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

Entregable N°02 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

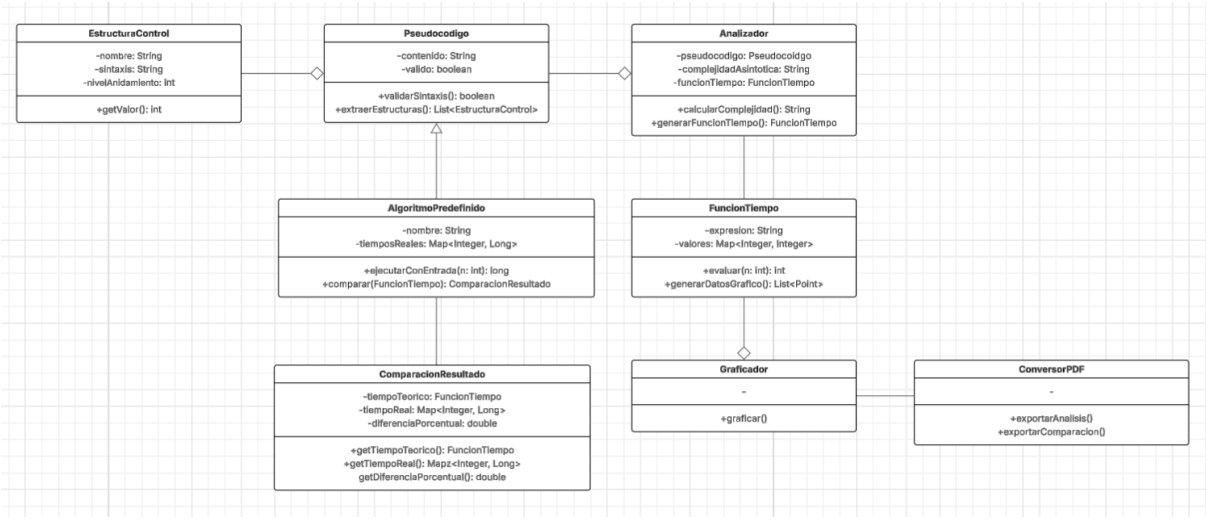
Lima, Perú

2025

Índice

1. Diagrama de clases

A continuación se presenta la primera versión del diagrama de clases que se usará para el presente proyecto. El diagrama de clases involucra a todos las clases y métodos que se van a utilizar durante el desarrollo del programa.



2. Diccionario de clases

El diccionario de clases nos permite describir de manera detallada los atributos y métodos de cada clase en el desarrollo de un sistema, teniendo en cuenta que forma parte de la documentación del diseño orientado a objetos. Una vez que hayamos definido el diagrama de clases, se ha desarrollado el diccionario de clases para describir justamente el diagrama de clases realizado.

Clase: EstructuraControl					
Atributo	**Tipo**	**Visibilidad**	**Descripción**		
nombre	String	private	Es el nombre de la estructura de control (if, for, while)		
sintaxis	String	private	Guarda la forma en la que está escrita la estructura en el código original.		
nivelAnidamiento	int	private	Indica cuantás veces está anidada		
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
getValor	public	-	-	int	Retorna un valor numérico

Clase: Pseudocodigo					
Atributo	**Tipo**	**Visibilidad**	**Descripción**		
contenido	String	private	Contiene el pseudocodigo en forma de texto		
valido	boolean	private	Indica si el pseudicodigo cumple con la sintaxis válida		
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
validarSintaxis	public	-	-	boolean	Revisa si el contenido del pseudocódigo tiene una estructura sintáctica correcta
extraerEstructuras	public	-	-	List<EstructuraControl>	Devuelve una lista de las estructuras de control que contiene.

Clase: FuncionTiempo					
Atributo	**Tipo**	**Visibilidad**	**Descripción**		
expresion	String	private	Es una cadena de texto que representa la función de tiempo del algoritmo		
valores	Map <Integer, Integer>	private	Lista o colección de valores numéricos que representan resultados de evaluar la función para distintos tamaños de entrada		
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
evaluar	public	int	Recibe un entero que es el tamaño de la entrada	int	Resultado de evaluar la expresión con ese número de entrada
generarDatosGrafico	public	-	-	List<Point>	Genera una lista de puntos para construir un gráfico

Clase: ComparacionResultado			
Atributo	**Tipo**	**Visibilidad**	**Descripción**
tiempoTeorico	FuncionTiempo	private	Es una lista o conjunto de valores calculados a partir de la función de tiempo teórica
tiemposReal	Map <Integer, Long>	private	Son los valores medidos realmente al ejecutar el algoritmo con distintos tamaños de entrada (n).
diferenciaPorcentual	double	private	Es el porcentaje de diferencia entre los tiempos teóricos y los reales.

