

Informe Práctica 3

Diseño Arquitectónicos y Patrones
Grado en Ingeniería Informática
Universidad de La Laguna

Alejandro Rodríguez Rojas
alu0101317038@ull.edu.es

1. Introducción	2
2. Desarrollo del framework y estructura del código	2
2.1 Filosofía general	2
2.2 Estructura de paquetes y ficheros	3
2.3 Clase DivConqTemplate (Template Method)	4
2.4 Interfaces Problem y Solution	5
2.5 Paquete sum - Suma de elementos	6
2.6 Paquete max - Cálculo del máximo	10
2.7 Paquete mergesort - Ordenación Merge Sort	10
2.8 Diagrama de Clases UML para la parte de código	11
3. Desarrollo de la interfaz gráfica (Swing + Adapters)	13
3.1 Objetivo	13
3.2 Estructura	13
3.3 Diseño y flujo	15
3.4 Diagrama de Clases UML para la parte gráfica	15
4. Decoradores MeasuredAlgorithm y TracingAlgorithm	16
4.1 Objetivo	16
4.2 Arquitectura y orden	16
4.3 MeasuredAlgorithm	16
4.4 TracingAlgorithm	16
4.5 Diagrama de Clases UML para los decoradores	16
5. Estimación de tiempos	18
6. Ejecución del programa	18
7. Uso de la Inteligencia Artificial en el desarrollo	19
7.1 Fase de planificación y estructura inicial	20
7.2 Fase de implementación de algoritmos concretos	20
7.3 Fase de desarrollo de la interfaz gráfica	21
7.4 Fase de instrumentación: MeasuredAlgorithm y TracingAlgorithm	21
7.5 Fase de depuración y refinamiento	22
7.6 Aportaciones y conclusiones sobre el uso de IA	22
8. Bibliografía	23

1. Introducción

El objetivo de esta práctica es desarrollar un framework genérico para la familia de algoritmos Divide y Vencerás, utilizando el **patrón de diseño Método Plantilla** (Template Method).

Esta arquitectura permite definir el flujo de resolución de forma general y reutilizable, dejando que las subclases concreten los pasos específicos según el problema.

Sobre esta base, se han implementado tres algoritmos representativos:

- **Suma** de los elementos de un array
- Cálculo del **máximo** de un array
- Ordenación por **Merge Sort**

El framework se completa con una **interfaz gráfica** en Java Swing que permite visualizar los resultados y la traza de recursión, y dos decoradores:

- **MeasuredAlgorithm**, que mide el rendimiento interno y las llamadas recursivas.
- **TracingAlgorithm**, que genera una representación visual del árbol recursivo.

Durante el desarrollo, se ha utilizado **ChatGPT-5** como asistente técnico, pidiéndole ayuda con prompts para:

- definir la estructura modular del proyecto,
- depurar problemas de visibilidad y recursión,
- diseñar el árbol de recursión visual.

2. Desarrollo del framework y estructura del código

2.1 Filosofía general

El framework implementa un modelo genérico de resolución Divide & Vencerás:

- 1) Dividir el problema en subproblemas más pequeños.
- 2) Resolver recursivamente los subproblemas.

- 3) Combinar sus soluciones parciales.

El patrón Template Method es el núcleo que define este proceso. Las subclases se limitan a redefinir las operaciones de división, resolución simple y combinación, manteniendo intacto el flujo recursivo.

2.2 Estructura de paquetes y ficheros

En la Figura 1, se puede observar la estructura global del proyecto. A lo largo del informe se comentará cada una de sus partes:

