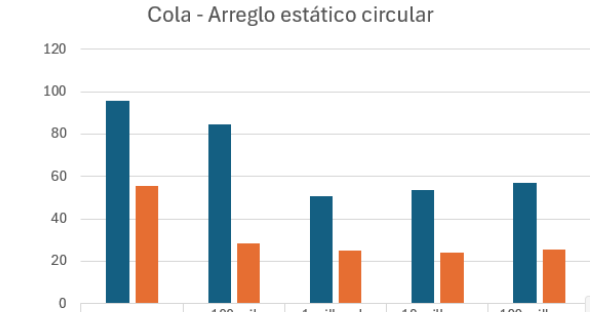
1. Arreglos Estáticos

Para el uso de arreglos estáticos, en cada caso se creó una clase con un array de longitud equivalente a la cantidad de datos especificada.

Colas

En el caso de la estructura de cola, se realizaron dos enfoques para encolar y desencolar: El primero, utilizando un arreglo estático; y el segundo, agregando la metodología de arreglo circular.

- Cola con arreglo circular:

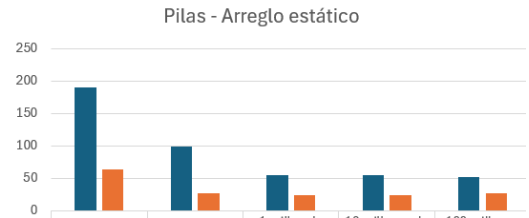


	10 mil datos	100 mil datos	1 millon de datos	10 millones de datos	100 millones de datos
Enqueue	95,53	84,665	50,482	53,56444	56,872233
Dequeue	55,67	28,273	24,8526	24,23669	25,355851

- Cola con arreglo normal:

	10 mil datos	100 mil datos	1 millon de datos
enqueue	455,94	73,628	38,7697
dequeue	37863,51	295841,513	10243793,61

Pilas



	10 mil datos	100 mil datos	1 millon de datos	10 millones de datos	100 millones de datos
Push	190,58	98,487	54,962	54,54717	51,311578
Pop	63,32	27,305	23,66363	23,66363	27,564322

Listas desordenadas

	10 mil datos	100 mil datos	1 millon de datos	10 millones de datos	100 millones de datos
Inserción	133,61	109,294	58,9794	57,56286	49,306475
Eliminación	100,78	58,475	36,3202	34,12132	31,508599
Búsqueda	57,12	56,572	25,3755	22,8227	21,230309

Listas Ordenadas

	10 mil datos	100 mil datos	1 millon de datos
Inserción	8828,17	67761,43	693913,48
Eliminación	12907,24	345210,4754	4224294,354
Búsqueda	56	82,252	25,3755

Al analizar los gráficos de las distintas estructuras se concluye que en ciertos casos el procesamiento de la estructura en sus distintos métodos no está fuertemente relacionado con la cantidad de datos y en otros si. Por lo tanto, para los siguientes métodos, la complejidad es O(1): enqueue y dequeue en colas con arreglo circular, enqueue en colas con arreglo estático, push y pop en pilas, búsqueda en listas ordenadas, y tanto búsqueda como eliminación e inserción en listas desordenadas.

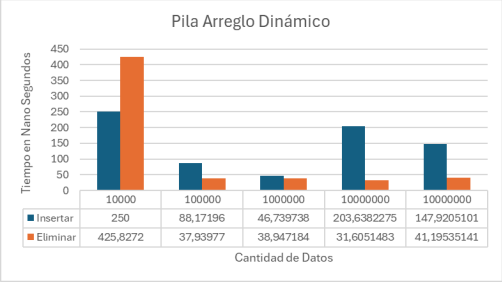
La complejidad es O(n) para los siguientes métodos: dequeue en colas con arreglo normal, y tanto inserción como eliminación en listas ordenadas.

