



PRÁCTICA 01

ESQUEMA DIVIDE AND CONQUER

Torneo de tenis

Autores

Antonio José Jiménez Luque
Adrián Jiménez Benítez

Asignatura

Estructura de Datos y Algoritmos II

Titulación

Grado en Ingeniería Informática



Índice.

1. Antecedentes.....	3
2. Objetivos.....	3
2.1. Realización de las pruebas experimentales.....	4
2.2. Realización de las pruebas.....	5
3. Proyecto.....	6
3.1. Caso 1. N es potencia de 2 mediante simplificación.....	6
3.1.1. Estudio de la implementación.....	7
3.1.2. Estudio teórico.....	7
3.1.3. Estudio experimental.....	8
3.2. Caso 2.....	10
3.2.1. Estudio teórico.....	10
3.2.2. Estudio experimental.....	11
3.3. Caso 1 con doble llamada recursiva.....	15
3.3.1. Estudio teórico.....	15
3.3.2. Estudio experimental.....	16
A. Anexo.....	18
A.1. Diseño del código.....	19
A.2. Esquema archivos fuente.....	20
B. Bibliografía.....	22

Índice de tablas.

Tabla 1. Organización de las tareas.....	4
Tabla 2. Propiedades equipo utilizado.	5
Tabla 3: Caso 1 potencia de dos	9
Tabla 4: Caso 2 Par	13
Tabla 5: Caso 2 Impar	14
Tabla 6: Caso 1 con dos llamadas recursivas	17

Índice de Ilustraciones.

Ilustración 1. Ejemplo test.....	5
Ilustración 2. Ejecución correcta de todos los test.	6
Ilustración 3. Ejemplo ejecución caso 1.....	7
Ilustración 4. Ejemplo de salida caso 01 para $n = 8$	8
Ilustración 5. Resultados experimentales primer caso.	9
Ilustración 6. Orden espacial caso 1.....	10
Ilustración 7: Ejemplo ejecución caso 2.....	10
Ilustración 8. Ejemplo de salida caso 02 para $n = 11$	12
Ilustración 9: Resultados experimentales segundo caso par	13
Ilustración 10: Resultados experimentales segundo caso impar	14
Ilustración 11. Orden espacial caso 2 par.....	15
Ilustración 12. Orden espacial caso 2 impar.	15
Ilustración 13: Ilustración 5: Ejemplo de salida caso 01 con doble llamada recursiva para $n = 16$	16
Ilustración 14: Resultados experimentales primer caso dos llamadas recursivas	17
Ilustración 15. Orden espacial caso 1 doble llamada recursiva.....	18
Ilustración 16. Diagrama de Clases.....	19
Ilustración 17. Diagrama de Clases para las pruebas.	20

1. Antecedentes.

La necesidad de diseñar e implementar soluciones eficientes para problemas complejos es un pilar fundamental en el campo de la ingeniería informática, particularmente en lo que respecta a la manipulación y gestión de datos a través de estructuras y algoritmos avanzados. En este contexto, el proyecto encargado a nuestro equipo, como parte de la asignatura Estructura de Datos y Algoritmos II, consiste en la elaboración de un esquema para un torneo de tenis, aprovechando el enfoque algorítmico divide-and-conquer (divide y vencerás). Este enfoque no solo constituye un reto académico, sino que también simula una situación real en la que EDASoft, nuestro equipo de desarrollo se enfrenta a la demanda de soluciones software eficientes por parte de un cliente, en este caso, el club de tenis de EDAland.

El torneo de tenis, denominado EDAland-Garros, representa un desafío significativo debido a su estructura de todos contra todos, implicando la necesidad de desarrollar un cuadro de cruces que garantice que cada jugador se enfrente exactamente una vez contra cada uno de sus competidores, con restricciones adicionales como la limitación de descanso. La complejidad del problema radica en la organización de los emparejamientos y la programación de los partidos para optimizar los tiempos y recursos disponibles, reflejando así la aplicación práctica de los conceptos teóricos aprendidos en la asignatura.

Para abordar este desafío, se explorará la aplicación del método divide-and-conquer, un enfoque algorítmico que descompone un problema en subproblemas más pequeños resuelve estos subproblemas de manera recursiva, y finalmente combina sus soluciones para resolver el problema original. Este método no solo es relevante por su eficacia en la solución de problemas complejos, sino también por su capacidad para ilustrar de manera práctica los principios de diseño y análisis de algoritmos. La implementación de este esquema en el contexto del torneo de tenis de EDAland-Garros permitirá no solo cumplir con los requisitos del cliente, sino también poner a prueba y expandir nuestra comprensión y habilidades en la materia.

Este proyecto, por tanto, no solo busca cumplir con un requisito académico, sino que también pretende simular un escenario realista en el que los conocimientos teóricos deben ser aplicados para resolver problemas prácticos complejos, preparándonos así para los desafíos que enfrentaremos en nuestra futura carrera profesional. La solución propuesta será evaluada no solo en términos de su correctitud y eficiencia, sino también en su capacidad para ser implementada y adaptada en situaciones reales, reflejando la relevancia y aplicabilidad de los conceptos y métodos estudiados en Estructura de Datos y Algoritmos II.

2. Objetivos.

La presente práctica tiene como objetivo principal explorar y aplicar el método algorítmico Divide y Vencerás en el contexto de la organización de un torneo de tenis. Se busca implementar soluciones eficientes que respondan a las necesidades específicas de dicho torneo, abordando diferentes escenarios estructurales para el emparejamiento de los jugadores. Los objetivos específicos de la práctica se detallan a continuación:

- i. Primer Caso (Potencia de Dos - Esquema de Simplificación): Desarrollar e implementar un algoritmo que utilice el enfoque de simplificación del método Divide y Vencerás para organizar un torneo en el cual el número de participantes, n , es una potencia de dos. El algoritmo debe garantizar que todos los jugadores se enfrenten entre sí en un formato de todos contra todos, optimizando el esquema para completar el torneo $n-1$ días.
- ii. Segundo Caso (Número Par o Impar - Esquema de Simplificación): Ampliar la aplicación del método Divide y Vencerás para abordar torneos donde el número de jugadores, n , puede ser par o impar. Aquí, el algoritmo debe adaptarse para asegurar la organización eficiente del torneo, terminando en $n-1$ días si n es par, o en n días si n es impar, manteniendo el principio de que cada jugador compita exactamente una vez contra cada uno de sus adversarios.
- iii. Tercer Caso (Potencia de Dos - Doble Llamada Recursiva): Implementar un algoritmo que, manteniendo el número de participantes como una potencia de dos, haga uso de una estrategia de doble llamada recursiva. Este enfoque debe superar las limitaciones presentadas en el esquema de simplificación mediante la inclusión de restricciones específicas, que se explica-

rán en profundidad en el apartado correspondiente, para ofrecer una solución más refinada y eficiente para la organización del torneo.

