UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
"La decana de América"
Facultad de Ingeniería de Sistemas e Informática

Entregable N°01 del Proyecto del Curso

Integrantes del grupo 10:

Gúzman Romero, Diego Alonso

Madrid Ruiz, Giacomo Salvador

Pardave Jara, Asthri Joanne

Patricio Julca, Vilberto Alberto

Segura Pacherres, Leonardo Gabriel

Curso:

Análisis y Diseño de Algoritmos

Docente:

Jorge Luis Chávez Soto

Lima, Perú

2025

Índice

1. Definición del proyecto

1.1. Definición

Este proyecto tiene como finalidad desarrollar un programa que permita analizar el comportamiento temporal de distintos algoritmos. Su propósito es desarrollar una herramienta que automatice el análisis de complejidad temporal, facilite la comparación entre algoritmos y promueva un entendimiento sobre la eficiencia de distintos algoritmos.

El sistema analizará el código fuente de algoritmos, especialmente aquellos que incluyen bucles anidados (oLoop), para estimar su complejidad temporal, construir su función de tiempo de ejecución en función del tamaño de entrada “n” y graficarla. Además, permitirá comparar tanto la complejidad teórica como el tiempo de ejecución real de múltiples algoritmos que resuelven el mismo problema.

1.2. Objetivos

Los objetivos planteados en este proyecto están relacionados con los temas abordados dentro del curso de Análisis y Diseño de Algoritmos. Durante el desarrollo del programa se busca aplicar conceptos importantes como el análisis de complejidad temporal, las técnicas de diseño algorítmico y la comparación de soluciones computacionales para aplicar lo aprendido en clase.

- Desarrollar un programa que analice el pseudocódigo de algoritmos e identifique su complejidad temporal estimada utilizando notaciones asintóticas.
- Generar y graficar automáticamente la función de tiempo de ejecución (T(n)) para diferentes tamaños de entrada n.
- Comparar el tiempo de ejecución teórico y real de múltiples algoritmos, como Bubble Sort, QuickSort u otros similares.

2. Marco Teórico

El presente proyecto tiene como objetivo desarrollar una aplicación en Java capaz de analizar el código fuente de un algoritmo para determinar su tiempo de ejecución estimado. A partir del análisis estructural del código, se construirá su función de

tiempo equivalente, la cual será graficada y comparada con otras funciones de complejidad.

Para ello, es necesario conocer conceptos clave como los algoritmos, la complejidad algorítmica, el análisis estático de código y la visualización de datos. También se requiere el uso de técnicas modernas de desarrollo en Java, como la programación orientada a objetos y el uso de estructuras de análisis sintáctico.

A continuación, se presentan los principales fundamentos que sustentan este proyecto:

- **Complejidad algorítmica:** La complejidad de un algoritmo es una medida de cuán eficiente es el algoritmo para resolver el problema. En otras palabras, es una medida de cuánto tiempo y espacio (memoria) requiere el algoritmo para producir una solución. (Calvo, 2022)

Se utiliza la notación asintótica, como:

- O(n): lineal
- $O(n^2)$: cuadrática
- O(log n): logarítmica
- O(n log n): casi lineal

- **Función de tiempo:** Representa matemáticamente la cantidad de operaciones que realiza un algoritmo en función de su entrada. Es el modelo que se grafica para visualizar el comportamiento del tiempo de ejecución.
- **Algoritmos de búsqueda:** Un algoritmo de búsqueda es un conjunto de instrucciones que están diseñadas para localizar un elemento con ciertas propiedades dentro de una estructura de datos. (Ponce, 2021)
- **Recursividad y bucles:** Los bucles permiten repetir un bloque de código mientras se cumpla una condición. En VBA existen varios tipos como For, While, Do While y Do Until, cada uno útil según el caso. Por ejemplo, un bucle For se usa para ejecutar instrucciones un número fijo de veces, como recorrer una matriz. La recursividad es una técnica en la que una función se llama a sí misma para resolver un problema dividiéndolo en partes más simples. Es útil en estructuras como árboles o cálculos como el factorial.
- **Visualización de datos:** La visualización de datos consiste en representar información mediante gráficos o diagramas, lo que facilita identificar patrones, tendencias y anomalías. En contextos de big data, es clave para analizar grandes volúmenes de datos y tomar decisiones informadas.