2. Arreglos Dinámicos

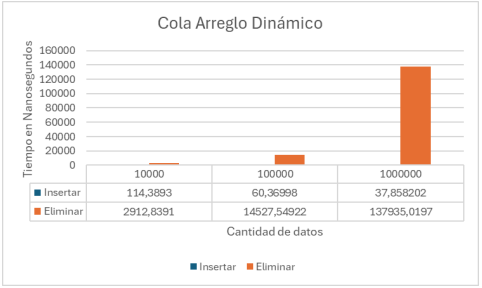
Al implementar arreglos dinámicos se inicializa un arreglo con 10 espacios, el cual, una vez está lleno, se duplica de tamaño; para las magnitudes de datos con las cuales se hicieron las pruebas fueron necesarias la siguientes cantidades de redimensionamientos:

Cantidad de Datos	Veces que fue redimensionado
10000	10
100000	14
1000000	17
10000000	20
100000000	24

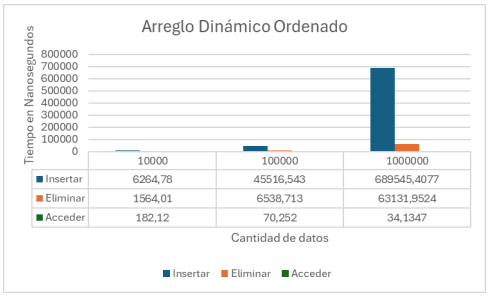
Pila



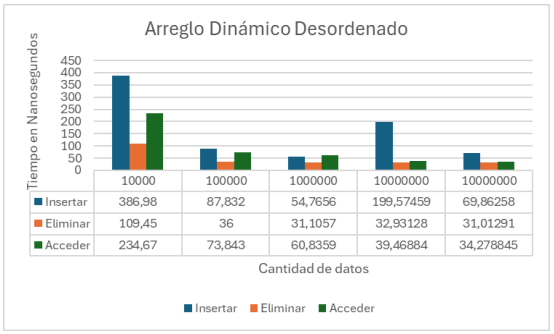
Cola



Arreglo Ordenado



Arreglo Desordenado



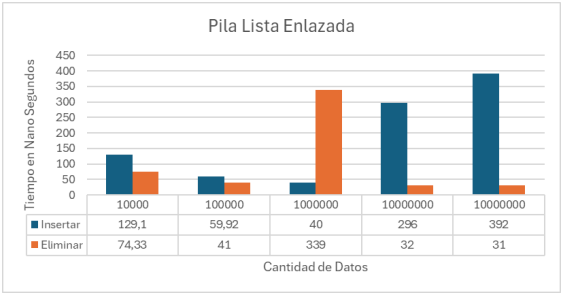
Al analizar los resultados es evidente que en ciertos casos los tiempos no dependen de la cantidad de datos ingresados, mientras que en otros el tiempo se ve fuertemente afectado, por lo tanto, los métodos que tienen complejidad constante O(1) son: inserción y eliminación en pilas, inserción en colas, acceso por índice en listas ordenadas, y acceso, inserción y eliminación en arreglos desordenados.

Además, los métodos que tienen complejidad lineal (O(n)) son: eliminación en colas y tanto inserción como eliminación en listas ordenadas

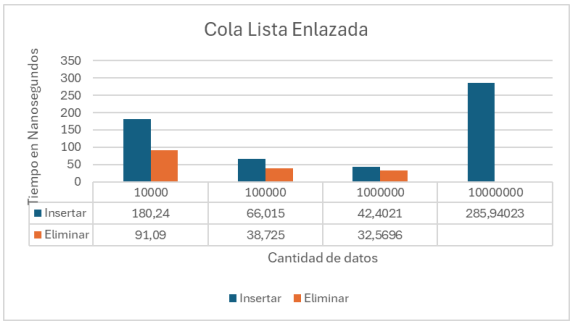
3. Linked List

Para desarrollar las listas enlazadas se utilizaron nodos con dos atributos: *data* de tipo genérico, que contiene el dato almacenado en el nodo, y *next* de tipo *Node*, que guarda la referencia del nodo posterior.