Clase: AlgoritmoPredefinido					
Atributo	**Tipo**	**Visibilidad**	**Descripción**		
nombre	String	private	Es el nombre del algoritmo predefinido		
tiemposReales	Map <Integer, Long>	private	Es una lista de tiempos obtenidos al ejecutar el algoritmo con diferentes entradas.		
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
ejecutarConEntrada	public	int	Recibe un entero que es el tamaño de la entrada	Long	Devuelve el tiempo que tardó en ejecutarse
comparar	public	FuncionTiempo	Compara los tiempos reales del algoritmo con los tiempos teóricos calculados por una instancia de FuncionTiempo	ComparaciónResultado	Devuelve un objeto ComparacionResultado con las diferencias.

Clase: Analizador					
Atributo	**Tipo**	**Visibilidad**	**Descripción**		
pseudocodigo	Pseudocodigo	private	Contiene el algoritmo ya transformado desde su lenguaje original.		
complejidadAsintotica	String	private	Es una cadena de texto que representa la notación Big-O del algoritmo		
funcionTiempo	FuncionTiempo	private	Representa la función matemática que estima el tiempo de ejecución del algoritmo en función del tamaño de entrada.		
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
calcularComplejidad	public	-	-	String	Devuelve una cadena con esa notación Big-O
generarFuncionTiempo	public	-	-	FuncionTiempo	Crea una instancia de la clase FuncionTiempo

Clase: Graficador					
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
graficar	public	-	-	void	-

Clase: ConversorPDF					
Método	**Visibilidad**	**Parámetro**		**Retorna**	
		Tipo	**Descripción**	**Tipo**	**Descripción**
exportarAnalisis	public	-	-	void	
exportarComparacion	public	-	-	void	

3. Pseudocódigo de las clases

A continuación se presenta el pseudocódigo de las clases, métodos y atributos definidos en el diagrama de clases, siguiendo la estructura del pseudocódigo visto en clase. Este pseudocódigo nos permite tener un entendimiento mayor con respecto a la descripción de estos componentes del diagrama de clases descritos en la sección de “Diccionario de clases”.

Clase `EstructuraControl`

```
Clase EstructuraControl
    Atributos:
        sintaxis: String
        nombre: String
        nivelAnidamiento: Entero

    Constructor EstructuraControl(str: String, n: Entero)
        nombre ← str
        nivelAnidamiento ← n

    Métodos:
        getNivelAnidamiento(): Entero
            Retornar nivelAnidamiento

        getSintaxis(): String
            Retornar sintaxis

        setSintaxis(sintaxis: String)
            this.sintaxis ← sintaxis

        getNombre(): String
```

Retornar nombre

FinClase

Clase `Pseudocodigo`

Clase Pseudocodigo

Atributos:

contenido: String

valido: Booleano

Constructor Pseudocodigo(texto: String)

contenido ← texto

valido ← Falso

Métodos:

validarSintaxis(): Booleano

// Analizar sintaxis del contenido

// Si es válida, valido ← Verdadero

Retornar valido

extraerEstructuras(): Lista<EstructuraControl>

lista ← nueva Lista vacía

// Extraer estructuras de control (for, while, if, etc.)

