

# GVoice - Segunda Entrega

<https://github.com/mora200217/GVoice.git>

Hernández Triana Iván, Morales Martínez Andrés,

Ramírez Montes Juan, Rodríguez Fuentes Miguel

No. Equipo de trabajo: 3

## I. INTRODUCCIÓN

Las soluciones tecnológicas de visualización digital han contribuido con la pedagogía y la consecuente aplicación de conceptos diseños, simulaciones y prototipado de proyectos en las últimas décadas. En las asignaturas como cálculo multivariable, álgebra lineal, entre otras que requieran de un planteamiento geométrico y/o gráfico complejo, resulta de extrema utilidad recurrir a herramientas digitales. No obstante, muchos estudiantes y profesores no dominan por completo el entorno, y en algunos casos, invierten demasiado tiempo en la construcción del caso que quieren ilustrar.

Con el fin de enfocarse mayormente en el concepto a explicar en vez de invertir tiempo en el proceso de elaboración de la gráfica, surge la propuesta de una herramienta de visualización matemática controlada por comandos de voz. Esto permitiría un mayor dinamismo en el proceso de aprendizaje, y una transición más fluida entre planteamiento y resolución de un ejercicio en las aulas.

Se busca usar estructuras de datos para optimizar el procesamiento de instrucciones con sistemas predictivos y relaciones entre sinónimos de instrucciones para diversificar los comandos que se le pueden indicar al sistema; además de almacenar un historial de comandos y agrupar elementos en el espacio.

La descripción de la implementación de estructuras de datos, los requerimientos funcionales del software, la etapa preliminar para la interfaz de usuario, las pruebas del prototipo y análisis comparativo, entre otros, son algunas de las características que serán abordadas a mayor profundidad en el presente documento.

## II. DESCRIPCIÓN DEL PROBLEMA

Herramientas como Wolfram Mathematica, GeoGebra o incluso Matlab, son los entornos predilectos para graficar datos o manipular geometrías. Para simplificar el flujo de trabajo al graficar múltiples escenarios, se ha optado por manejar estructuras genéricas parametrizadas. Esto permite al usuario únicamente ingresar los parámetros de la superficie para expandir o desplazar la geometría sin tener

que expresar por completo la función, agilizando su visualización. Sin embargo, para gráficas más elaboradas se recurre a escribir directamente la expresión, lo cual en ciertas ocasiones no es tan sencillo como se desearía, y en varias ocasiones es necesario el uso de comandos que sean propios de la aplicación.

Buscando un mayor control del espacio visualizado, han surgido herramientas como el Leap Motion o incluso las tabletas digitalizadoras de Wacom que permiten tener un control total del elemento Rotando, trasladando o aumentando el campo de visión. A pesar de su eficiencia, debido a ser una solución de hardware, implica una considerable inversión para adquirir las herramientas, siendo económicamente inviable para estudiantes independientes o instituciones que carezcan de un apoyo financiero sólido para la adquisición de estas herramientas tecnológicas.

Tanto el control del espacio de visión, como las instrucciones de construcción geométrica se pueden abordar desde el desarrollo por comandos de voz. Esto permitiría una mayor versatilidad y eficacia en el uso de la herramienta, ahorrando tiempo tanto de escritura de expresiones como en la visualización de puntos específicos de la gráfica. Así mismo, como valor agregado, el uso de comandos hablados incita el correcto uso del lenguaje matemático para expresar una idea que pueda ser entendida por el sistema de reconocimiento y por los pares del estudiante profesor u otro usuario.

## III. USUARIOS DEL PRODUCTO DE SOFTWARE

GVoice principalmente se enfoca a estudiantes y perfiles académicos que requieran un apoyo gráfico para el entendimiento y visualización de composiciones gráficas y relaciones geométricas. Por lo que estaríamos hablando de una clasificación de usuarios en función de la frecuencia uso del software, encabezado por académicos y estudiantes, en áreas de ingeniería, algunas ciencias puras hasta economía. Seguido por usuarios en los que no sean tan frecuente el uso de herramientas gráficas, como lo son aquellos que quieran incursionar en el estudio autónomo de las