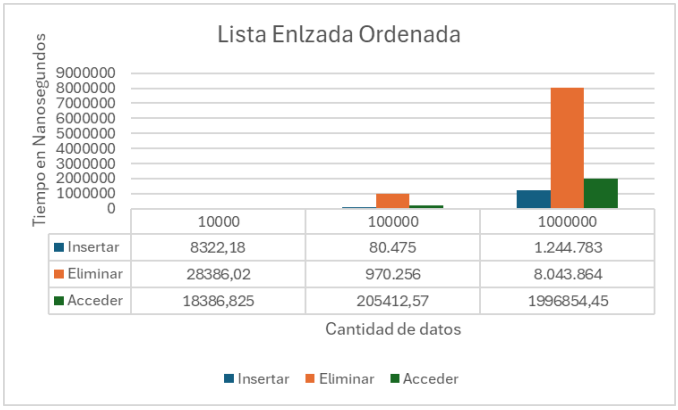
Pilas



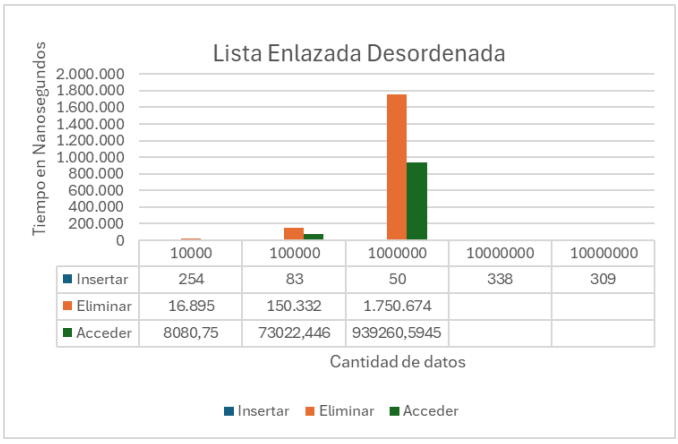
Cola



Listas Ordenadas



Listas Desordenadas



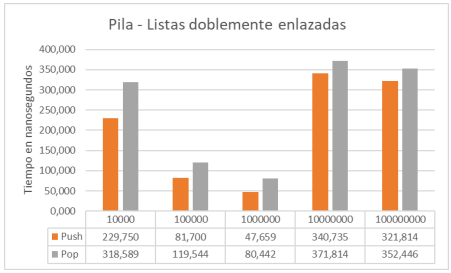
Después de revisar las gráficas obtenidas y llevar a cabo el respectivo análisis asintótico, se determinó que los métodos push, pop, enqueue, dequeue, e inserción en listas desordenadas poseen una complejidad O(1); así mismo, las funcionalidades que tienen complejidad O(n) son inserción para listas ordenadas y tanto eliminación como acceso para listas ordenadas y no ordenadas.

4. Double Linked List

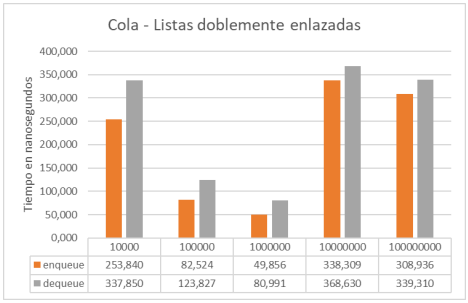
Al igual que las listas enlazadas, las doblemente enlazadas utilizan, por nodo, los atributos *data* y *next*, el primero de tipo genérico y el segundo de tipo *Node*, sin embargo, esta implementación utiliza un nodo adicional llamado *prev* de tipo *Node*, que almacena la referencia del nodo anterior.

En las estructuras de datos generales encontramos la siguiente información.

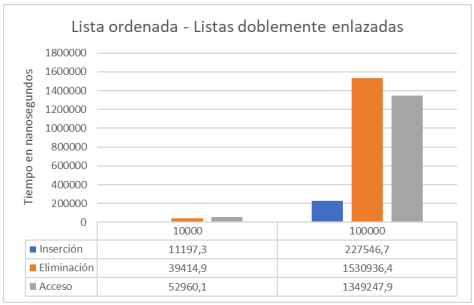
Pilas



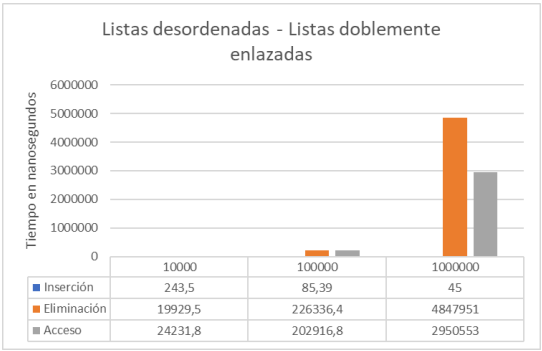
Colas.