Retornar lista

FinClase

Clase AlgoritmoPredefinido extiende Pseudocodigo

Atributos:

nombre: String

tiemposReales: Mapa<Integer, Long>

Constructor AlgoritmoPredefinido(nombre: String, texto: String)

Llamar al constructor de Pseudocodigo con texto

this.nombre ← nombre

Métodos:

ejecutarConEntrada(n: Entero): Long

// Por implementar

Retornar null

comparar(f: FuncionTiempo): ComparacionResultado

// Por implementar

Retornar null

FinClase

Clase `Analizador`

Clase Analizador

Atributos:

pseudo: Pseudocodigo

complejidadAsintotica: String

funcionTiempo: FuncionTiempo

Métodos:

calcularComplejidad(): String

complejidadAsintotica ← ""

// Por implementar

Retornar complejidadAsintotica

generarFuncionTiempo(): FuncionTiempo

// Por implementar

Retornar null

FinClase

Clase `FuncionTiempo`

Clase FuncionTiempo

Atributos:

expresion: String

valores: Mapa<Integer, Integer>

Constructor FuncionTiempo(str: String)

expresion ← str

Métodos:

evaluar(n: Entero): Entero

// Por implementar

Retornar n

```
    generarDatosGrafico(): Lista<Point>
        lista ← nueva Lista vacía
        // Por implementar
        Retornar lista

    getExpresion(): String
        Retornar expresion

    setExpresion(expresion: String)
        this.expresion ← expresion

    getValores(): Mapa<Integer, Integer>
        Retornar valores

    setValores(valores: Mapa<Integer, Integer>)
        this.valores ← valores
FinClase
```

Clase `ComparacionResultado`

```
Clase ComparacionResultado
  Atributos:
    tiempoTeorico: FuncionTiempo
    tiempoReal: Mapa<Integer, Long>
    diferenciaPorcentual: Real
  Métodos:
```

getTiempoTeorico(): FuncionTiempo

Retornar tiempoTeorico

setTiempoTeorico(tiempoTeorico: FuncionTiempo)

this.tiempoTeorico ← tiempoTeorico

getTiempoReal(): Mapa<Integer, Long>

Retornar tiempoReal

setTiempoReal(tiempoReal: Mapa<Integer, Long>)

this.tiempoReal ← tiempoReal

getDiferenciaPorcentual(): Real

Retornar diferenciaPorcentual

setDiferenciaPorcentual(diferenciaPorcentual: Real)

this.diferenciaPorcentual ← diferenciaPorcentual

FinClase

Clase `Graficador`

Clase Graficador

Métodos:

graficar()

// Por implementar: generar visualización de datos

FinClase

Clase `ConversorPDF`

Clase ConversorPDF

Métodos:

exportarAnalisis()

// Por implementar: exportar análisis en formato PDF

exportarComparacion()