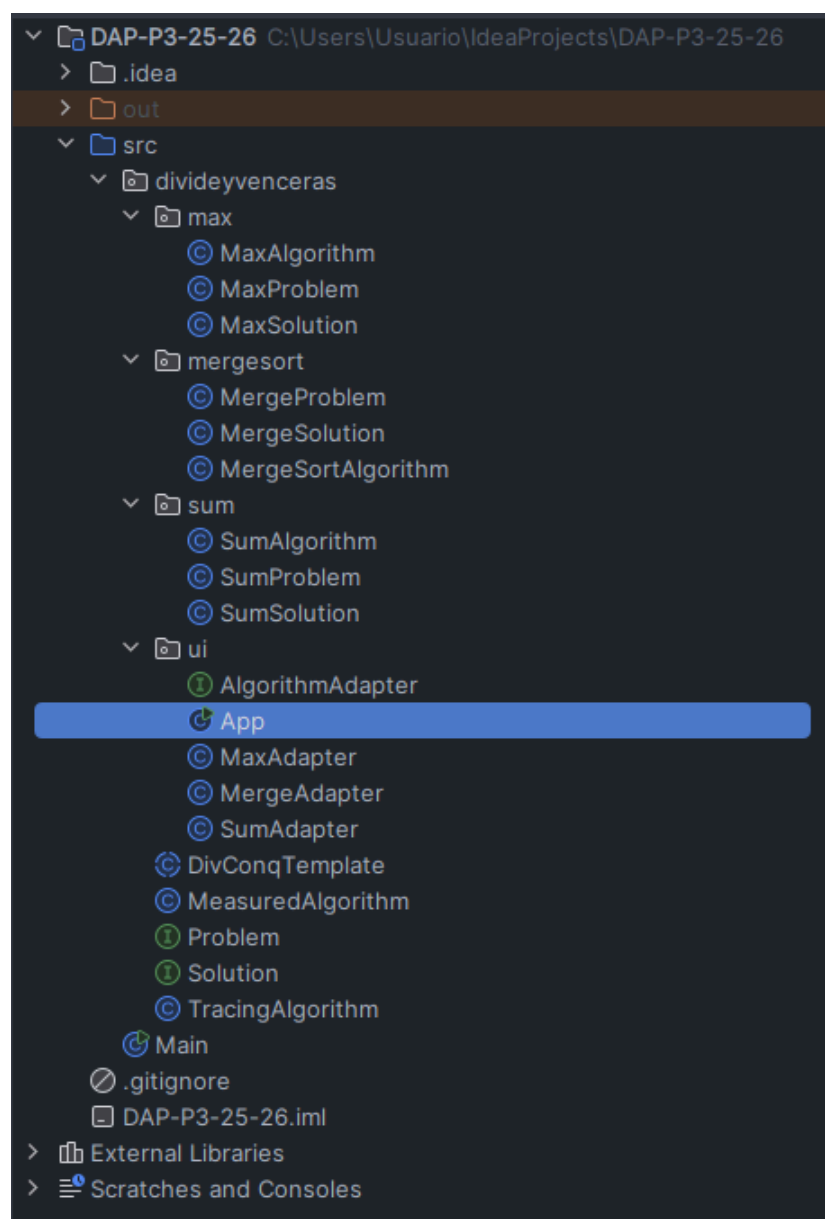


Figura 1: Estructura del proyecto

2.3 Clase DivConqTemplate (Template Method)

La clase abstracta DivConqTemplate, que se muestra en la Figura 2, es el núcleo del framework. Define el método plantilla solve(Problem p), que describe el flujo completo del algoritmo Divide y Vencerás:

```
1 package divideyvenceras;
2
3 public abstract class DivConqTemplate {
4
5     public final Solution solve(Problem p) {
6         Problem[] subProblems;
7
8         if (isSimple(p)) {
9             return simplySolve(p);
10        } else {
11            subProblems = decompose(p);
12        }
13
14        Solution[] subSolutions = new Solution[subProblems.length];
15        for (int i = 0; i < subProblems.length; i++) {
16            subSolutions[i] = solve(subProblems[i]);
17        }
18
19        return combine(p, subSolutions);
20    }
21
22    protected abstract boolean isSimple(Problem p);
23    protected abstract Solution simplySolve(Problem p);
24    protected abstract Problem[] decompose(Problem p);
25    protected abstract Solution combine(Problem p, Solution[] ss);
26 }
```

Figura 2: Clase abstracta DivConqTemplate

Explicación de cada parte

1) solve()

- Método final, no sobrescribible, para garantizar el flujo estándar.
- Controla la recursión completa del algoritmo.

2) **isSimple(Problem p)**

- Determina si el problema es lo bastante simple para resolverse directamente.
- Es el caso base de la recursión.

3) **simplySolve(Problem p)**

- Devuelve la solución directa de un caso simple (p. ej., un único elemento en el array).

4) **decompose(Problem p)**

- Divide el problema en subproblemas más pequeños.

5) **combine(Problem p, Solution[] ss)**

- Combina las soluciones de los subproblemas para obtener el resultado global.

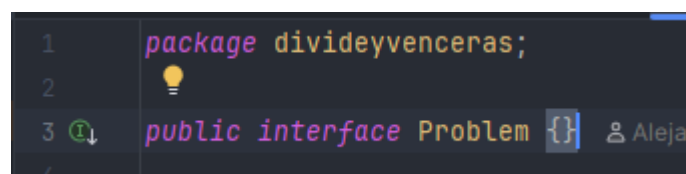
Principios aplicados

- SRP: una sola responsabilidad (definir el flujo genérico).
- OCP: se amplía mediante herencia sin modificar el método plantilla.
- LSP: cualquier subclase puede sustituirse sin alterar el comportamiento del cliente.

2.4 Interfaces Problem y Solution

Problem.java

Interfaz marcador (sin métodos). Representa un problema concreto dentro del paradigma Divide y Vencerás. Cada tipo de problema (SumProblem, MaxProblem, MergeProblem) almacena sus propios datos (array, índices, etc.). En la Figura 3 se observa la interfaz Problem:



```
1 package divideyvenceras;
2
3 public interface Problem {}
```

Figura 3: Interfaz Problem

Solution.java

Interfaz marcador complementaria, que encapsula la solución de un Problem.

```
1 package divideyvenceras;  
2   
3 public interface Solution {}  
4
```

Figura 4: Interfaz Solution

La separación clara entre problema y solución favorece la cohesión y la reutilización del framework. Por ejemplo, cualquier algoritmo puede operar con su propio tipo de problema sin modificar el núcleo.

2.5 Paquete sum - Suma de elementos

En la Figura 5, se observa la estructura y el contenido del paquete sum:

```
▼ [icon] sum  
  © SumAlgorithm  
  © SumProblem  
  © SumSolution
```

Figura 5: Paquete sum con su contenido

SumProblem

En la Figura 6, se muestra como se almacena el array de enteros y el rango a procesar (start, end):

```
1 package divideyvenceras.sum;
2
3 import divideyvenceras.Problem;
4
5 public class SumProblem implements Problem { 11 usages  Alejandro R
6     public int[] data;
7     public int start, end; 5 usages
8
9     public SumProblem(int[] data, int start, int end) { 4 usag
10         this.data = data;
11         this.start = start;
12         this.end = end;
13     }
14 }
```

Figura 6: SumProblem del paquete sum

SumSolution

Guarda el valor entero de la suma parcial, se puede observar en la Figura 7:

```
1 package divideyvenceras.sum;
2
3 import divideyvenceras.Solution;
4
5 public class SumSolution implements Solution { 6 usages  Alejandro R
6     private int value; 2 usages
7
8     > public SumSolution(int value) { this.value = value; }
11
12     > public int getValue() { return value; }
15 }
16
```

Figura 7: SumSolution del paquete sum

SumAlgorithm

Implementa los pasos específicos:

- isSimple: si $\text{end} - \text{start} == 1 \rightarrow$ caso base.
- simplySolve: devuelve `data[start]`.
- decompose: divide el array en dos mitades.
- combine: suma las soluciones parciales.