Cada uno de estos casos deberá ser abordado considerando no solo la correcta ejecución del esquema algorítmico propuesto, sino también su eficiencia desde el punto de vista de la complejidad temporal y espacial. Asimismo, se realizará un análisis comparativo cualitativo y cuantitativo de las soluciones implementadas, enfocado tanto en la teoría como en la práctica, para validar la eficacia del método Divide y Vencerás en el ámbito de la organización de torneos deportivos.

Para este proyecto seguidamente detallaremos el líder y las tareas que ha realizado cada uno de los componentes del equipo.

El líder para esta práctica es Adrián Jiménez Benítez.

En la siguiente tabla se recogerá cada una de las tareas a implementar por los distintos miembros del equipo.

<i>Tareas</i>	<i>Miembro del equipo</i>
Primer caso	Antonio José Jiménez Luque
Segundo caso	Adrián Jiménez Benitez
Tercer caso	Antonio José Jiménez Luque
Mantenimiento del repositorio	Antonio José Jiménez Luque
Realización de la memoria	Antonio José Jiménez Luque Adrián Jiménez Benitez

Tabla 1. Organización de las tareas.

Para la organización de la realización de la práctica se ha desarrollado un Excel donde se recoge más detalladamente como se ha organizado el equipo para la realización de la práctica.

A continuación, se detallarán algunas pautas que se han seguido para la elaboración de las pruebas experimentales.

2.1. Realización de las pruebas experimentales.

Para la realización de las pruebas experimentales, se va a utilizar un equipo portátil. Debido a errores por el tamaño de memoria asignado en el almacenamiento dinámico de Java se ha modificado el tamaño de almacenamiento dinámico de Java, con referente en *-Xms* (para establecer el tamaño de almacenamiento dinámico inicial) y *-Xmx* (para establecer el tamaño máximo de almacenamiento dinámico) en los siguientes parámetros:

- Xms30GB
- Xmx30GB

Con esta configuración podremos realizar pruebas con variables de mayor tamaño. Con respecto al equipo utilizado, se ha utilizado para las pruebas un equipo (portátil) de un miembro del equipo, debido a que para estas pruebas en concreto no sería lo idóneo mezclar ejecuciones en equipos diferentes. Las propiedades del equipo se detallan en la siguiente tabla:

Equipo	Portátil
Sistema Operativo (SO)	Windows 11 Pro
Modo	Rendimiento
Estado (Batería / Corriente)	Corriente
CPU	12th Gen Intel i7-12700H 14 Núcleos <ul style="list-style-type: none">- 6 Performance-cores- 8 Efficient-cores Total de subprocesos: 20 Frecuencia turbo máxima 4.70 GHz
GPU	Nvidia GeForce RTX 3060 Laptop GPU
Memoria RAM	32GB DDR5
Aplicación Utilizada	Eclipse IDE for java Developers Versión 2023-12
JDK	Java SE 17

Tabla 2. Propiedades equipo utilizado.

Las pruebas se han realizado como se muestra en la tabla con el perfil de Rendimiento y el equipo siempre conectado a corriente. Para las pruebas se han ejecutado el algoritmo 10 veces en las cuales hemos sacado el tiempo medio para así poder suavizar las irregularidades que hemos podido tener en cada una de las ejecuciones por motivos del sistema operativo u motivos externos.

2.2. Realización de las pruebas.

Para la comprobación del correcto funcionamiento de los algoritmos desarrollados se ha creado la clase *DivideAndConquerTest.java* donde hemos agregado los distintos juegos de pruebas para comprobar el correcto funcionamiento de los algoritmos empleados. Para la visualización de los distintos métodos empleados se pueden visualizar en el anexo (véase A.1 Diseño del código).

A continuación, se adjuntan una captura con un ejemplo de un test empleado y otra ilustración con los test pasados.

```
@Test
public void testCasolConCuatroJugadores() {
    // Caso con 4 jugadores
    DivideAndConquer dyv = new DivideAndConquer(4,1);
    dyv.DivideAndConquer1(4);

    int[][] resultadoEsperado = {
        { 2, 3, 4 },
        { 1, 4, 3 },
        { 4, 1, 2 },
        { 3, 2, 1 }
    };

    assertEquals(resultadoEsperado, dyv.getTabla());
}
```

Ilustración 1. Ejemplo test.

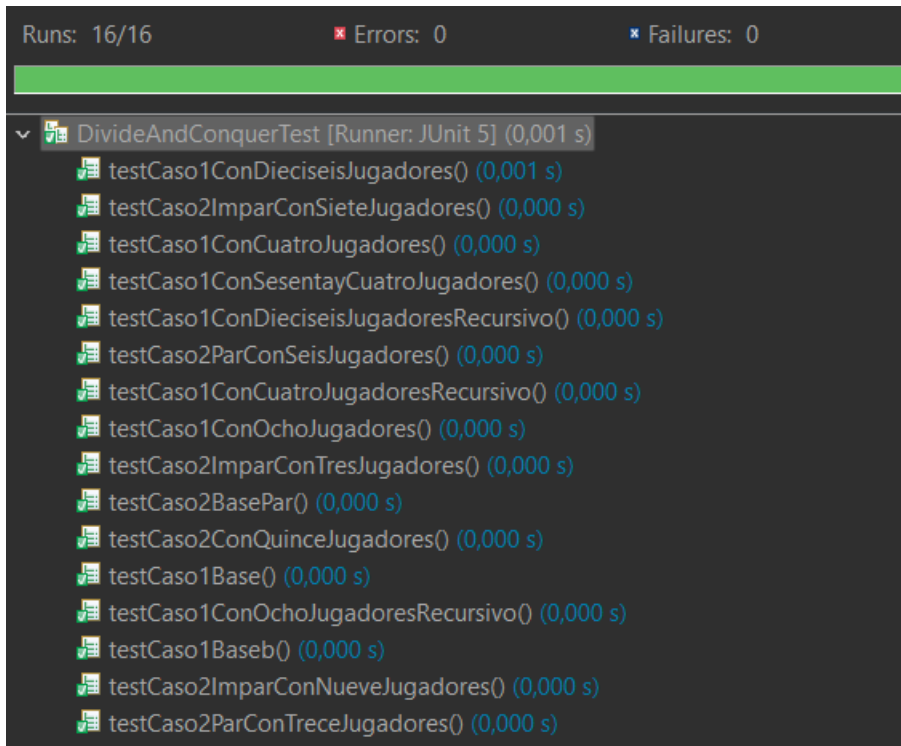


Ilustración 2. Ejecución correcta de todos los test.