- **Comparación de algoritmos:** Evaluar la eficiencia de un algoritmo implica determinar qué tan bien resuelve un problema en términos de tiempo de ejecución y uso de memoria, dos recursos críticos en computación. Esta evaluación se puede hacer de dos formas.
    - Análisis teórico.- Describe cómo crece el número de operaciones del algoritmo en función del tamaño de la entrada (n). Se utiliza la notación asintótica.
    - Experimental.- Se basa en ejecutar el algoritmo con diferentes tamaños de entrada y medir su tiempo de ejecución real o número de operaciones efectivas. Esto se hace con herramientas de profiling o funciones de cronometraje.
- **Optimización de algoritmos:** La optimización de algoritmos se refiere al proceso de modificar algoritmos existentes para mejorar su eficiencia en cuanto al uso de recursos computacionales, como tiempo de ejecución y memoria. Este proceso implica identificar cuellos de botella y aplicar técnicas como la reducción de la complejidad algorítmica, el uso adecuado de estructuras de datos y la eliminación de operaciones redundantes. La optimización es crucial en aplicaciones que requieren procesamiento rápido o manejo de grandes volúmenes de datos, ya que permite obtener soluciones más eficientes y escalables (Rao, 2019). Además, la optimización puede involucrar la selección entre diferentes enfoques algorítmicos según el contexto del problema y las restricciones del sistema.
- **Análisis sintáctico:** El análisis sintáctico, también conocido como parsing, es una etapa fundamental en la interpretación de lenguajes de programación y pseudocódigo. Consiste en examinar la secuencia de símbolos de entrada para determinar su estructura gramatical según reglas predefinidas. Esta técnica permite identificar elementos como bucles, condicionales y llamadas recursivas, facilitando el análisis automático de la estructura y el comportamiento de los algoritmos. El análisis sintáctico es esencial para construir herramientas que procesen código fuente, ya que asegura que el código cumpla con la sintaxis esperada y posibilita la extracción de información relevante para el análisis de complejidad y optimización (Aho & Ullman, 2007; Gutiérrez & Rodríguez, 2018).

3. Alcances y Limitaciones

El desarrollo de este proyecto abordará el cumplimiento de los alcances que se definen en este documento y de las limitaciones que implica la implementación de ciertas funcionalidades.

3.1. Alcances

- El programa permite el ingreso de pseudocódigo de algoritmos (que contengan estructuras repetitivas y/o recursivas) para analizar su complejidad.
- La función de tiempo de ejecución T(n) se grafica de manera automática y se tiene en cuenta distintos valores de entrada “n”.
- Se compara el tiempo de ejecución teórico y real entre distintos algoritmos que resuelven el mismo problema (Bubble Sort vs QuickSort).
- El programa será desarrollado con el lenguaje de programación Java, utilizando el paradigma de programación orientada a objetos y la arquitectura MVC.
- Los análisis pueden exportarse como reportes en PDF para visualizar las estadísticas.

3.2. Limitaciones

- El análisis de complejidad está limitado a fragmentos de pseudocódigo que siga una sintaxis ya definida.
- Se va a estimar el tiempo de ejecución estimado del fragmento de pseudocódigo que se ingrese, aplicando las técnicas aprendidas en clase.
- Se estimará la función de ejecución de manera teórica y real (ejecución de código) en la sección de comparación de distintos algoritmos que resuelven el mismo problema.
- El programa solo se insertará pseudocódigo para su análisis de complejidad algorítmica y función de tiempo de ejecución teórica.

4. Relación de recursos humanos y tecnológicos.

Para este proyecto, hemos organizado a nuestro equipo asignando roles, los cuales representan a nuestros recursos humanos. Por otro lado, hemos seleccionado las tecnologías específicas utilizaremos para la implementación del proyecto.