Este ejemplo es el más simple y se usa como modelo de referencia para validar el framework. En la Figura 8 se puede observar su comportamiento:

```

1      package divideyvenceras.sum;
2
3      import divideyvenceras.*;
4
5      public class SumAlgorithm extends DivConqTemplate { 4 usages  ⌵ Ale
6
7          @Override 3 usages  ⌵ Alejandro Rodríguez Rojas
8      ↗      protected boolean isSimple(Problem p) {
9          SumProblem sp = (SumProblem) p;
10         return (sp.end - sp.start) == 1;
11     }
12
13     @Override 3 usages  ⌵ Alejandro Rodríguez Rojas
14 ↗      protected Solution simplySolve(Problem p) {
15         SumProblem sp = (SumProblem) p;
16         return new SumSolution(sp.data[sp.start]);
17     }
18
19     @Override 3 usages  ⌵ Alejandro Rodríguez Rojas
20 ↗      protected Problem[] decompose(Problem p) {
21         SumProblem sp = (SumProblem) p;
22         int mid = (sp.start + sp.end) / 2;
23         return new Problem[] {
24             new SumProblem(sp.data, sp.start, mid),
25             new SumProblem(sp.data, mid, sp.end)
26         };
27     }
28
29     @Override 3 usages  ⌵ Alejandro Rodríguez Rojas
30 ↗      @
31     protected Solution combine(Problem p, Solution[] ss) {
32         int total = 0;
33         for (Solution s : ss) {
34             total += ((SumSolution) s).getValue();
35         }
36         return new SumSolution(total);
37     }

```

Figura 8: SumAlgorithm del paquete sum

2.6 Paquete max - Cálculo del máximo

En la Figura 9 se muestra la estructura de dicho paquete:

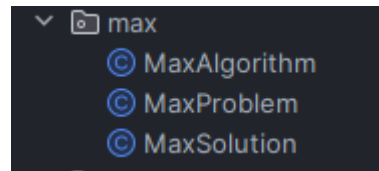


Figura 9: Estructura y contenido del paquete max

- MaxProblem: igual estructura que SumProblem, con el array y los índices.
- MaxSolution: contiene el valor máximo encontrado.
- MaxAlgorithm:
 - isSimple: si el rango tiene un único elemento.
 - simplySolve: devuelve ese elemento como máximo.
 - decompose: parte el array en dos.
 - combine: devuelve el máximo entre las dos soluciones.

Muestra la reutilización total del método plantilla sin repetir lógica de recursión.

2.7 Paquete mergesort - Ordenación Merge Sort

Este paquete mantiene el mismo formato que las otras implementaciones anteriores del algoritmo, como podemos observar en la Figura 10:

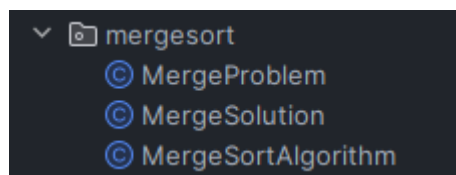


Figura 10: Estructura y contenido del paquete mergesort

- MergeProblem

Igual patrón: array + índices start y end.

- MergeSolution

Contiene un nuevo array ordenado (resultado parcial).

- MergeSortAlgorithm
 - isSimple: si el subarray tiene tamaño 1.
 - simplySolve: devuelve el array con un solo elemento.
 - decompose: divide en mitad izquierda y derecha.
 - combine: fusiona las dos mitades ya ordenadas.

Este algoritmo demuestra el poder del framework: a pesar de su complejidad, el flujo de resolución es idéntico al de los ejemplos más simples.

2.8 Diagrama de Clases UML para la parte de código

En el UML generado por IntelliJ IDEA, se observa la clase abstracta DivConqTemplate como superclase, con tres subclases concretas (SumAlgorithm, MaxAlgorithm, MergeSortAlgorithm) (Figura 11) y las relaciones de asociación con las interfaces Problem y Solution. Se destacan las composiciones entre cada algoritmo y sus clases de datos (Problem y Solution específicos) (Figura 12 y 13).