3. Proyecto.

Este documento se ha elaborado con el objetivo principal de profundizar en los conocimientos adquiridos sobre el tema de Divide y Vencerás, aplicando la teoría estudiada y presentada para poder aplicarla de manera práctica y ante un posible ejercicio que simule un problema real que podamos encontrar en nuestra vida profesional.

Para la realización de esta práctica, podemos tener en consideración algunas mejoras de la implementación original. Debido a que principalmente como se expone en el informe de la práctica, para poder facilitar la implementación, en algunos casos se genera una fila y una columna extra para poder tratar de mejor manera los índices. En nuestra implementación por ejemplo del caso 1 (véase Apartado 4) hemos decidido realizar una implementación sin necesidad de tener que generar una fila y una columna extra, debido a que hemos creído oportuno no crearlo para así *poder optimizar el uso de la memoria*.

Para la visualización del diagrama de clases se ha agregado el plugin de *Amateras* para poder crear los diagramas de clases.

3.1. Caso 1. N es potencia de 2 mediante simplificación.

En este apartado, se va a detallar el primer caso de la práctica, donde n es potencia de dos ($n = 2^k$). En este caso, aplicaremos la técnica de Divide y Vencerás de simplificación (solo una llamada recursiva) para construir una tabla donde cada fila pertenece a n jugadores y cada columna representar los días en los que cada jugador juega $n - 1$ días. A continuación, se ilustra un ejemplo de la ejecución.

	d1
j1	2
j2	1

	d1	d2	d3
j1	2	3	4
j2	1	4	3
j3	4	1	2
j4	3	2	1

	d1	d2	d3	d4	d5	d6	d7
j1	2	3	4	5	6	7	8
j2	1	4	3	6	7	8	5
j3	4	1	2	7	8	5	6
j4	3	2	1	8	5	6	7
j5	6	7	8	1	4	3	2
j6	5	8	7	2	1	4	3
j7	8	5	6	3	2	1	4
j8	7	6	5	4	3	2	1

Ilustración 3. Ejemplo ejecución caso 1.

3.1.1. Estudio de la implementación.

La implementación que se ha propuesto para detallar es una técnica de Divide y Vencerás por simplificación (una sola llamada recursiva). En este caso, se ha decidido para realizar la implementación, generando una variable de clase para generar la matriz inicial, generándola solo inicialmente. También agregar que se ha decidido como se ha explicado en el apartado (véase Apartado 3) generar la matriz sin la implementación que se ofrece en el guion de la práctica (generando una columna y fila adicional) en este caso hemos generado la matriz con el rango necesario, siendo las filas n jugadores y las columnas $n - 1$. De esta manera podemos optimizar el uso de memoria cada vez que se ejecute el algoritmo.

La estructura general de la implementación ha sido la siguiente:

Para la implementación se ha optado por una función principal *DivideAndConquer1* y tres funciones auxiliares que llenan las partes específicas de la matriz. Concretamente las funciones son:

- CuadranteInferiorIzquierdo.
- CuadranteSuperiorDerecho.
- CuadranteInferiorDerecho.

La función principal *DivideAndConquer1(n)* es la función principal que implementa la estrategia de dividir y vencerás mediante simplificación. Divide el problema en subproblemas más pequeños hasta alcanzar un caso base ($n = 2$), donde se establecen los enfrentamientos directamente. Después, organizamos los enfrentamientos en los diferentes cuadrantes de la matriz mediante las llamadas auxiliares anteriormente detalladas.

3.1.2. Estudio teórico.

En este apartado vamos a desarrollar el estudio teórico.

- Caso base: Para $n = 2$, el algoritmo realiza operaciones en tiempo constante, $O(1)$.
- Caso principal: La función principal se llama a sí misma con la mitad del tamaño del problema $\frac{n}{2}$, lo que implica una profundidad de recursión de $O(\log n)$.
 - o Cuadrantes: Cada una de las funciones del cuadrante realiza iteraciones que dependen de n , pero trabajan sobre la mitad de n para jugadores y días, con la complejidad de cada función siendo $O(\frac{n^2}{4})$. Como disponemos de tres funciones que realizan este trabajo después de cada llamada recursiva, el trabajo total en cada nivel de recursión es $O\left(3 \cdot \frac{n^2}{4}\right) = O(\frac{n^2}{4})$, simplificando las constantes.

Dado que el algoritmo realiza una división recursiva del problema, y en cada nivel de la recursión se realiza un trabajo proporcional a $\frac{n^2}{4}$, la complejidad total del algoritmo es $O(\log n) + O(n^2)$. Por lo que cuando n crece significativamente, la complejidad es $O(n^2)$.

Con respecto al orden espacial, el algoritmo crea una matriz cuadrada de tamaño $n \times n$ para almacenar los resultados de los enfrentamientos entre jugadores y los días. En cada llamada recursiva, la matriz se divide en cuatro cuadrantes y se aplican operaciones en cada uno de ellos. Los métodos “CuadranteInferiorIzquierdo”, “CuadranteSuperiorDerecho” y “CuadranteInferiorDerecho” manipulan diferentes secciones de la matriz según su posición relativa. Dado que la matriz ocupa n^2 espacios de memoria, el orden espacial del algoritmo es $O(n^2)$, lo que significa que el espacio requerido aumenta cuadráticamente con respecto al tamaño de entrada n .