id	Requerimientos funcionales de evaluación	Dirección de Mejora	Factor de Importancia	Característica Voluble						
				Reconocimiento de Voz	Procesador de Texto	Controlador Gráfico	Cola de Renderizado	Historial de Acciones	Sistema de Ejes dinámicos	Manejo Tridimensional
1	Minimizar tiempos de Graficación	↑	3	0	0	3	9	0	1	0
2	Minimizar Tiempos de Cálculos	↑	4	3	3	9	3	0	0	0
3	Escalabilidad de Datos	↑	4	3	9	1	9	3	3	0
4	Facilidad de Implementación	↑	5	0	0	3	3	9	3	0
5	Almacenamiento de datos	↓	1	0	0	1	3	9	0	0
6	Estética de la aplicación	↑	4	3	3	3	1	1	9	9
7	Percepción de Usuario	↑	5	9	1	9	9	9	9	9
	Relevancia en el flujo de trabajo	-	5	0	9	9	9	0	3	0
Puntaje bruto				81	110	167	187	115	126	81
Peso relativo %				9,34%	12,69%	19,26%	21,57%	13,26%	14,53%	9,34%
Orden de importancia				6	5	2	1	4	3	7

Figura 1: Requerimientos funcionales y dirección de mejora

matemáticas, quienes buscan una herramienta de fácil acceso y dinámica que les ayude en su comprensión de las matemáticas a partir de gráficas que ilustren los conceptos.

#### IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Gvoice tendrá una función muy importante la cual está enfocada principalmente en graficar de forma óptima, sencilla y versátil cualquier función que reciba por medio de comandos de voz, pero para llegar al éxito de esta función es necesario plantear un conjunto de requerimientos funcionales con los que buscamos identificar aquellos incisos que son relevantes para el proyecto, los cuales podemos ver en la parte izquierda de la Figura 1. Cada uno de estos requerimientos tendrá una dirección de mejora, que no es más que la forma en que buscamos que el ítem sea afectado, ya sea maximizándolo (↑) o minimizándolo (↓). Por ejemplo, para el id 1, necesitamos que sea maximizada la “minimización de los tiempos de graficación”, por otro lado, se busca que el “almacenamiento de datos” sea minimizado.

Dentro de la misma figura encontramos el factor de importancia (Relevancia del requerimiento funcional para el proyecto) y un conjunto de características volubles (particularidades que como diseñadores podremos modificar para que dependiendo de su implementación mejore el requerimiento funcional específico), las cuales van a tener una asignación

numérica como se muestra en la siguiente tabla:

	Ponderación
Factor de importancia *	1 - 5
Característica voluble †	0,1,3,9

Tabla 1: Ponderaciones

Cada uno de los requerimientos funcionales evaluados se le relacionará por medio de la ponderación anterior una influencia de las diferentes características volubles sobre él; por ejemplo, el “minimizar tiempos de graficación” que como dijimos anteriormente, se hace necesaria una maximización, la característica voluble que mayor influencia tendrá para llegar a los tiempos requeridos será por medio de la cola de renderizado, en donde veremos la primera implementación de las estructuras de datos. Las demás características volubles se les asignará un valor de acuerdo a su influencia para cumplir la dirección de mejora.

Una vez asignadas todas las ponderaciones correspondientes obtenemos un puntaje bruto, el cual es una relación entre la característica voluble y el factor de importancia, por cada requerimiento. Con estos datos, hallamos un peso relativo, esto nos ayuda a ver cuál sería el orden de importancia de cada una de las

\*Siendo 1 la importancia más baja y 5 la más alta del requerimiento funcional para el proyecto.

†Entre mayor ponderación, mayor influencia sobre el requerimiento funcional, de acuerdo a la dirección de mejora (sea maximizándola o minimizándola).

características tratadas para priorizar tareas en cada entrega. En base a lo anterior, las tres características en las cuales se iba a trabajar en esta entrega son el controlador gráfico, la cola de renderizado y el sistema de ejes dinámicos.

#### A. Cola de renderizado:

- **Descripción funcionalidad:**

Esta parte del proyecto consiste en un programa que se encarga de recibir todas las gráficas solicitadas en cada momento para posterior a ello ir renderizando cada una de las mismas, después de haberla mostrado en pantalla, encola dicha gráfica y continúa con la siguiente que esté en la cabeza de la cola, esto garantiza que todas las gráficas necesarias se estarán renderizando al menos una vez por cada ciclo, garantizando que ninguna de ellas se pierda.