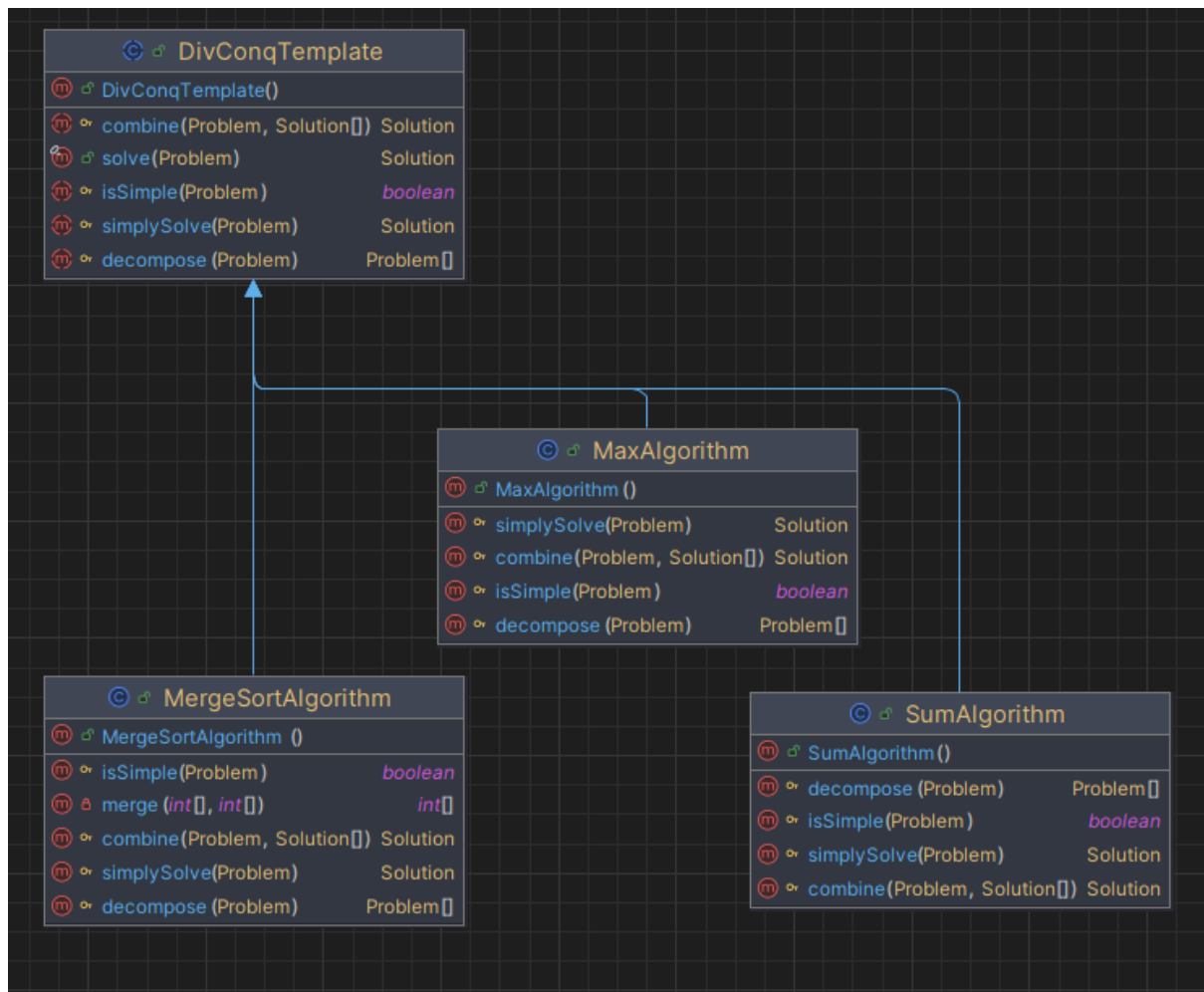


Figura 11: DivConqTemplate como superclase, con tres subclases concretas (SumAlgorithm, MaxAlgorithm, MergeSortAlgorithm)