3.1.3. Estudio experimental.

En este apartado se detallará el estudio experimental. En primer lugar, se adjuntará una ilustración para ver un ejemplo de salida según la implementación para $n = 8$.

```
=====
EOA
=====
BIENVENIDOS A LA PRÁCTICA 01 DE LOSJIMENEZ
=====

Selecciona una opción:
1. Ejecutar el caso 1
2. Ejecutar el caso 2 Par
3. Ejecutar el caso 2 Impar
4. Ejecutar el caso 1 con 2 llamadas Recursivas
5. Estudio Experimental: caso 1
6. Estudio Experimental: caso 2 Par
7. Estudio Experimental: caso 2 Impar
8. Estudio Experimental: caso 1 2 llamadas Recursivas
9. Salir
Opción: 1
Caso 01 - n es potencia de 2
Por favor, introduce un número que sea potencia de 2: 8
+---+---+---+---+---+---+
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
+---+---+---+---+---+---+
| 1 | 4 | 3 | 6 | 7 | 8 | 5 |
+---+---+---+---+---+---+
| 4 | 1 | 2 | 7 | 8 | 5 | 6 |
+---+---+---+---+---+---+
| 3 | 2 | 1 | 8 | 5 | 6 | 7 |
+---+---+---+---+---+---+
| 6 | 7 | 8 | 1 | 4 | 3 | 2 |
+---+---+---+---+---+---+
| 5 | 8 | 7 | 2 | 1 | 4 | 3 |
+---+---+---+---+---+---+
| 8 | 5 | 6 | 3 | 2 | 1 | 4 |
+---+---+---+---+---+---+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 |
+---+---+---+---+---+---+
```

Ilustración 4. Ejemplo de salida caso 01 para $n = 8$.

Para el estudio experimental se han tomado las medidas a partir de un n significativo, ya que por debajo de ese n no hemos tenido valores significativos, debido a que para la práctica se pedían medidas en milisegundos y con valores tan bajos no teníamos valores significantes. Los valores explorados han sido los

siguientes para n potencia de 2: 256,512,1024,2048,4096,8192,16384,32768,65536. A partir de 65536 para el siguiente número potencia de 2 no se ha podido lanzar debido a las limitaciones del hardware actual. Los datos resultantes según la exploración inicial han sido las siguientes:

n	t (milisegundos)	Log(n)	Log(t)	tamaño (MB)
256	0,11111	2,408239965	-0,954242509	0,24905
512	0,22222	2,709269961	-0,653212514	0,99808
1024	1,00000	3,010299957	0	3,99612
2048	3,33333	3,311329952	0,522878745	15,99222
4096	14,22222	3,612359948	1,15296746	63,98441
8192	57,33333	3,913389944	1,758407192	255,96878
16384	228,66667	4,214419939	2,359202861	1023,93753
32768	933,88889	4,515449935	2,970295208	4095,87503
65536	3822,33333	4,816479931	3,582328558	16383,75003

Tabla 3: Caso 1 potencia de dos

La gráfica siguiente detallan las relaciones entre los valores:

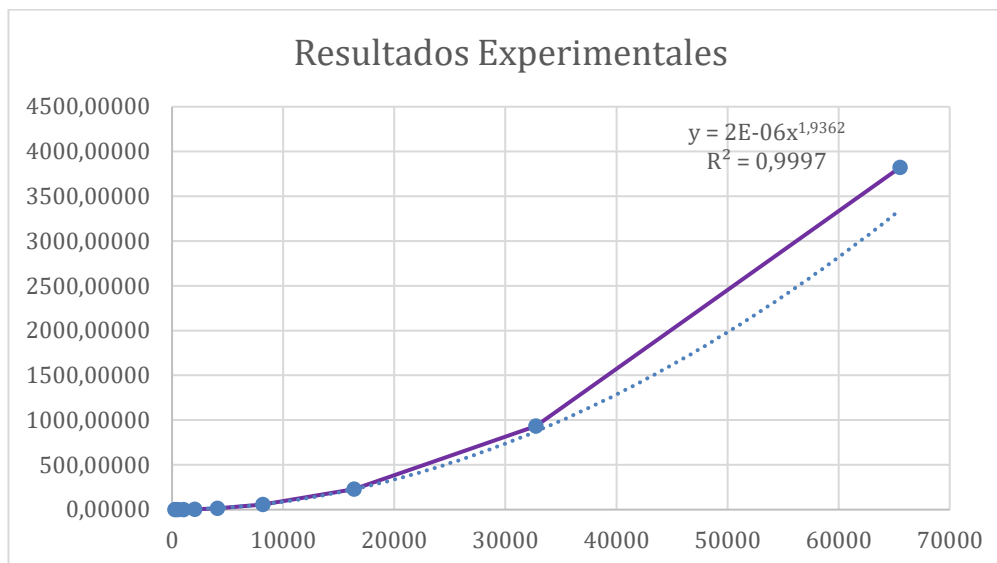


Ilustración 5. Resultados experimentales primer caso.

Como podemos ver, aunque hemos observado una concentración de puntos de datos cerca del origen y una dispersión más amplia a medida que los valores de x aumentan, esta distribución desigual puede indicar una necesidad de más puntos de datos en el rango intermedio para validar la consistencia del modelo polinómico a través del espectro completo. A pesar de la falta de datos más amplios debido a limitaciones computacionales, la función esperada sería modelada por la función $n^{1,9362}$. Sin embargo, nuestros valores experimentales aún muestran un rendimiento por encima de lo esperado, lo que sugiere que, aunque nuestro análisis teórico está incompleto debido a las limitaciones de cálculo, el algoritmo se comporta de manera consistente con nuestras expectativas teóricas.