4.1. Recursos humanos

Rol	Descripción	Responsable
Jefe del proyecto	Persona que coordina, planifica y supervisa el desarrollo del proyecto.	Asthri Pardave Jara
Diseñador UI/UX	Persona encargada del diseño de las interfaces gráficas del programa.	Vilberto Patricio Julca
Analista de algoritmos	Persona encargada de crear las soluciones algorítmicas, analizar su eficiencia y complejidad.	Diego Guzman Romero
Analista QA	Persona encargada de evaluar el funcionamiento y el rendimiento del programa.	Leonardo Segura Pacherres
Jefe del equipo de desarrollo	Persona encargada de liderar el desarrollo del programa.	Giacomo Madrid Ruiz
Equipo de desarrolladores	Personas encargadas del desarrollo e implementación del programa.	Asthri Pardave Jara Diego Guzman Romero Giacomo Madrid Ruiz Leonardo Segura Pacherres Vilberto Patricio Julca

4.2. Recursos tecnológicos

Tecnología	Descripción
Java	Es un lenguaje de programación de propósito general, orientado a objetos y multiplataforma.
Java Swing	Es un conjunto de herramientas de interfaz gráfica de usuario (GUI) que forma parte de Java Foundation Classes (JFC), el cuál permite crear interfaces gráficas de usuario para aplicaciones de escritorio.
FlatLaf	Es una librería de código abierto para Java Swing que proporciona una apariencia y comportamiento moderno para las aplicaciones de escritorio. Permite actualizar la interfaz visual de las aplicaciones Swing.
Figma	Es una herramienta de diseño de interfaces de usuario (UI) y experiencia de usuario (UX). Permite a diseñadores y equipos crear prototipos, maquetas, wireframes y sistemas de diseño para sitios web, aplicaciones móviles y software de escritorio.
Apache Netbeans	Es un Entorno de Desarrollo Integrado (IDE) de código abierto, ampliamente utilizado para el desarrollo en Java, aunque soporta múltiples lenguajes de programación.
Git	Git es un sistema de control de versiones distribuido (DVCS) utilizado para rastrear cambios en el código fuente durante el desarrollo de software.
GitHub	GitHub es una plataforma basada en la web que aloja repositorios Git, proporcionando herramientas adicionales para facilitar la colaboración, la gestión de proyectos y el control de versiones.

5. Cronograma de actividades

El presente proyecto tiene una duración total de tres meses, con fecha de inicio el 1 de abril de 2025 y fecha de finalización el 3 de julio de 2025. El cronograma está estructurado en tres hitos principales: **Análisis**, **Diseño** y **Desarrollo**, los cuales agrupan las distintas actividades necesarias para el cumplimiento de los objetivos del proyecto.

- **Hito 1: Análisis** abarca desde la definición de reglas y procesos del negocio hasta el modelamiento de diagramas y diseño de interfaces gráficas, con actividades desarrolladas entre el 1 de abril y el 9 de junio de 2025.
- **Hito 2: Diseño** comprende tareas como la implementación de clases, el desarrollo de la interfaz gráfica, la integración de la lógica del sistema y la ejecución de pruebas, llevándose a cabo entre el 10 de junio y el 30 de junio de 2025.
- **Hito 3: Desarrollo** cierra el proyecto con la corrección de errores detectados en las pruebas, finalizando el 3 de julio de 2025.

Cada fase ha sido asignada a profesionales con distintos roles, incluyendo analistas, desarrolladores, especialistas UX y responsables de aseguramiento de la calidad (SQA), garantizando así un enfoque multidisciplinario en todas las etapas del proyecto.

Actividad	Apellido/Rol	Inico	Fin
Definir las reglas del negocio	Guzman - Analista	01/04/2025	08/04/2025
Definir los procesos del negocio	Guzman - Analista	09/04/2025	15/04/2025
Realizar análisis de requerimientos	Pardavé - desarrollador	16/04/2025	22/04/2025
Identificación de entidades	Madrid - Jefe de desarrollo	23/04/2025	29/04/2025
Hito 1: Análisis			
Modelamiento de los procesos del negocio	Segura - desarrollador	30/04/2025	11/05/2025
Modelamiento de las clases	Madrid - desarrollador	12/05/2025	26/05/2025
Modelamiento del diagrama de clases	Madrid - desarrollador	27/05/2025	09/06/2025
Diseño de las interfaces gráficas	Patricio - diseñador UX	27/05/2025	09/06/2025
Hito 2: Diseño			
Desarrollo de las clases con métodos y	Segura - desarrollador	10/06/2025	17/06/2025

atributos			
Desarrollo de la interfaz gráfica	Patricio - desarrollador	10/06/2025	17/06/2025
Desarrollo de la sintaxis del pseudocódigo	Pardavé - desarrollador	10/06/2025	17/06/2025
Integración de las clases con la GUI	Guzman - desarrollador	18/06/2025	22/06/2025
Realizar pruebas	Segura - SQA	23/06/2025	30/06/2025
Corrección de errores	Todo el equipo de desarrollo	30/06/2025	03/07/2025
Hito 3: Desarrollo			