// Por implementar: exportar comparación en formato PDF

FinClase

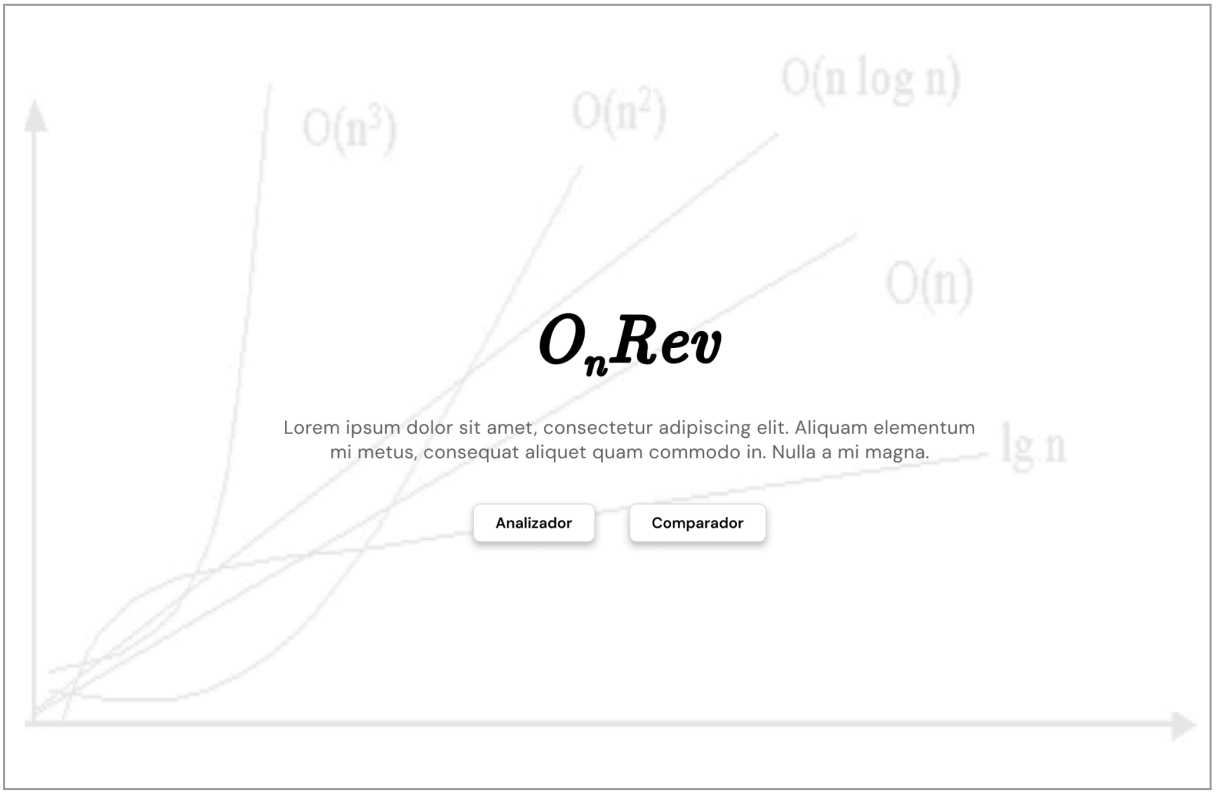
4. Diseño de la arquitectura gráfica

La arquitectura gráfica nos permite visualizar la organización y la conexión de todas las pantallas (interfaces de usuario) correspondientes al flujo de interacción del programa. Además, se tiene un diagrama, el cuál está representado mediante la conexión de estas pantallas para la interacción del usuario dentro del documento de diseño realizado en la herramienta de Figma. Se tienen las siguientes vistas del programa.

Enlace al documento de diseño de las vistas en Figma:
https://www.figma.com/design/EMrCWBtdgTQ2gstEvZn9LU/Design-OnRev?node-id=0-1&t=aLPZ0tlbG9YsXCNb-1

Esquema general de interacción

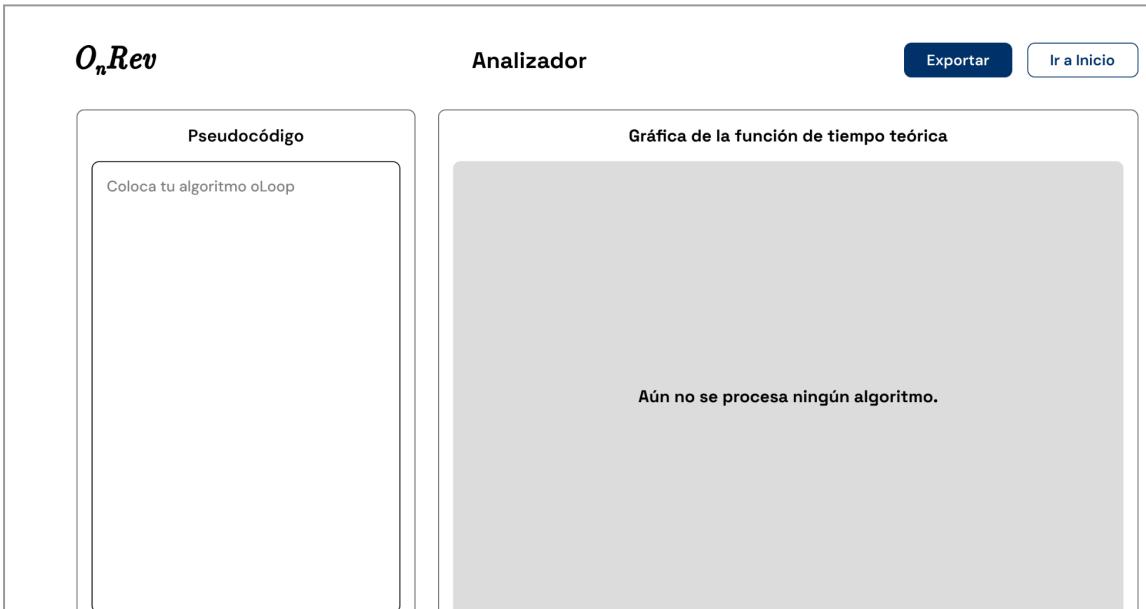
Vistas



O_nRev

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam elementum mi metus, consequat aliquet quam commodo in. Nulla a mi magna.