- **Acciones iniciadoras y comportamiento esperado:**

La cola de renderizado se encargará de mostrar las gráficas en pantalla, por lo cual se espera que acorde a la función en cuestión, represente en pantalla la serie de puntos. Como entrada, va a contar con el requerimiento desde el Controlador Gráfico, el cual le da las instrucciones de qué puntos poner en pantalla para cada gráfica solicitada por el usuario, es por esta razón que los posibles errores que pudiesen surgir por una entrada de información errónea son considerablemente menores al garantizar el retorno de información de la clase de controlador gráfico.

#### B. Controlador gráfico:

- **Descripción funcionalidad:**

Como su nombre lo indica esta sección es la encargada de indicarle al programa que será aquello que debe encontrarse en la pantalla durante la ejecución, respaldado por la cola de renderizado (explicado a continuación) es la parte fundamental para el proyecto. Por lo que la función principal del controlador gráfico es comunicarle al software que va a estar o que no va a estar en dicho momento en la interfaz gráfica.

- **Acciones iniciadoras y comportamiento esperado:**

En forma general el controlador gráfico va a depender de aquello que el usuario quiera graficar, ya que empezara un flujo empezando por

la cola de renderizado anteriormente mencionado, que luego por medio del controlador gráfico, seremos capaces de mostrar lo que el usuario necesita.

#### C. Sistema de ejes dinámicos:

- **Descripción funcionalidad:**

Como parte de una mayor interacción con el usuario, un sistema de ejes dinámicos les proporciona a los usuarios una diferente perspectiva para cada una de las funciones que va a graficar, ya que todas estas gráficas van a estar sujetas espacialmente respecto al eje que se presenta en la interfaz gráfica. Este dinamismo se realiza gracias a una pila que guarda la posición del mouse una vez esta va interactuando con el software.

- **Acciones iniciadoras y comportamiento esperado:**

Por medio del mouse el programa captura la información de la ubicación del mismo mientras se mantenga presionado el clic derecho, lo cual se convierte en una traslación en los ejes a través de la pantalla, a su vez, las gráficas se mueven junto con los ejes, permitiendo mostrar información que no se pueda ver en la pantalla en un inicio para mayor comodidad del usuario.

## V. INTERFAZ DE USUARIO

Para la entrega anterior se planteó una posible implementación para la interfaz de usuario, la cual fue la base para realizar la interfaz correspondiente para la segunda entrega. Las siguientes figuras corresponden primero a al borrador planteado y segundo a la primera implementación final:

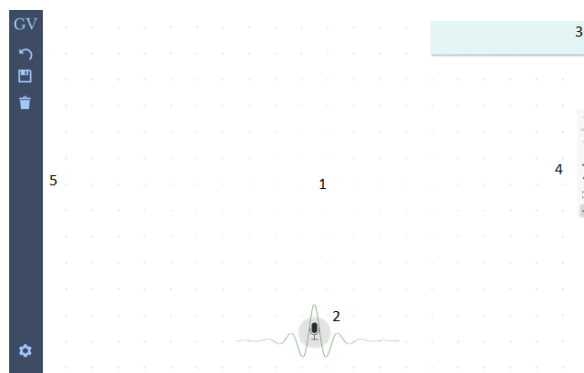


Figura 2: Propuesta Interfaz de usuario



Figura 3: Implementación Interfaz de usuario

Por ser una primera etapa de implementación se tuvieron en cuenta solamente los botones de guardado, eliminación, y el botón para añadir mayor cantidad de gráficas, encontrado en el avance anterior. La descripción de cada uno de estos módulos se encuentra a continuación:

#### 1) *Entorno de graficación:*

Espacio en el cual se van a mostrar todas las funciones requeridas y en donde cualquier interacción con las gráficas se llevará a cabo.

#### 2) *Módulo de ajuste del entorno:*

En este módulo encontramos los botones necesarios para guardar el archivo, eliminar todas las gráficas, las cuales corresponden a las funcionalidades específicas para esta entrega.

#### 3) *Botón añadir gráfica*

Este botón se encuentra en la parte inferior derecha de la interfaz, y cumple el objetivo de añadir de forma aleatoria una función con el añadido que situara de forma automática los puntos críticos de la función como se muestra a continuación.