6. Procesos de Negocio

En el desarrollo del proyecto, se identifican varios procesos importantes que permiten el cumplimiento de los objetivos planteados. Estos procesos están alineados con la estructura del programa a desarrollar y tienen como objetivo garantizar la correcta ejecución y evaluación de los algoritmos mediante la automatización de su análisis. A continuación, se detallan los principales procesos de negocio involucrados:

1. **Recopilación del pseudocódigo**: Este proceso involucra la **entrada del pseudocódigo** de un algoritmo que será analizado por el programa. El pseudocódigo debe seguir una **estructura predefinida** para ser procesado, como bucles, condicionales, etc.

   Flujo del proceso:

   Entrada del pseudocódigo:

   El usuario ingresa el pseudocódigo del algoritmo en una interfaz de texto o lo carga desde un archivo.

   El sistema valida que el pseudocódigo siga una estructura definida (por ejemplo, debe contener bucles for, while, o estructuras condicionales if).

   Validación del pseudocódigo:

El sistema verifica si el pseudocódigo sigue la sintaxis correcta. Si no es válido, se muestra un error y se solicita corrección.

Si es válido, el pseudocódigo se interpreta y el proceso pasa al análisis de la complejidad temporal.

Resultado esperado:

El sistema guarda el pseudocódigo y lo prepara para el análisis posterior.

2. **Análisis de la complejidad temporal**: Una vez que el pseudocódigo es ingresado y validado, el sistema realiza un análisis **teórico** de la complejidad temporal del algoritmo. Este análisis se basa en la estructura del pseudocódigo y las operaciones involucradas, pero no en su ejecución real.

   Flujo del proceso:

   Análisis de la complejidad teórica:

   El sistema evalúa las estructuras de control del pseudocódigo, como bucles, recursión y condiciones.

   El sistema calcula el número de operaciones teóricas en función del tamaño de entrada n, utilizando notaciones asintóticas como **O(n)**, **O(n²)**, **O(log n)**, etc.

   Generación de la función de tiempo T(n):

   A partir del análisis teórico, el sistema genera la **función de tiempo T(n)**, que describe cómo se espera que el tiempo de ejecución crezca conforme aumenta el tamaño de la entrada.

   Se muestra la **notación asintótica** correspondiente (por ejemplo, $O(n^2)$).

   Generación de la gráfica teórica:

   El sistema crea una **gráfica teórica** que representa el crecimiento del tiempo de ejecución en función del tamaño de entrada **n**. Esta gráfica es solo una predicción basada en el análisis estructural del pseudocódigo.

   Resultado esperado:

La **función de tiempo teórica** y la **gráfica teórica** son generadas y presentadas al usuario para que pueda revisar cómo el algoritmo debería comportarse según su complejidad teórica.

3. **Ejecución y análisis empírico**: En este paso, el sistema **ejecuta realmente el algoritmo** con diferentes tamaños de entrada **n**. Este análisis **empírico** ayuda a verificar la precisión de las predicciones teóricas y muestra cómo se comporta el algoritmo en un entorno real.

   Flujo del proceso:

   Ejecución del algoritmo:

   El sistema genera **entradas automáticas** (sin intervención del usuario) para probar el algoritmo. Estas entradas son utilizadas para medir el tiempo real de ejecución.

   Medición del tiempo real de ejecución:

   El sistema mide el tiempo que toma el algoritmo para ejecutar con diferentes tamaños de entrada.

   Los resultados del tiempo de ejecución real se registran y se preparan para ser comparados con las predicciones teóricas.

   Comparación con la complejidad teórica:

   El sistema **compara los tiempos reales** con los tiempos teóricos calculados en el paso 2. Esto permite verificar si las predicciones realizadas con el análisis teórico son precisas.

   Resultado esperado:

   El **tiempo real de ejecución** es registrado y comparado con la función de tiempo teórica.

4. **Generación de la función de tiempo de ejecución**: En este proceso, el sistema genera una representación gráfica más precisa del comportamiento temporal del algoritmo, utilizando tanto la **predicción teórica** como los **resultados empíricos**.