Analizador | Comparador



O_nRev

Analizador

Exportar | Ir a Inicio

Pseudocódigo

```
Clase CTorres viene-de CObjeto
 Atributos
  nDiscos
 Métodos
  Hanoi(nDiscos, Origen, Destino, Auxiliar)
fClase

Metodo CTorres.Hanoi (nDiscos, Origen, D
 Si (nDiscos = 1) entonces
  Escribir("Mover disco Origen a Destino")
 sino
  Torres.Hanoi(nDiscos -1, Origen, Auxiliar,
  Escribir("Mover disco Origen a Destino")
  Torres.Hanoi(nDiscos – 1,Auxiliar, Destinc
 fSi
FMétodo
```

Limpiar

Verificar

Gráfica de la función de tiempo teórica

Aún no se procesa ningún algoritmo.

Complejidad Algorítmica

ns

Función de tiempo

ns

O_nRev

Analizador

Exportar | Ir a Inicio

Pseudocódigo

```
Clase CTorres viene-de CObjeto
 Atributos
  nDiscos
 Métodos
  Hanoi(nDiscos, Origen, Destino, Auxiliar)
fClase

Metodo CTorres.Hanoi (nDiscos, Origen, D
 Si (nDiscos = 1) entonces
  Escribir("Mover disco Origen a Destino")
 sino
  Torres.Hanoi(nDiscos -1, Origen, Auxiliar,
  Escribir("Mover disco Origen a Destino")
  Torres.Hanoi(nDiscos – 1,Auxiliar, Destinc
 fSi
FMétodo
```

Limpiar

Procesar

Gráfica de la función de tiempo teórica

Algoritmo listo para procesar.

Complejidad Algorítmica

ns

Función de tiempo

ns

O_nRev

Analizador

Exportar | Ir a Inicio

Pseudocódigo

```
Clase CTorres viene-de CObjeto
 Atributos
  nDiscos
 Métodos
  Hanoi(nDiscos, Origen, Destino, Auxiliar)
fClase

Metodo CTorres.Hanoi (nDiscos, Origen, D
 Si (nDiscos = 1) entonces
  Escribir("Mover disco Origen a Destino")
 sino
  Torres.Hanoi(nDiscos -1, Origen, Auxiliar,
  Escribir("Mover disco Origen a Destino")
  Torres.Hanoi(nDiscos – 1,Auxiliar, Destinc
 fSi
FMétodo
```

Limpiar

Verificar

Gráfica de la función de tiempo teórica

Complejidad Algorítmica

$O(n)$

Función de tiempo

$T(n) = \Theta(n \log n)$

O_nRev

Comparador

Exportar | Ir a Inicio

Algoritmos predefinidos

Seleccionar algoritmo

Se mostrará información referente al algoritmo que se eliga.

Comparar

Gráfica de la función de tiempo teórica

Función de tiempo teorica

Función de tiempo real

Complejidad Algorítmica

ns

Función de tiempo

ns

Tiempo de ejecución

ns

$O_n Rev$

Comparador

Exportar | Ir a Inicio

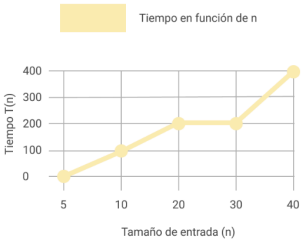
Algoritmos predefinidos

QuickSort

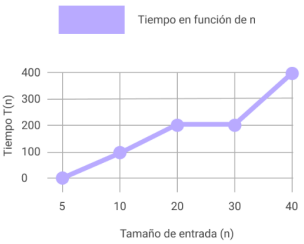
Características del algoritmo
- Tipo de algoritmo: Divide y vencerás.
- Técnica principal: Partición del arreglo con respecto a un pivote.
- Es recursivo dado que se llama a sí mismo en subarreglos menores.
- No es estable dado que no garantiza la conservación del orden de elementos iguales.
- Es muy eficiente para grandes volúmenes de datos.