Seguidamente podemos ver que el orden espacial es de $O(n^2)$ como podemos recoger en la siguiente gráfica:

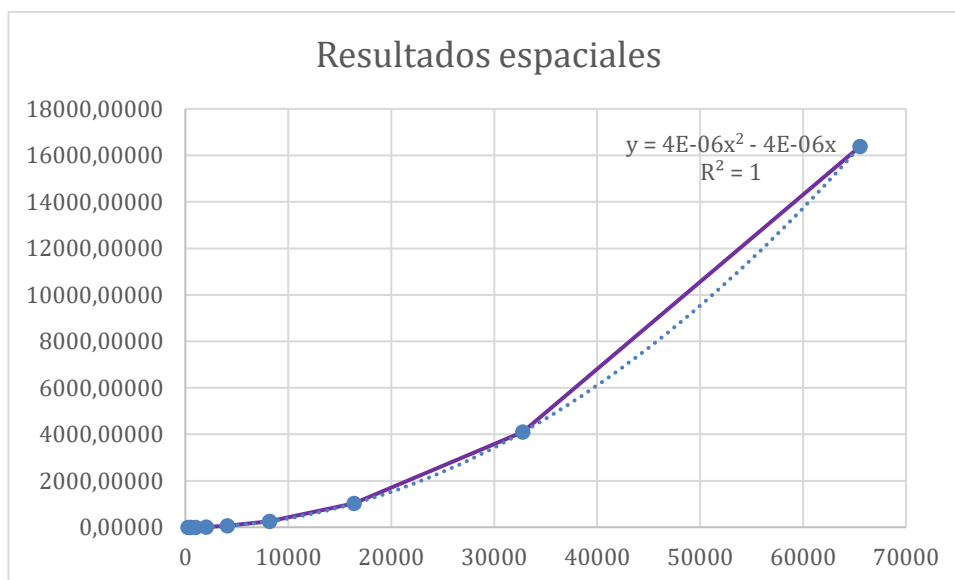


Ilustración 6. Orden espacial caso 1.

3.2. Caso 2.

Para el siguiente apartado, se va a detallar el segundo caso de la práctica, donde n es un número que no es potencia de dos y puede ser par o impar. En ambos casos, aplicaremos la técnica de Divide y Vencerás de simplificación (solo una llamada recursiva) para construir una tabla donde cada fila pertenece a n jugadores y cada columna representa los días en los que cada jugador juega $n - 1$ días en el caso par, y n días en el caso impar. A continuación, se ilustra un ejemplo de la ejecución.

	d1	d2	d3
j1	2	3	0
j2	1	0	3
j3	0	1	2

$m = 3$

	d1	d2	d3	d4	d5
j1	2	3	0		
j2	1	0	3		
j3	0	1	2		
j4	5	6	0		
j5	4	0	6		
j6	0	4	5		

cuadrantes 1º y 2º

	d1	d2	d3	d4	d5
j1	2	3	4	5	6
j2	1	5	3	6	4
j3	6	1	2	4	5
j4	5	6	1	3	2
j5	4	2	6	1	3
j6	3	4	5	2	1

cuadrantes 3º y 4º

Ilustración 7: Ejemplo ejecución caso 2

3.2.1. Estudio teórico.

En este apartado se va a realizar el estudio teórico.

- Caso Base: Cuando $n = 2$ se trata del caso base en el que solo hay 2 jugadores y simplemente se asignan valores correspondientes a la matriz. Esto tiene una complejidad constante de $O(1)$, ya que no depende del tamaño de la entrada.
- Caso Impar: Cuando n es impar, se realiza una llamada recursiva con $n+1$. Luego se eliminan los registros del jugador $n+1$ de la tabla. Este proceso implica dos bucles anidados que recorren la matriz de tamaño $n \times n$, lo que resulta en una complejidad de $O(n^2)$.
- Caso Par: Cuando n es par el proceso se divide en dos:
 - o Caso Par dentro de la recursión:
 - Se realiza una llamada recursiva con $m = n/2$

- Se llenan los tres cuadrantes de la matriz:
 - Cuadrante inferior izquierdo: Se llenan $m \times (m - 1)$ celdas.
 - Cuadrante superior derecho: Se llenan $m \times (n - m)$ celdas.
 - Cuadrante inferior derecho: Se llenan $m \times (n - m)$ celdas.
- Cada operación tiene una complejidad constante por lo que la complejidad total del caso par es $O(n^2)$.
- Caso Impar dentro de la recursión:
 - Se realiza una llamada recursiva con $m = n/2$.
 - Se llenan tres cuadrantes de la matriz y se rellenan los ceros de los cuadrantes izquierdos:
 - Cuadrante inferior izquierdo: Se llenan $m \times m$ celdas.
 - Cuadrante superior derecho: Se llenan $m \times (n - m - 1)$ celdas.
 - Cuadrante inferior derecho: Se llenan $m \times (n - m - 1)$ celdas.
 - Se rellenan los ceros de los cuadrantes izquierdos.
 - Cada operación tiene una complejidad constante, por lo que la complejidad total del caso impar es $O(n^2)$.

La complejidad total de la llamada recursiva se reduce a la mitad en cada nivel de recursión, aproximadamente logarítmica en función de n . Y dentro de cada llamada recursiva las operaciones tienen una complejidad de $O(n^2)$.

Por lo tanto, la complejidad total del algoritmo es aproximadamente $O(n^2 \log n)$, ya que la llamada recursiva se realiza aproximadamente logarítmicamente en función de n y dentro de cada llamada recursiva, las operaciones tienen una complejidad de $O(n^2)$.

El orden espacial del algoritmo aborda dos escenarios posibles, cuando el número de jugadores es impar o es par. Para ambos casos, el espacio requerido por el algoritmo es principalmente para almacenar la matriz bidimensional, cuyo tamaño es $n \times n$. A medida que el algoritmo opera en distintas partes de la matriz, utiliza variables adicionales como m para representar la mitad del número de jugadores. Dado que el espacio ocupado por la matriz aumenta en n^2 , el orden espacial del algoritmo sigue siendo $O(n^2)$, lo que significa que el espacio utilizado aumenta de forma cuadrática con respecto al tamaño de la entrada n .

3.2.2. Estudio experimental.

En este apartado se detallará el estudio experimental. En primer lugar, se adjuntará una ilustración para ver un ejemplo de salida según la implementación para $n = 11$.