   Flujo del proceso:

   Generación de la función de tiempo de ejecución:

   Basado en los resultados obtenidos en el paso 2 (análisis teórico) y el paso 3 (análisis empírico), el sistema genera la **función de tiempo de ejecución** T(n) para mostrar la comparación entre el tiempo teórico y el tiempo real.

   Generación de la gráfica de comparación:

   El sistema crea una **gráfica de comparación** que muestra tanto la **curva teórica** como la **curva real**. Esta comparación visual permite al usuario entender la diferencia entre el comportamiento esperado del algoritmo y su rendimiento real.

   Resultado esperado:

   La **gráfica de comparación** entre el tiempo teórico y el real se genera y se presenta al usuario.

5. **Comparación de algoritmos predefinidos**: Este proceso se centra en **comparar algoritmos predefinidos**, como **QuickSort**, **MergeSort**, **Bubble Sort**, etc. Los algoritmos ya están integrados en el sistema y el usuario puede seleccionar los que desea comparar.

   Flujo del proceso:

   Selección de algoritmos predefinidos:
   El usuario selecciona los algoritmos que desea comparar desde una lista predefinida de algoritmos disponibles (por ejemplo, QuickSort, MergeSort, etc.).

   Generación de la gráfica teórica:
   Para cada algoritmo, el sistema genera su **gráfica teórica**, mostrando su notación asintótica y la función de tiempo teórica correspondiente.

Ejecución interna de los algoritmos:
El sistema ejecuta cada algoritmo internamente con entradas generadas automáticamente. Los usuarios no interactúan con los datos de entrada.

Generación de la gráfica de tiempo real:

El sistema genera una **gráfica de tiempo real** para cada algoritmo, mostrando cómo el tiempo de ejecución crece con el tamaño de entrada.

Generación de la gráfica de comparación:

El sistema genera una **gráfica de comparación** que superpone tanto las **curvas teóricas** como las **curvas de tiempo real** de los algoritmos seleccionados. Esta visualización ayuda a ver cómo se comportan diferentes algoritmos en términos de eficiencia.

Resultado esperado:

Las **gráficas de comparación** de los algoritmos seleccionados se generan y se presentan al usuario.

6. **Exportación de resultados**: Los resultados obtenidos de los análisis (ya sea del pseudocódigo o de la comparación de algoritmos predefinidos) se pueden **exportar a un archivo PDF**. Este archivo incluirá todas las gráficas, las funciones de tiempo T(n), la notación asintótica, y los tiempos de ejecución real.

   Flujo del proceso:

   Generación del reporte:

   El sistema organiza toda la información relevante en un formato de reporte (PDF).

   Exportación del reporte:
   El usuario puede **descargar o visualizar el reporte** generado que incluye las gráficas y los resultados detallados de su análisis.

   Resultado esperado:

El reporte en formato PDF con los resultados de los análisis es generado y exportado para su revisión.

7. **Evaluación y retroalimentación:** El equipo de desarrollo evalúa los resultados obtenidos y obtiene retroalimentación de los usuarios para mejorar el sistema.

Flujo del proceso:

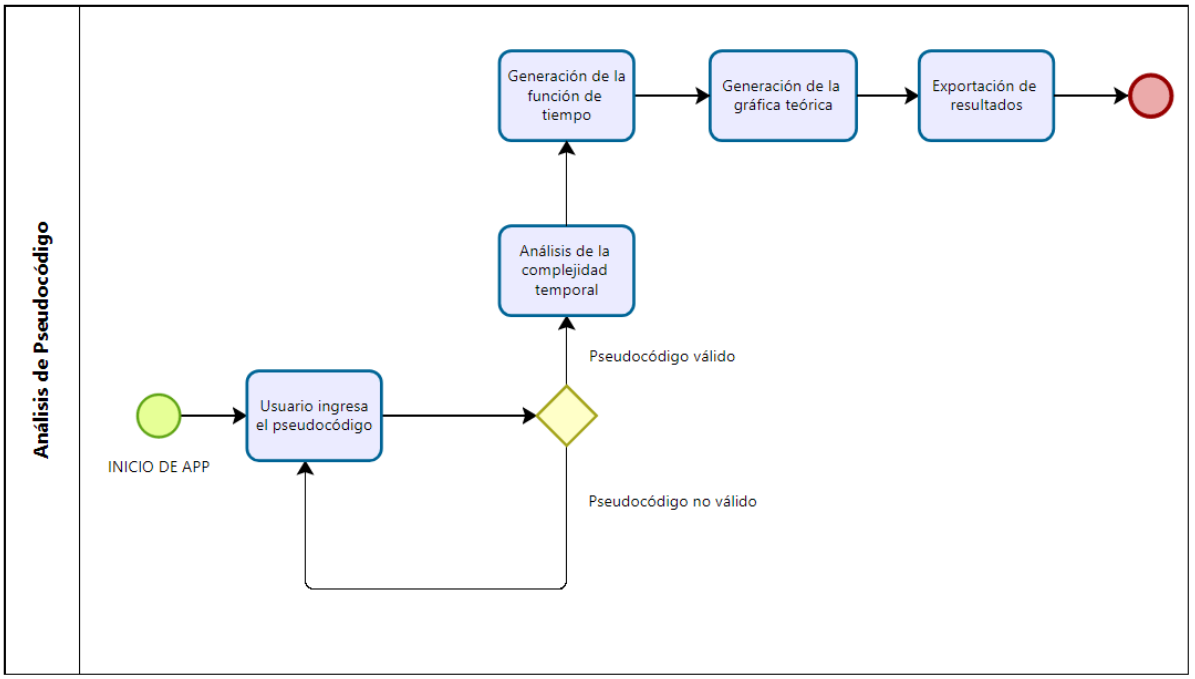
Evaluación de los resultados:
El equipo de desarrollo revisa los resultados del análisis y observa si el sistema está funcionando correctamente.

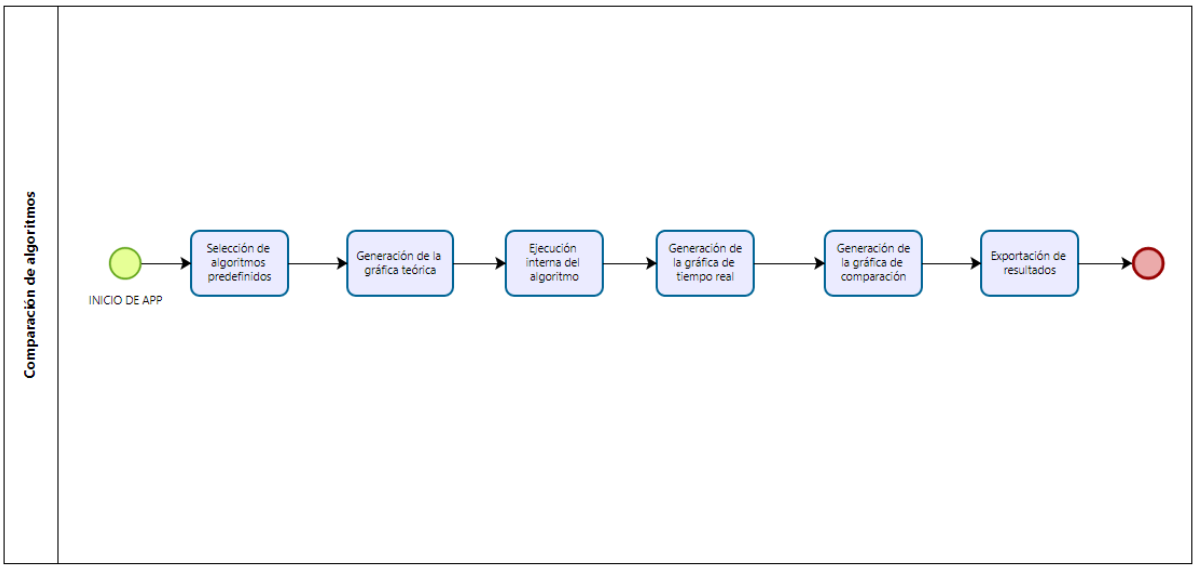
Retroalimentación de los usuarios:
Se obtiene retroalimentación de los usuarios sobre la precisión del análisis, la claridad de las gráficas y la funcionalidad general del sistema.

Resultado esperado:

El sistema es **mejorado continuamente** a partir de la retroalimentación recibida.