Comparar

Gráfica de la función de tiempo teórica



Gráfica de la función de tiempo real



Complejidad Algorítmica

$O(n \log n)$

Función de tiempo

$T(n) = \Theta(n \log n)$

Tiempo de ejecución

00:02:04

Reportes

Reporte del análisis de algoritmo oLoop

Complejidad Algorítmica $O(n)$

Función de tiempo $T(n) = \Theta(n \log n)$

Pseudocódigo ingresado

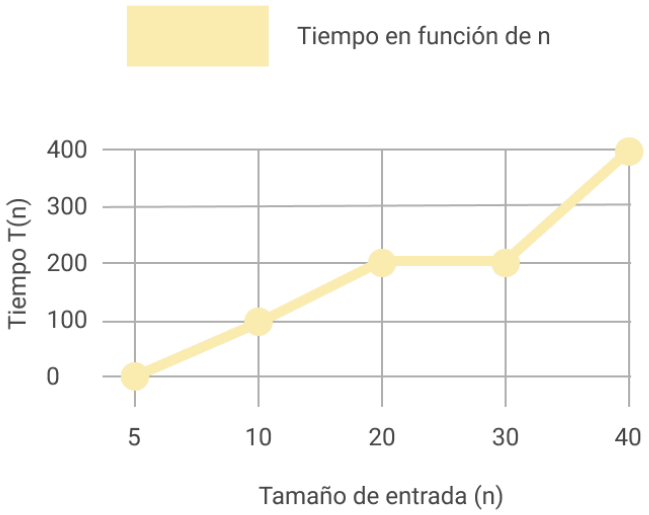
```
Clase CTorres viene-de CObjeto
 Atributos
  nDiscos
 Métodos
  Hanoi(nDiscos, Origen, Destino, Auxiliar)
fClase

Metodo CTorres.Hanoi (nDiscos, Origen, Destino, Auxiliar)
 Si (nDiscos = 1) entonces
  Escribir("Mover disco Origen a Destino")
 sino
  Torres.Hanoi(nDiscos -1, Origen, Auxiliar, Destino)
  Escribir("Mover disco Origen a Destino")
  Torres.Hanoi(nDiscos – 1,Auxiliar, Destino , Origen)
 fSi
FMétodo
```

Reporte de la comparación de algoritmos predefinidos
QuickSort

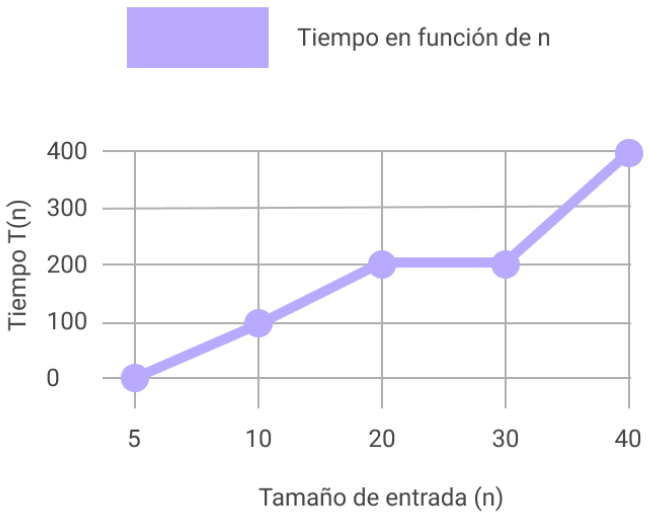
Complejidad Algorítmica $O(n \log n)$

Tiempo de ejecución 00:02:04

Gráfica de la función de tiempo teórica



Gráfica de la función de tiempo real



5. Conclusión

Este informe permite al equipo desarrollar los componentes que permiten documentar y forjar la estructura de nuestro programa, la cuál tiene como finalidad analizar y comparar el tiempo de ejecución de algoritmos, utilizando los conceptos aprendidos en clase y otras herramientas. Se ha diseñado una jerarquía de clases, creando el pseudocódigo y estableciendo una estructura de control básica teniendo en cuenta el diagrama de clases del proyecto.

Hasta el momento, se ha logrado definir la estructura del sistema, estableciendo las clases principales y sus métodos, lo que permite facilitar la implementación de las funcionalidades restantes. Además, se ha avanzado en la creación de un prototipo gráfico que proporciona una interfaz de usuario intuitiva y sencilla, permitiendo abordar todas las funcionalidades de la aplicación.