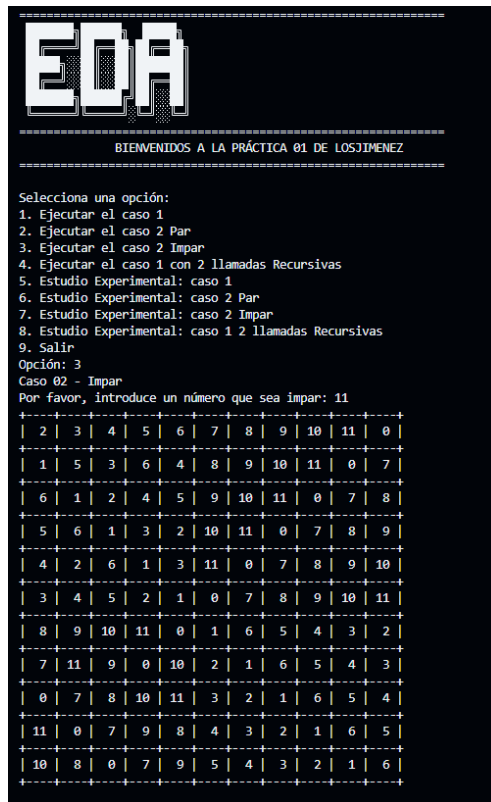


Ilustración 8. Ejemplo de salida caso 02 para $n = 11$

Como en el caso anterior se han tomado medidas para un n significativo.

Para el caso par los valores explorados han sido: 258, 4100, 7936, 11776, 15616, 19456, 23296, 27136, 30976, 34816, 38656, 42496, 46336, 50176, 54016, 57856, 61696, 65538, 69376, 73216.

Los datos resultantes han sido:

n	t (milisegundos)	Log(n)	Log(t)	tamaño (MB)
258	0,22222222	2,411619706	-0,65321251	0,249053955
4098	8,77777778	3,612571954	0,94338458	63,98440552
7936	31,44444444	3,899601659	1,49754393	240,2197571
11776	69,22222222	4,070997797	1,84024554	528,9551086
15616	123,7777778	4,1935698	2,09264268	930,1904602
19456	192,6666667	4,289053558	2,28480658	1443,925812
23296	269,8888889	4,367281358	2,43118501	2070,161163
27136	370,33333	4,433545831	2,5685928	2808,89651
30976	494,77778	4,491025336	2,69441019	3660,13187
34816	606,77778	4,541778874	2,78302967	4623,86722
38656	745,66667	4,587216913	2,87254473	5700,10257
42496	901,66667	4,628348053	2,95504601	6888,83792
46336	1073,33333	4,66591854	3,03073462	8190,07327
50176	1255,11111	4,700496037	3,09868217	9603,80862
54016	1463,77778	4,732522421	3,16547515	11130,04398
57856	1647,33333	4,762348404	3,21678149	12768,77933
61696	1909,33333	4,790257008	3,28088175	14520,01468
65538	2104,33333	4,816493184	3,32311453	16383,75003
69376	2440,77778	4,841209256	3,38752824	18359,98538

73216	2726,33333	4,864605998	3,43557895	20448,72073
-------	------------	-------------	------------	-------------

Tabla 4: Caso 2 Par

La gráfica siguiente detallan las relaciones entre los valores:

Resultados Experimentales

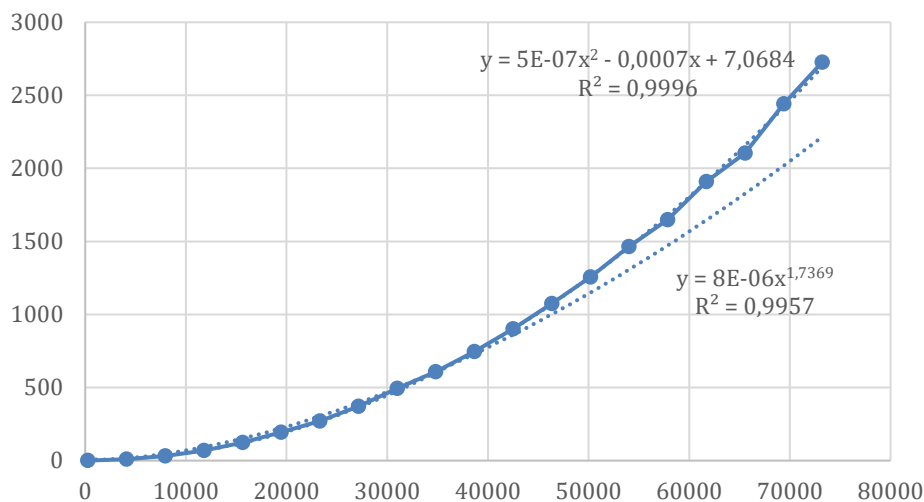


Ilustración 9: Resultados experimentales segundo caso par

Tal y como se puede observar en la gráfica, el algoritmo concuerda con un R^2 del 0,9996 con la línea de tendencia polinómica de grado 2, mientras que cuando se observa la línea de tendencia potencial, R^2 es de 0,9957.

Teniendo en cuenta que el estudio teórico del algoritmo ha concluido que el orden de complejidad es $O(n^2 \log n)$ y lo que se está obteniendo se acerca más a $O(n^2)$ se puede decir que el resultado puede ser como es por varias razones, como la implementación del algoritmo, la influencia de factores externos como la optimización del compilador o la gestión de memoria del sistema operativo, datos experimentales limitados, porque a lo mejor los datos recopilados no son suficientes para reflejar correctamente el comportamiento del algoritmo.

Para el caso impar los valores explorados han sido: 257, 4097, 7937, 11777, 15617, 19457, 23297, 27137, 34817, 42497, 50177, 54017, 61697, 70537, 77357.

Los datos resultantes han sido:

n	t (milisegundos)	Log(n)	Log(t)	tamaño (MB)
257	1,222222222	2,409933123	0,087150176	0,25100708
4097	22,33333333	3,612465964	1,348953548	64,01565552
7937	72,44444444	3,89965638	1,860005086	240,280304
11777	157,8888889	4,071034675	2,198351568	529,0449524
15617	289,6666667	4,19359761	2,461898522	930,3096008
19457	438,3333333	4,289075879	2,641804498	1444,074249
23297	624,1111111	4,3673	2,795261914	2070,338898
27137	865,4444444	4,433561835	2,937239194	2809,103546
34817	1409,222222	4,541791347	3,148979483	4624,132843
42497	2137,444444	4,628358273	3,329894836	6889,16214

50177	2954,666667	4,700504692	3,470508493	9604,191437
54017	3328,666667	4,732530461	3,522270307	11130,45609
61697	4394,444444	4,790264047	3,642903978	14520,48538
65537	4821,222222	4,816486557	3,68315715	16384,25003
70537	5945	4,848416985	3,774151859	18979,63654
77357	6808,555556	4,888499619	3,833054986	22827,25563