Diagramas en bizagi:





A través de estos procesos, el proyecto busca automatizar el análisis de algoritmos, facilitando la comprensión de su complejidad y mejorando la eficiencia en la selección de algoritmos adecuados para la resolución de problemas computacionales.

7. Reglas de Negocio

Código	Regla de negocio
RN-001	El sistema solo permitirá el ingreso de algoritmos que sigan una sintaxis de pseudocódigo estructurada (por ejemplo, bucles para, mientras, condiciones si, etc.) y ya definida.
RN-002	Todo pseudocódigo ingresado por el usuario será interpretado para poder identificar que contenga estructuras que impacten en la complejidad temporal.
RN-003	La función de tiempo T(n) se calculará en base a la cantidad de operaciones teóricas según las estructuras de control y no por su ejecución real.
RN-004	Se estimará la complejidad algorítmica usando notaciones asintóticas estándar vistas en clase.

RN-005	No se permitirá el ingreso de algoritmos que no contengan estructuras repetitivas o de control y que no sigan la sintaxis del pseudocódigo.
RN-006	El sistema tendrá un conjunto limitado de algoritmos clásicos predefinidos (ej. Bubble Sort, QuickSort, Merge Sort, etc.).
RN-007	Para cada algoritmo predefinido se mostrará su función de tiempo teórica, su notación asintótica, y una curva de crecimiento en base al tiempo de ejecución.
RN-008	La ejecución real se realizará internamente con entradas generadas automáticamente por el sistema (el usuario no proporciona entradas).
RN-009	La gráfica de comparación mostrará tanto el crecimiento teórico como el tiempo real.
RN-010	El usuario puede elegir entre dos secciones: análisis de pseudocódigo y comparación de algoritmos clásicos.
RN-011	El usuario podrá exportar el análisis realizado (función T(n), complejidad y gráficas) en un archivo PDF.
RN-012	La información procesada no se almacenará permanentemente, es decir, se tendrá información de manera temporal por cada sesión del usuario.

Donde RN significa Regla de Negocio.

8. Conclusión

El proyecto presentado permite lograr desarrollar una herramienta capaz de analizar y comparar la complejidad temporal de algoritmos a partir de pseudocódigo, facilitando la comprensión y la visualización de su eficiencia. El programa permite identificar automáticamente las estructuras que afectan la complejidad, como bucles y recursividad, y genera funciones de tiempo teóricas, así como gráficas comparativas para distintos algoritmos.

El enfoque del proyecto es automatizar los procesos de análisis de algoritmos promoviendo el aprendizaje y la aplicación práctica de conceptos importantes, permitiendo a los usuarios comparar tanto el desempeño teórico como el real de algoritmos con estructuras repetitivas, condicionales o recursivas. Además, el uso de tecnologías como Java, Swing y herramientas de diseño permiten asegurar una experiencia de usuario intuititva.

Finalmente, el proyecto sienta las bases para futuras mejoras, como la ampliación del soporte a más tipos de algoritmos y estructuras, o la integración de análisis más avanzados.

Referencias

Aho, A. V., & Ullman, J. D. (2007). Parsing Theory. En Foundations of Computer Science (pp. 165-180). Recuperado de https://www.cs.cornell.edu/courses/cs3110/2008fa/recitations/rec16.html

Calvo, J. (2022, December 7). *La complejidad de los algoritmos – Europeanvalley*. Europeanvalley. Retrieved June 4, 2025, from https://europeanvalley.es/noticias/la-complejidad-de-los-algoritmos/

Gutiérrez, J. A., & Rodríguez, F. (2018). Análisis sintáctico de lenguajes de programación. Revista Educación en Ingeniería, 13(25), 81-90. https://revistas.udistrital.edu.co/index.php/reving/article/view/14232/14607

Ponce, J. (2021, February 21). *Algoritmos de búsqueda*. Jahaziel Ponce. Retrieved June 4, 2025, from https://jahazielponce.com/algoritmos-de-busqueda/

Rao, S. S. (2019). Algorithm Optimization Techniques. arXiv preprint arXiv:1902.00103. https://arxiv.org/abs/1902.00103

Sedgewick, R., & Wayne, K. (2021). Algorithms (4ta ed.). Addison-Wesley.

¿Qué es la visualización de datos? Definición, ejemplos y recursos. (n.d.). Tableau. Retrieved June 4, 2025, from https://www.tableau.com/es-mx/learn/articles/data-visualization