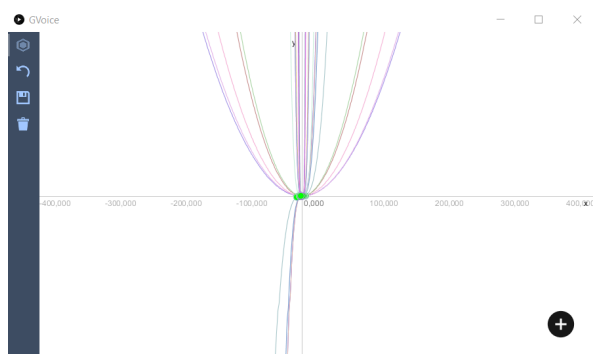


Figura 4: Interfaz de usuario con funciones de prueba y sus respectivos puntos críticos

#### 4) *Ventana emergente - Calculadora*

Se tiene una ventana emergente la cual cuenta con una calculadora, buscando ayudar al usuario en cualquier cálculo rápido que necesite como parte de la creación de sus funciones. Esta se encuentra todavía en una etapa de desarrollo, se ha implementado tanto el árbol de expresión, el cual se va a encargar de jerarquizar las operaciones para su posterior gestión, así como la interfaz gráfica que se muestra a continuación.

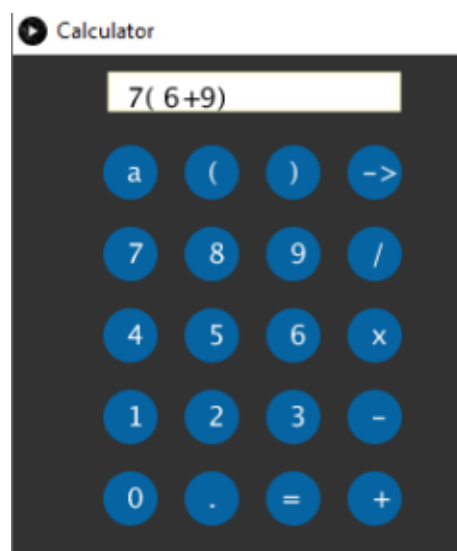


Figura 5: Calculadora

### VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El software será desarrollado en el entorno de desarrollo integrado Processing, el cual está basado en Java, la elección se debe a que la mayoría de los integrantes del equipo cuentan con conocimientos de este entorno, además que al estar basado en Java las estructuras de datos se implementan igual que en este, otra ventaja sustancial y la principal razón por la cual no se hizo el desarrollo directamente en Java es porque Processing facilita el prototipado de aplicaciones al estar enfocado en resultados visuales prácticamente inmediatos, facilitando en gran medida el desarrollo de la interfaz base para que de esta manera el equipo se pueda enfocar en las estructuras y su implementación en un prototipo funcional.

La aplicación en principio aprovechará las virtudes de la máquina virtual de Java para poder ser ejecutada en todos los entornos de escritorio y en sistemas operativos como Linux, Mac y Windows, para trabajo futuro si la aplicación tiene buena acogida se pensaría en el desarrollo de una versión para dispositivos móviles.

## VII. SEGUNDO PROTOTIPO DE SOFTWARE

Para el segundo prototipo se tenía como objetivo principal el tener una interfaz gráfica más llamativa, con algunas funcionalidades desde un menú, como se presentó anteriormente, y finalmente para las funciones que ya eran posibles graficar obtener los puntos críticos, para los cuales se utilizaron arboles como estructuras de datos, o más específicamente los heaps.

Como se mencionó anteriormente, se obtuvieron los puntos críticos a partir de una estructura de datos no lineal (Minheap), la cual se organizaba de acuerdo a la coordenada Y de un conjunto de Pvector obtenidos de la derivada de la función.