Tabla 5: Caso 2 Impar

Con respecto al caso en el que n sea impar la gráfica se comporta de la misma manera:

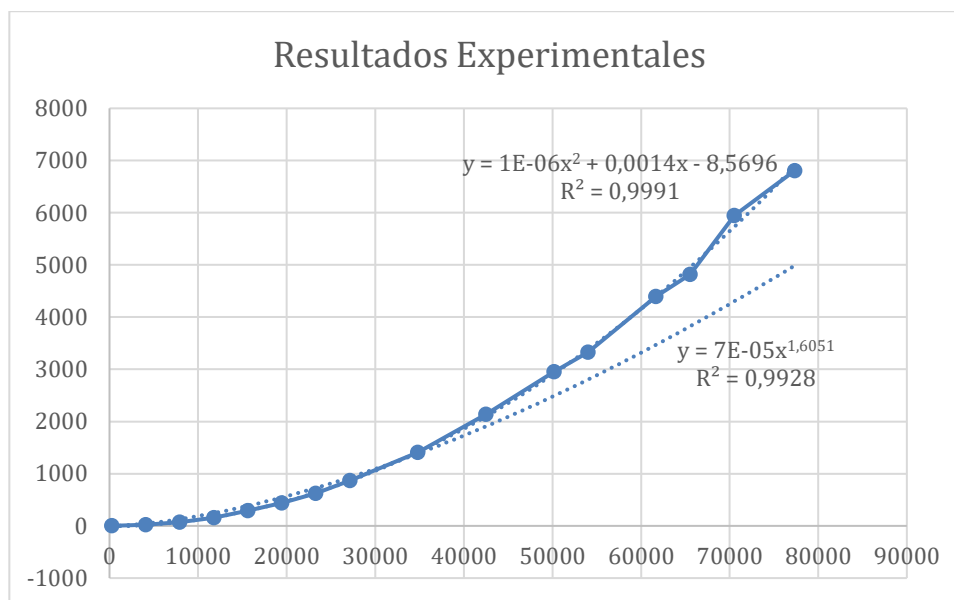


Ilustración 10: Resultados experimentales segundo caso impar

La conclusión que se puede sacar es la misma que se ha sacado con respecto al caso 2 par, que se está obteniendo un resultado más cerca de $O(n^2)$ que de $O(n^2 \log n)$.

En el caso del orden espacial podemos ver que es $O(n^2)$, como se detalla en la siguiente ilustración para ambos casos, par e impar.

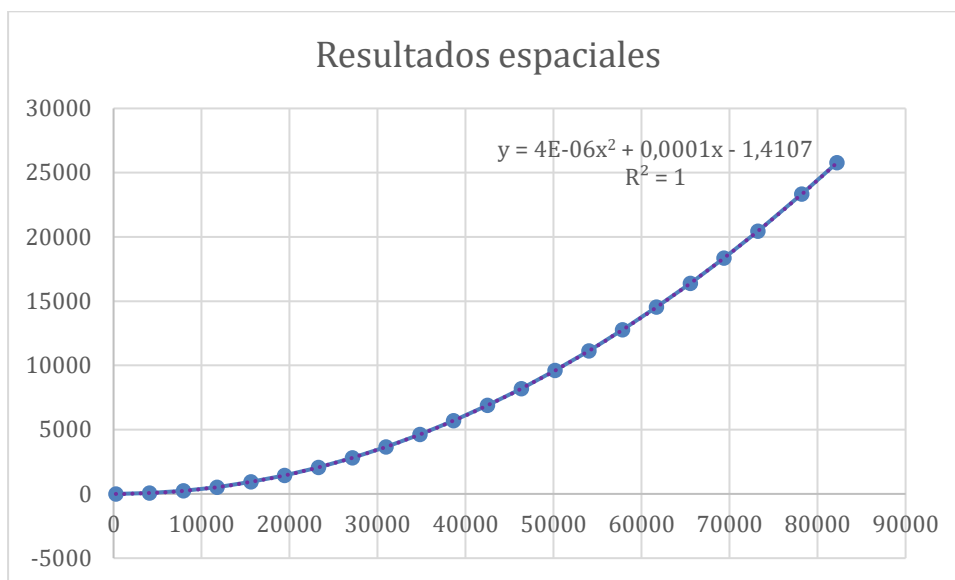


Ilustración 11. Orden espacial caso 2 par.

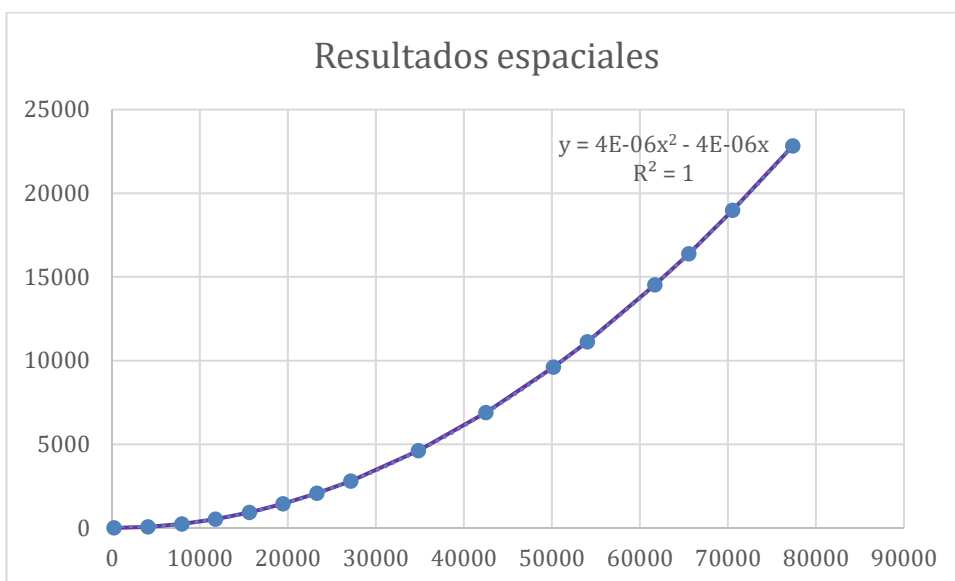


Ilustración 12. Orden espacial caso 2 impar.