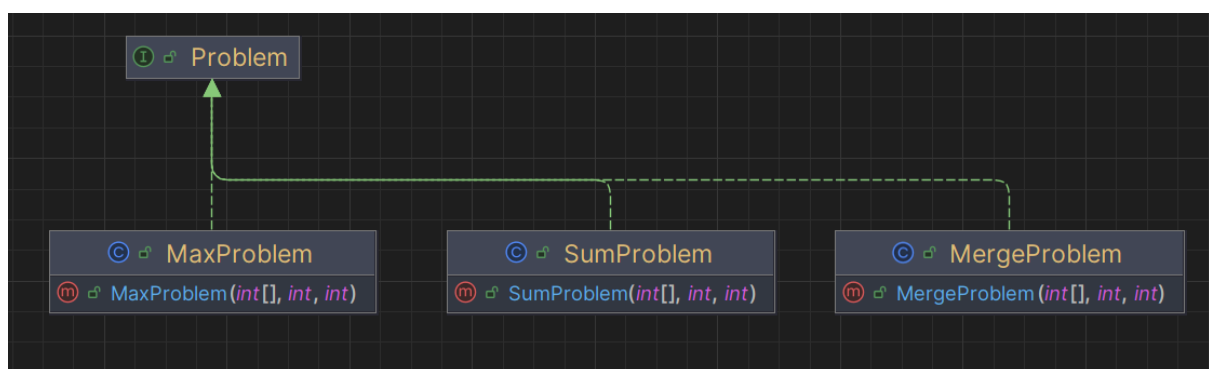


Figura 12: Interfaz Problem y sus implementaciones

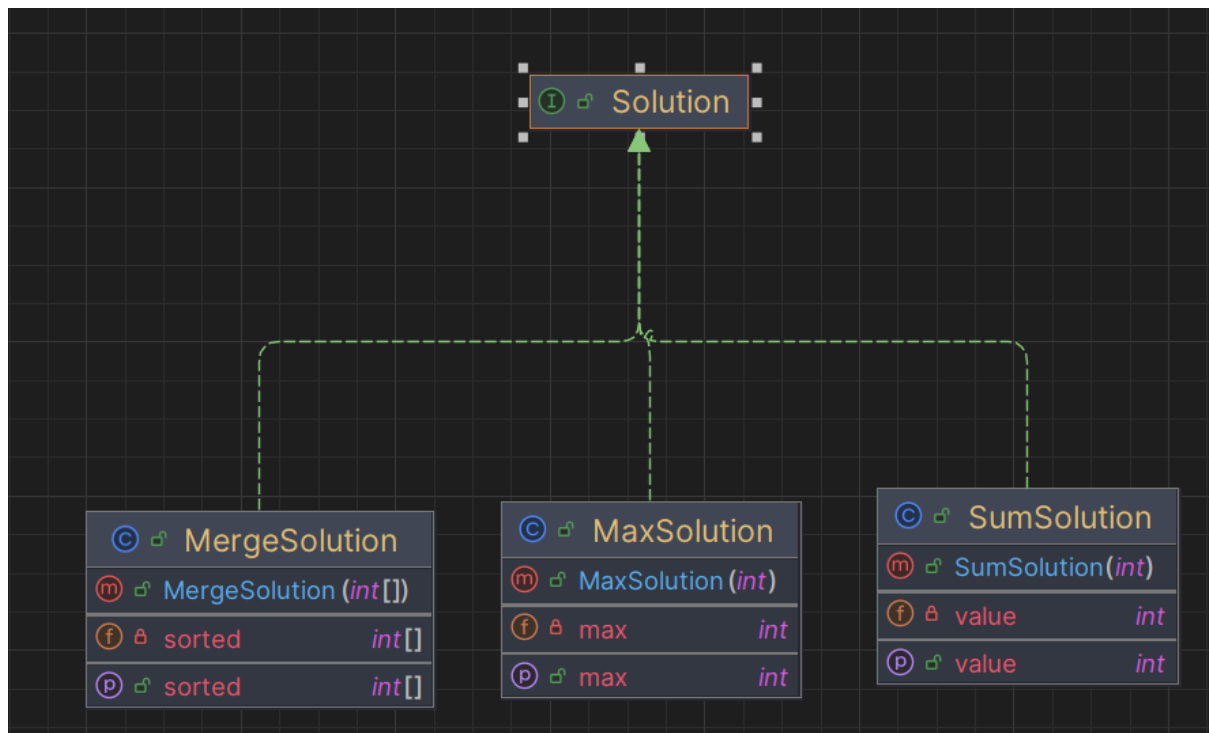


Figura 13: Interfaz Solution con sus implementaciones

3. Desarrollo de la interfaz gráfica (Swing + Adapters)

3.1 Objetivo

La interfaz gráfica implementa una vista interactiva que permite ejecutar y visualizar los algoritmos Divide y Vencerás, mostrando resultados y trazas recursivas.

3.2 Estructura

En la Figura 14, se muestra la estructura del paquete encargado de toda la parte gráfica:

- AlgorithmAdapter: interfaz genérica con métodos makeProblem, solve, render, y getAlgorithm.
- SumAdapter, MaxAdapter, MergeAdapter: implementan la interfaz para cada algoritmo, creando problemas y soluciones adecuadas.
- App.java: gestiona la ventana Swing y el flujo de ejecución.

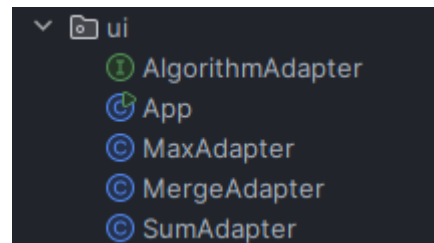


Figura 14: Paquete ui encargado de englobar la parte gráfica del programa

Por ejemplo, en la Figura 15, se observa la implementación MergeAdapter:

```

1  package divideyvenceras.ui;
2
3  import divideyvenceras.*;
4  import divideyvenceras.mergesort.*;
5  import java.util.Arrays;
6
7  public class MergeAdapter implements AlgorithmAdapter { 1 usage  Alejandro Rodríguez Rojas
8      private final MergeSortAlgorithm alg = new MergeSortAlgorithm(); 2 usages
9
10     @Override public String name() { return "Merge Sort"; } Alejandro Rodríguez Rojas
11
12     @Override public Problem makeProblem(int[] data) { 1 usage  Alejandro Rodríguez Rojas
13         return new MergeProblem(data, start: 0, data.length);
14     }
15
16     @Override public Solution solve(Problem p) { no usages  Alejandro Rodríguez Rojas
17         return alg.solve(p);
18     }
19
20     @Override public String render(Solution s) { 1 usage  Alejandro Rodríguez Rojas
21         int[] sorted = ((MergeSolution) s).getSorted();
22         return "Sorted array = " + Arrays.toString(sorted);
23     }
24
25     @Override 1 usage  Alejandro Rodríguez Rojas
26     public DivConqTemplate getAlgorithm() { return alg; }
27 }
28
29
30

```

Figura 15: Código de MergeAdapter

3.3 Diseño y flujo

- Entrada de datos → selección de algoritmo → ejecución decorada (MeasuredAlgorithm + TracingAlgorithm) → salida en JTextArea.
- Permite generar arrays aleatorios mediante un botón adicional.

3.4 Diagrama de Clases UML para la parte gráfica

El UML generado por el IDE muestra el paquete ui con App y los adaptadores (SumAdapter, MaxAdapter, MergeAdapter), todos implementando la interfaz común AlgorithmAdapter, la cual actúa como capa de abstracción entre la GUI y el framework de algoritmos. Se puede observar en la Figura 16:

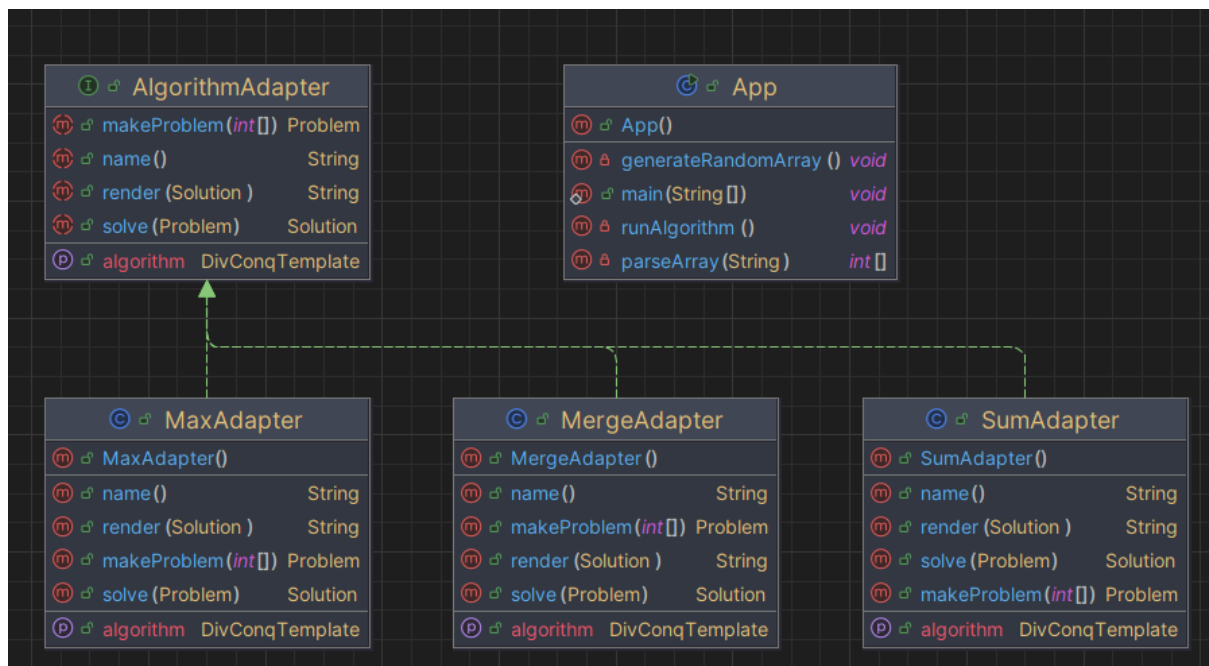


Figura 16: Diagrama de Clases UML de la parte gráfica

4.Decoradores MeasuredAlgorithm y TracingAlgorithm

4.1 Objetivo

Ambos decoradores extienden DivConqTemplate para añadir funcionalidades:

- MeasuredAlgorithm mide el rendimiento interno (tiempo, recursiones, hojas).
- TracingAlgorithm dibuja el árbol de recursión con indentación jerárquica.

4.2 Arquitectura y orden

baseAlg → MeasuredAlgorithm → TracingAlgorithm → solve()

4.3 MeasuredAlgorithm

- Campos: solveCalls, leafCount, workNanos.
- Métodos delegan en inner y miden tiempos con System.nanoTime().
- Los resultados se muestran en la UI junto con la traza.

4.4 TracingAlgorithm

- Gestiona la variable depth para controlar niveles de recursión.
- Utiliza símbolos Unicode (└─, ┆, │) para representar la jerarquía.
- Si detecta un MeasuredAlgorithm, muestra el nombre real del algoritmo interno.
- Permite visualizar el flujo exacto del método plantilla.

4.5 Diagrama de Clases UML para los decoradores

En el UML se observa que ambos decoradores heredan de DivConqTemplate y contienen internamente una referencia a otra instancia de la misma clase, lo que representa la composición característica del patrón Decorator.

Este diseño mantiene el principio Open/Closed, permitiendo ampliar funcionalidades sin modificar las clases originales. Figura 17:

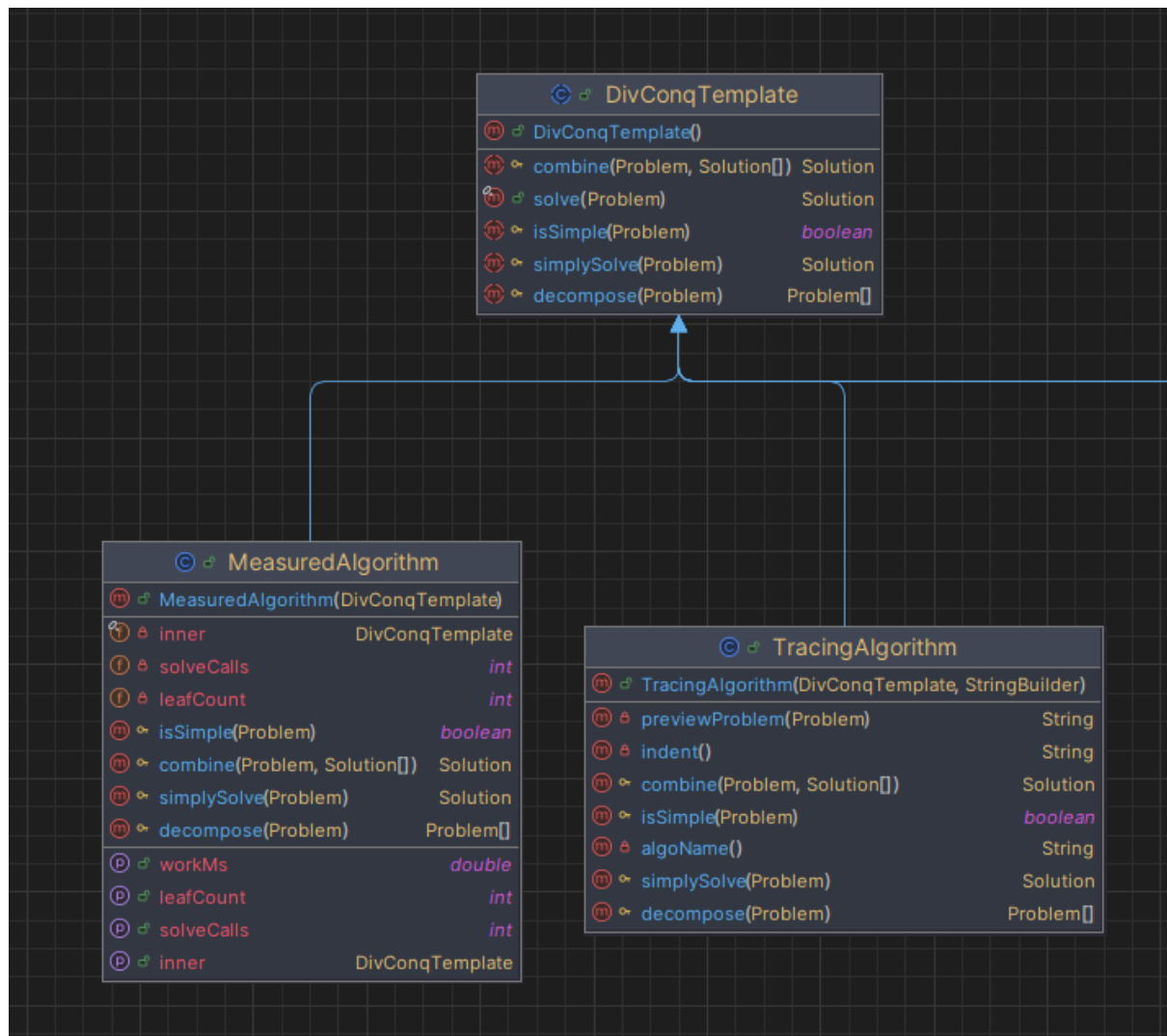


Figura 17: Diagrama de clases UML para los decoradores

5. Estimación de tiempos

En la Figura 18 se puede observar, una comparativa de los tiempos estimados para cada tarea que compone esta práctica así como el tiempo real que requirió cada tarea:

Tarea	Tiempo estimado	Tiempo real
Implementación de Problem, Solution y algoritmos (sum, max, mergesort)	2 h 00 min	2 h 10 min
Diseño de AlgorithmAdapter y estructura UI	1 h 30 min	1 h 40 min
Implementación de generación de arrays aleatorios	0 h 45 min	0 h 50 min
Desarrollo de TracingAlgorithm	1 h 15 min	1 h 30 min
Desarrollo de MeasuredAlgorithm	1 h 00 min	1 h 10 min
Depuración y validación visual	0 h 45 min	1 h 00 min
Redacción del informe y documentación	1 h 30 min	1 h 50 min
Totales	9 h 15 min	10 h 55 min

Figura 18: Tabla de tiempos

6. Ejecución del programa

A continuación se muestra en la Figura 19, un ejemplo de la ejecución del archivo App. En este, se utiliza el modo random para obtener un vector aleatorio y se ha seleccionado la opción de Merge Sort. Como resultado, se obtiene en la ventana el vector con el resultado del algoritmo correspondiente, así como las estadísticas que aporta el decorador MeasuredAlgorithm y el árbol que muestra el decorador TracingAlgorithm.

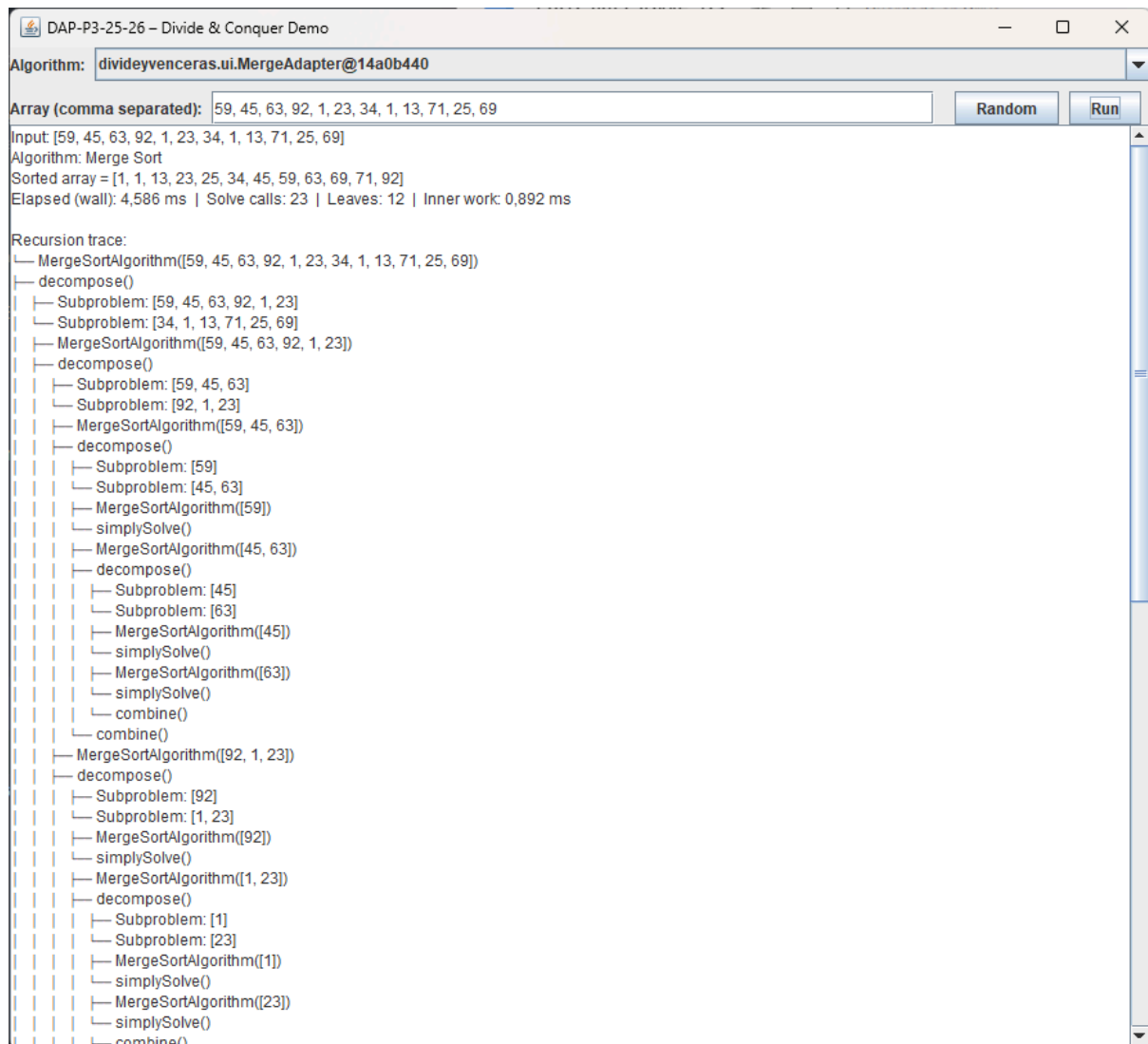


Figura 19: Ejecución del programa (App)

7. Uso de la Inteligencia Artificial en el desarrollo

Durante el desarrollo de este proyecto se empleó inteligencia artificial generativa (ChatGPT-5) como asistente de programación y diseño arquitectónico, actuando como una herramienta de apoyo continuo durante todas las fases del trabajo. El uso de la IA no sustituyó el razonamiento ni la implementación manual del código, sino que complementó el proceso, ofreciendo asistencia en tareas de diseño, depuración y documentación técnica.