Por otro lado, la interfaz gráfica fue modificada de tal manera que presentara un menú en el lado izquierdo de la ventana, la que en el momento contará con tres botones funcionales, que serán eliminar, que eliminara las funciones se encuentren en la interfaz, rehacer, que permite borrar la última gráfica agregada y finalmente la funcionalidad de guardar, que genera una imagen formato .JPG de la función presentada en la interfaz, como la que se presenta a continuación:

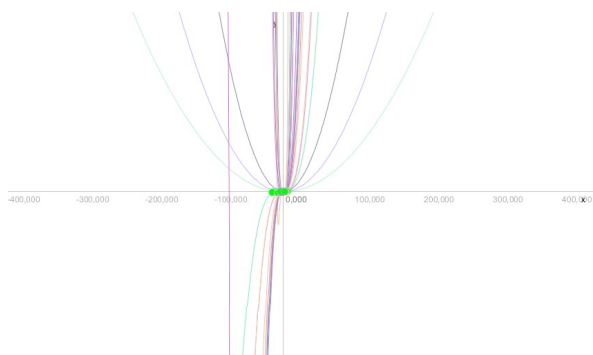


Figura 6: Ejemplo imagen obtenida con botón de guardado

## VIII. IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Para este proyecto se desarrollaron las diversas implementaciones de estructuras lineales y fueron agregadas en una librería destinada para este fin, a partir de esto, cuando se requiriera alguna de las estructuras para implementar alguna funcionalidad simplemente era llamada desde esta librería.

Este desarrollo modular facilita de manera considerable la implementación de las pruebas, a su vez que es más fácil encontrar errores y corregirlos a tiempo.

La cola fue implementada tanto por medio de referencias como por medio de arreglos. A la cola

implementada por referencias se le agregó puntero en la cola con el fin de que la inserción de datos fuese de complejidad  $O(1)$ .

Esta fue utilizada para implementar la cola de renderizado descrita anteriormente, ya que era necesario tener la propiedad de que la primera función en ser encolada fuese la primera en salir. En la siguiente imagen tenemos uno de los usos que se le dio a la cola:

```
for (int j = 0; j < inScreen.numInside(); j++) {
    memoria= inScreen.dequeue();
    this.savePoints(memoria);
    // Guarda los puntos como imagen
    inScreen.enqueue(memoria);
}
```

Figura 7: Implementación de la cola

La pila fue implementada por medio de referencias y arreglos, se decidió usar la implementación por arreglos para la funcionalidad del sistema de ejes dinámicos (y en general todo el plano), dado que la implementación por referencias ocupaba espacio en memoria que no representaba una ventaja significativa para esta funcionalidad. En la siguiente imagen tenemos uno de los usos que se le dio a la pila:

```
// Mouse drag
if ( this.mouseDragged() ) {
    dragPositions.push(new PVector(mouseX, mouseY));
    this.hasToGenerate = true;
} else if (!dragPositions.empty()
() && dragPositions.numInside < 2) {
    dragPositions.pop();
}
```

Figura 8: Implementación de la pila 1

```
pgf.beginDraw();

while (!graphsArray.empty())
    pgf.image(graphsArray.pop(), 0, 0);
pgf.endDraw();

this.imgToShow = pgf;
```

Figura 9: Implementación de la pila 2

Las listas enlazadas se implementaron para correr de manera más óptima las pruebas con las diferentes estructuras de datos en el programa, dado que no conocíamos la cantidad de datos iniciales a recolectar, era indispensable contar con una estructura de tamaño variable durante el tiempo de ejecución; este problema

podía ser solventado con un arreglo dinámico pero la cantidad de tiempo que tardaría moviendo los datos de un lugar a otro, adicional a la cantidad de memoria sin utilizar al realizar pruebas con valores muy grandes, provocaría un gasto innecesario de procesamiento y espacio. En la siguiente imagen tenemos uno de los usos que se les dio a las listas:

```
public void beginSample() {
    this.countTime = System.nanoTime();
    this.countSize = System.nanoTime();
}

public void endSample() {
    this.step(System.nanoTime() - this.countTime);
}

private void step(long time) {
    println(time / 1000000);
    data.pushRear(new Record((int) time / 1000000));
}
```