Listas ordenadas.



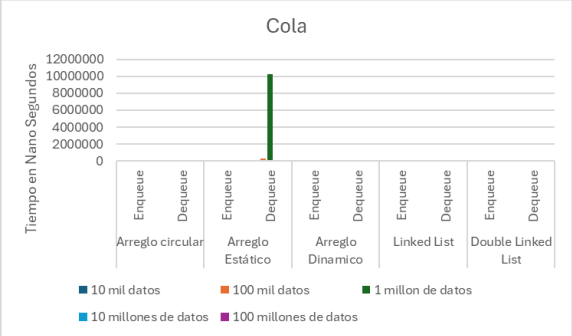
Listas desordenadas.

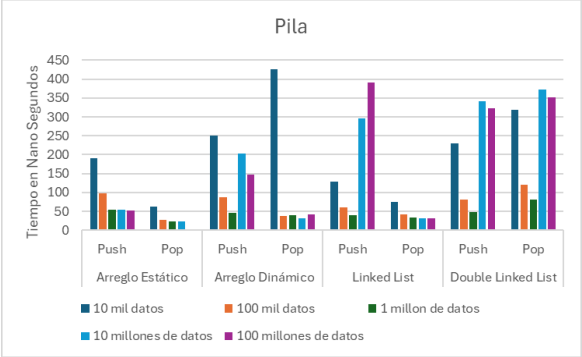


Al examinar cada una de las gráficas anteriores y realizar el respectivo análisis asintótico, se encontró que los métodos push, pop, enqueue, dequeue, e inserción en listas desordenadas poseen una complejidad O(1); de la misma forma, las funcionalidades que presentan complejidad O(n)

son inserción para listas ordenadas y tanto eliminación como acceso para listas ordenadas y no ordenadas.

En las estructuras de datos generales encontramos la siguiente información.





En conclusión, las mejores implementaciones para colas fueron con arreglos estáticos circulares, para las demás estructuras fueron con arreglos estáticos, que en general fueron las más eficientes; sin embargo, es necesario tener en cuenta el contexto de uso, ya que puede que se requiera almacenar una longitud indeterminada de datos. También depende del problema o de lo que se quiera hacer, pues puede ser más sencillo implementar unas estructuras en específico incluso si no son las más eficientes.

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

https://drive.google.com/drive/folders/15_-eG5SVKDk34VKrz7Br3tsFfCbIOmfQ?usp=sharing

XI. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS
Cristofer Ordoñez	Líder Experto	Descripción del problema a resolver. Descripción general del primer prototipo implementado.
Farid Ardila	Coordinador Observador	Entornos de desarrollo y de operación (ejecución y despliegue). Requerimientos funcionales del software. Tabla resumen de los integrantes del equipo, sus roles y las actividades realizadas.
Diego Arévalo	Técnico Investigador	Descripción preliminar de la interfaz (vista gráfica) de usuario. Usuarios del producto de software.
Ángel Beltran	Animador Secretario	Descripción del diseño y de la implementación.

XII. DIFICULTADES Y LECCIONES APRENDIDAS

El análisis asintótico y las pruebas de rendimiento fueron esenciales para determinar la complejidad de las principales funcionalidades aplicadas en el proyecto.

Las pruebas de rendimiento demandaron una cantidad considerablemente grande de tiempo.

Las reuniones recurrentes fueron un aspecto clave para el correcto desarrollo del proyecto.

Los roles de equipo ayudaron a distribuir la carga de trabajo de forma equitativa.

Los requerimientos funcionales deben resolver las necesidades del usuario de forma eficiente.

XIII. REFERENCIAS BIBLIOGRÁFICAS

[1] A. Montenegro González, «La “Atenas suramericana”. Búsqueda de los orígenes de la denominación dada a Bogotá», Mem.soc, vol. 7, n.º 14, pp. 133–143, mar. 2014.W.-K. Chen, *Linear Networks and Systems* (Referencia de libro). Belmont, CA: Wadsworth, 1993, pp. 123–135.