7.1 Fase de planificación y estructura inicial

En la etapa de diseño del proyecto, la IA se utilizó para definir la arquitectura del framework Divide & Vencerás y establecer una estructura de paquetes coherente. A través de distintos prompts, se pidió orientación sobre:

- cómo organizar los paquetes `divideyvencerás/`, `sum/`, `max/` y `mergesort/`;
- cómo aplicar el patrón Template Method de forma genérica;
- y cómo respetar los principios SOLID desde la fase de diseño.

Ejemplo de prompt utilizado:

“Quiero desarrollar un framework en Java para algoritmos divide y vencerás aplicando el patrón método plantilla. ¿Qué estructura de clases y paquetes me recomiendas y qué interfaces debería incluir?”

Resultado obtenido:

La IA propuso una estructura modular con una clase abstracta `DivConqTemplate` y las interfaces `Problem` y `Solution`, acompañada de una justificación de cómo cada subclase concreta debía implementar los métodos `isSimple`, `simplySolve`, `decompose` y `combine`. Esa propuesta se adoptó casi íntegramente y se convirtió en la base del proyecto.

7.2 Fase de implementación de algoritmos concretos

Durante la implementación de `SumAlgorithm`, `MaxAlgorithm` y `MergeSortAlgorithm`, la IA se empleó para verificar la corrección del flujo recursivo y asegurar que cada algoritmo respetase la estructura definida por la plantilla.

Ejemplo de prompts:

“Explícame paso a paso cómo debería implementarse un algoritmo de suma usando la plantilla `DivConqTemplate`.”

“¿Cómo se adaptaría el patrón para `MergeSort`, dividiendo el problema en dos subarrays?”

Resultados obtenidos:

La IA proporcionó ejemplos de código que mostraban cómo implementar la división y la combinación en cada algoritmo, junto con explicaciones de los casos base y de las condiciones de terminación.

Estos fragmentos se utilizaron como referencia para el desarrollo final, ajustándolos manualmente al estilo del proyecto.

7.3 Fase de desarrollo de la interfaz gráfica

La interfaz Swing (App.java) fue una de las partes donde el uso de la IA resultó más valioso, especialmente en la organización de componentes y la separación de responsabilidades entre la lógica y la presentación.

Prompts utilizados:

“Ayúdame a estructurar una ventana Swing con un campo de texto, un botón para ejecutar el algoritmo y un área para mostrar resultados.”

“Quiero añadir un botón para generar arrays aleatorios; ¿cómo integro esa función en mi clase App?”

Resultados obtenidos:

Se generaron ejemplos de código Swing bien estructurados con paneles BorderLayout y FlowLayout, que sirvieron como plantilla base para la versión final. Además, la IA sugirió implementar el patrón Adapter, mediante una interfaz AlgorithmAdapter para desacoplar la GUI del modelo de algoritmos, lo que mejoró significativamente la extensibilidad del proyecto.

7.4 Fase de instrumentación: MeasuredAlgorithm y TracingAlgorithm

La IA también se empleó para diseñar e integrar los decoradores del framework. El proceso fue iterativo: se fueron generando versiones incrementales de los decoradores hasta lograr un comportamiento correcto y visualmente legible.

Prompts utilizados:

“Quiero medir el número de llamadas recursivas y el tiempo de ejecución de mi algoritmo divide y vencerás sin modificar las clases existentes.”

“Necesito mostrar el árbol de recursión con indentaciones y símbolos (|—, L—, |) en el área de texto. ¿Cómo puedo hacerlo controlando el nivel de profundidad?”

Resultados obtenidos:

La IA propuso implementar MeasuredAlgorithm como un decorador de DivConqTemplate que contase las llamadas y midiera tiempos usando System.nanoTime().

Para TracingAlgorithm, proporcionó un esquema con el control de depth y el uso de caracteres Unicode para representar la estructura recursiva.

También ayudó a corregir problemas de visibilidad (métodos protected) y errores de indentación en la salida textual.

7.5 Fase de depuración y refinamiento

Durante la integración de todas las partes (núcleo, UI y decoradores), la IA asistió en la resolución de errores de compilación y en la mejora de la legibilidad del código.

Se discutieron problemas como:

- incompatibilidad entre tipos Problem y Solution;
- errores de ClassCastException al mezclar algoritmos en el main;
- y ajustes en la gestión de niveles de recursión (depth).

Ejemplo de prompt:

“Me aparece un ClassCastException cuando ejecuto MergeSort después de MaxAlgorithm. ¿A qué puede deberse y cómo lo soluciono?”

Resultado obtenido:

La IA explicó que el error se debía a la reutilización del método solve() con un tipo de Problem incorrecto, y recomendó no compartir instancias de plantilla entre algoritmos diferentes, corrigiendo así el bug de ejecución.

7.6 Aportaciones y conclusiones sobre el uso de IA

El uso de la inteligencia artificial supuso una mejora significativa en la productividad y la comprensión conceptual del patrón Template Method aplicado a Divide & Vencerás.

Las principales aportaciones fueron:

- Aceleración del diseño arquitectónico: la IA ayudó a estructurar el proyecto con claridad desde el inicio.

- Mejor comprensión de patrones: permitió obtener explicaciones detalladas y comparativas entre distintos enfoques.
- Soporte en depuración: identificó causas de errores comunes de recursión y tipado.
- Refuerzo pedagógico: proporcionó feedback inmediato sobre la aplicación de principios SOLID.
- Optimización del código Swing: generó soluciones limpias y adaptables a los requerimientos.

El resultado de esta colaboración es un sistema bien diseñado, documentado y didáctico, donde la inteligencia artificial actuó como mentor técnico y herramienta de apoyo al aprendizaje de diseño arquitectónico.

8. Bibliografía

Gamma, E. et al. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Wikipedia. Divide and conquer algorithm. Disponible en:
https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm

Oracle. Documentación oficial de Java SE 21. Disponible en: <https://docs.oracle.com/>

Universidad de La Laguna (ULL). Material docente de la asignatura Diseño Arquitectónico y Patrones (DAP), curso 2025/2026.