Figura 10: Implementación de las listas

Otra estructura utilizada en el desarrollo de esta segunda etapa del proyecto fue el árbol de expresión, el cual contaba con el objetivo de organizar de manera jerárquica cada operación con el fin de obtener el resultado de las operaciones de una manera rápida en la calculadora que se encuentra en etapa de desarrollo.

```
void calc(Node root) {
    Node p = root;
    while (p.childs[0].childs[0] != null) {
        p = p.childs[0];
    }

    if (p.val == "+") {
        p.val = String.valueOf(Integer.parseInt(p.childs[0].val) + Integer.parseInt(p.childs[1].val));
    } else if (p.val == "*") {
        p.val = String.valueOf(Integer.parseInt(p.childs[0].val) * Integer.parseInt(p.childs[1].val));
    } else if (p.val == "/") {
        p.val = String.valueOf(Integer.parseInt(p.childs[0].val) / Integer.parseInt(p.childs[1].val));
    } else if (p.val == "-") {
        p.val = String.valueOf(Integer.parseInt(p.childs[0].val) - Integer.parseInt(p.childs[1].val));
    }

    p.childs[0] = null;
    p.childs[1] = null;

    this.printTree();
    println("");

    if (root.childs[0] != null)
        this.calc(root);
}
}
```

Figura 11: Implementación de un árbol de expresión

## IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Para las pruebas de prototipo se utilizaron y compararon diferentes estructuras de datos, con el objetivo de conocer cuál era la más conveniente para ser parte de la cola de renderizado. El objeto que será

almacenado en las estructuras de datos corresponderá a las gráficas compuestas por un conjunto de puntos. En un principio se tenía planeado realizar el análisis con la cantidad de puntos que estábamos utilizando para cada una de las gráficas, pero esto no fue posible ya que lo que realmente se estaba graficando correspondía a un paquete de 700 puntos agrupados en cada función.

tiempo de graficación como especificamos a continuación.

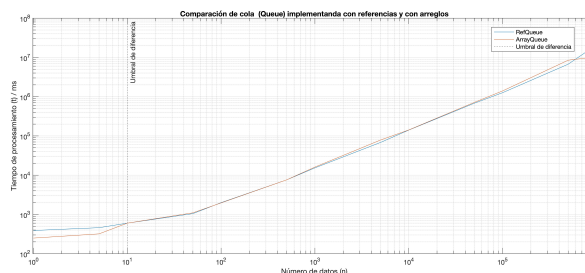


Figura 12: Comparación de cola implementada con referencias y con arreglos

cómo se evidencia en las gráfica anterior la complejidad sigue una tendencia lineal lo cual con cuerda con el análisis realizado de Big O el cual concluimos que es  $O(n)$ , aparte de ello encontramos que no se es posible hallar cuál de las dos implementaciones es mejor utilizar para este proyecto, por lo que entraría en juego la cantidad de memoria utilizada por cada una de ellas, en la cual tiene mayor ventaja una cola con arreglos.

Para esta segunda entrega se implementaron otras estructuras de datos adicionales, para la funcionalidad de máximos y mínimos se hizo uso de Heaps, hallando de esta forma los puntos críticos a partir de sus derivadas. La comparación de esta funcionalidad se realizó con la estructura de las listas ordenadas y este fue el resultado para los dos casos.

Para la segunda entrega, se procedió a reconocer los puntos críticos (PC) dentro de cada gráfica desplegada. Debido a que la funcionalidad se ejecuta acorde a la decisión del usuario, el procesamiento de los PC no requiere se invocado en la construcción de la función, ahorrando lapsos dentro del renderizado inicial. Únicamente, cuando el usuario requiera de los puntos críticos, se procede con su extracción.

Internamente, una gráfica se almacena como un array de pares ordenados continuos, por lo que el proceso de identificación máximos y mínimos requeriría de un recorrido completo de los datos por gráfica. Con el fin de optimizar los tiempos de búsqueda dentro de los conjuntos de datos de cada gráfica, se implementó un Min-Heap que mantiene una complejidad logarítmica en la extracción de mínimos principales. Usando el criterio de la primera derivada, se construye un min



heap con las magnitudes de las derivadas evaluadas en cada punto de la función. Al extraer los puntos mínimos que no superen un umbral propio del delta usado, reconocemos los puntos críticos de la función. Aunque la extracción del valor mínimo es  $O(1)$  el rebalanceo del min heap requiere una complejidad logarítmica.

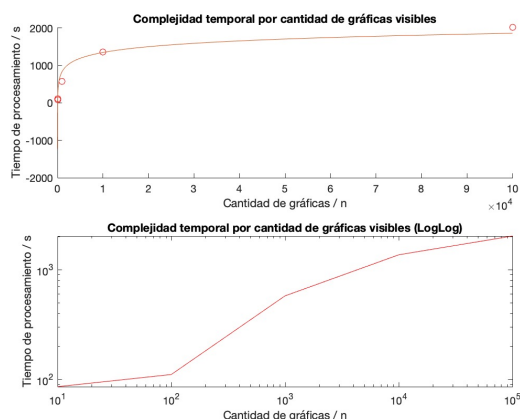


Figura 13: Renderizado puntos críticos Heaps

Aquí se puede apreciar el comportamiento logarítmico, por lo que se puede concluir que el uso de Heaps para la ubicación de máximos y mínimos resulta una solución útil. Por otro lado, dado el enfoque del proyecto no es posible realizar pruebas con más de cierta cantidad de gráficas al tiempo ya que el procesamiento gráfico es demasiado alto lo cual nos lleva a un límite de gráficas posibles de procesar en cierto tiempo.

Replicamos el proceso, pero sustituyendo los minheaps con listas ordenadas. Se mantiene la misma lógica de extraer los valores mínimos de las derivadas de cada función. El propósito de la lista ordenada es permitir una extracción constante, de los primeros valores de la lista. No obstante, el proceso de ordenamiento de la estructura aumenta la complejidad a una lineal  $O(n)$  con respecto a la logarítmica  $O(\log(n))$

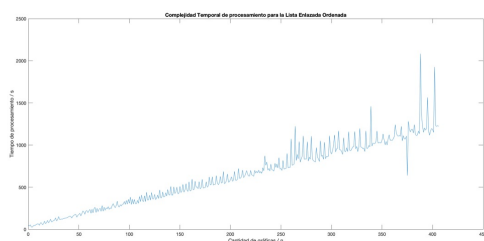


Figura 14: Renderizado puntos críticos con lista ordenada

Como se puede ver, conforme va aumentando el número de gráficas, su tendencia se vuelve inestable, esto se debe a el alto requerimiento respecto al componente de procesamiento que tuvo que llevar a cabo

el computador, por lo que si bien sigue un comportamiento lineal  $O(n)$ , es necesario tener cuidado para las grandes cantidades de datos

## X. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS
Andres Mauricio Morales Martinez	Experto	Finalización de la interfaz gráfica
	Observador	Presentación del proyecto
Ivan Mauricio Hernandez Triana	Lider	Implementar botones del menú
	Investigador	Pruebas de rendimiento
Juan Fernando Ramirez Montes	Animador	Presentación del proyecto
	Coordinador	Implementar funciones para calculadora
Miguel Angel Rodriguez Fuentes	Técnico	Interfaz para la calculadora
	Secretario	Pruebas de rendimiento
		Implementar las funciones para la derivada
		Dirección del informe
		Implementar estructura del Heap

Figura 15: Roles y actividades

## XI. DIFICULTADES Y LECCIONES APRENDIDAS

### A. Dificultades:

A partir del desarrollo de la aplicación se encontraron las siguientes dificultades:

- 1) En el momento de querer graficar los puntos críticos de las funciones, en un principio aparecían en lugares incorrectos y no permanecían de forma constante graficados en la interfaz, a pesar de tener las coordenadas adecuadas, por lo que se pensó que era un posible problema de la implementación del Heap.
- 2) La implementación correspondiente para las operaciones del árbol binario, tuvieron diferentes complicaciones en el momento de organizarlo de tal manera que las operaciones se realizan según la precedencia de signos La implementación.

### B. Lecciones:

A partir del desarrollo de la aplicación se aprendimos las siguientes lecciones:

- 1) Las estructuras de datos no lineales, aunque poseen mayor elaboración en su implementación, la complejidad en búsqueda es menor que las estructuras de datos lineales, por lo que para este tipo de proyectos es de gran ayuda.
- 2) Dado la naturaleza del proyecto, nos dimos cuenta de que 100.000 gráficas era el máximo límite de funciones en la interfaz, esto debido a las limitaciones del entorno de desarrollo y al mismo hardware en donde se ejecute el proyecto.

## REFERENCIAS

- [1] Mit Introduction to Programming in Java Course,  
[urlhttps://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/index.htm](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/index.htm)
- [2] Processing documentation, [urlhttps://processing.org/reference/](https://processing.org/reference/)
- [3] Nature of Code - Daniel Shiffman,  
[urlhttps://natureofcode.com/book/](https://natureofcode.com/book/)