3.3. Caso 1 con doble llamada recursiva.

En este apartado vamos a detallar la realización del ejercicio del caso 3 propuesto en la práctica. Para este caso, se propone una variante del primer caso realizado (véase apartado 4). En este caso se plantea una variante que consta de una implementación sin simplificación, para este caso haremos uso del esquema de Divide y Vencerás general (dos llamadas recursivas) para poder así dividir la matriz en dos mitades de igual tamaño, la siguiente modificación será que la solución se expresará en una matriz compuesta por filas (en este caso serán los días, siendo $n-1$) y las columnas serán los jugadores n .

3.3.1. Estudio teórico.

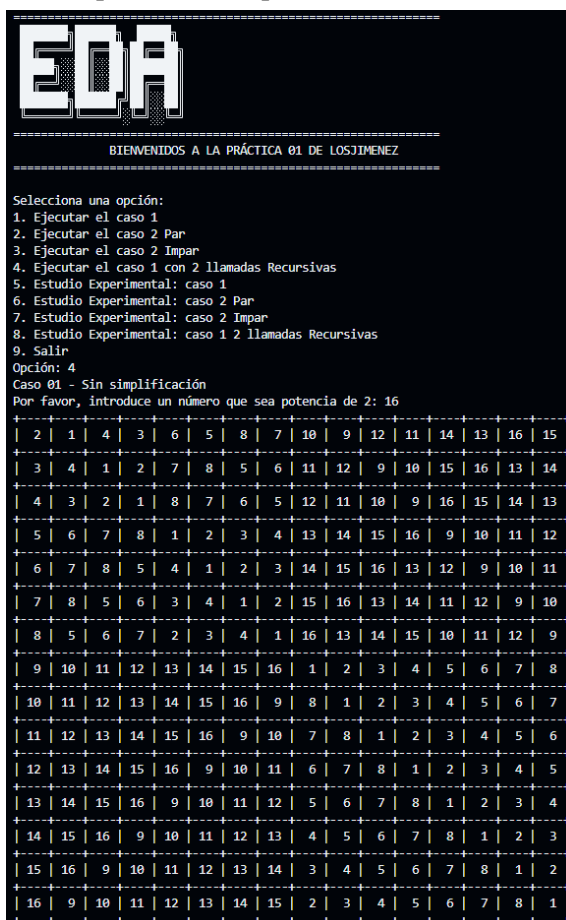
En este apartado se va a realizar el estudio teórico.

- Caso base: ($\text{jugadorFinal} == \text{jugadorInicial} + 1$). En este caso, se realiza una cantidad constante de operaciones, por lo que el orden de complejidad es $O(1)$.
- División recursiva: En cada llamada recursiva se realiza una cantidad constante de operaciones para calcular la mitad y las diferencias entre jugadores.
- Recursión: La recursión se realiza $\log_2(n)$ veces, donde n es el número total de jugadores. La complejidad es $O(\log n)$.
- Método Combinar:
 - o Hay dos bucles anidados. El bucle externo itera a través de los días del torneo, que es $n-1$. El bucle interno itera a través de los jugadores de la primera mitad, que es $n/2$. La complejidad es $O(n^2)$.
- Complejidad Total: Dado que la complejidad que hay es $O(\log n)$ y $O(n^2)$, la complejidad total del algoritmo es $O(\log n) + O(n^2)$. Por lo que cuando n crece significativamente, la complejidad es $O(n^2)$.

El espacio requerido por el algoritmo es principalmente para almacenar la matriz bidimensional. Dado que el espacio ocupado por la matriz aumenta con n^2 , el orden espacial del algoritmo sigue siendo $O(n^2)$, lo que significa que el espacio utilizado aumenta cuadráticamente con respecto al tamaño de la entrada n .

3.3.2. Estudio experimental.

En este apartado se detallará el estudio experimental. En primer lugar, se adjuntará una ilustración para ver un ejemplo de salida según la implementación para $n = 16$.



```

=====
EDC
=====
BIENVENIDOS A LA PRÁCTICA 01 DE LOSJIMENEZ
=====

Selecciona una opción:
1. Ejecutar el caso 1
2. Ejecutar el caso 2 Par
3. Ejecutar el caso 2 Impar
4. Ejecutar el caso 1 con 2 llamadas Recursivas
5. Estudio Experimental: caso 1
6. Estudio Experimental: caso 2 Par
7. Estudio Experimental: caso 2 Impar
8. Estudio Experimental: caso 1 2 llamadas Recursivas
9. Salir
Opción: 4
Caso 01 - Sin simplificación
Por favor, introduce un número que sea potencia de 2: 16

+-----+
| 2 | 1 | 4 | 3 | 6 | 5 | 8 | 7 | 10 | 9 | 12 | 11 | 14 | 13 | 16 | 15 |
+-----+
| 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 | 11 | 12 | 9 | 10 | 15 | 16 | 13 | 14 |
+-----+
| 4 | 3 | 2 | 1 | 8 | 7 | 6 | 5 | 12 | 11 | 10 | 9 | 16 | 15 | 14 | 13 |
+-----+
| 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 13 | 14 | 15 | 16 | 9 | 10 | 11 | 12 |
+-----+
| 6 | 7 | 8 | 5 | 4 | 1 | 2 | 3 | 14 | 15 | 16 | 13 | 12 | 9 | 10 | 11 |
+-----+
| 7 | 8 | 5 | 6 | 3 | 4 | 1 | 2 | 15 | 16 | 13 | 14 | 11 | 12 | 9 | 10 |
+-----+
| 8 | 5 | 6 | 7 | 2 | 3 | 4 | 1 | 16 | 13 | 14 | 15 | 10 | 11 | 12 | 9 |
+-----+
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
+-----+
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 9 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
+-----+
| 11 | 12 | 13 | 14 | 15 | 16 | 9 | 10 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 |
+-----+
| 12 | 13 | 14 | 15 | 16 | 9 | 10 | 11 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 |
+-----+
| 13 | 14 | 15 | 16 | 9 | 10 | 11 | 12 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 |
+-----+
| 14 | 15 | 16 | 9 | 10 | 11 | 12 | 13 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 |
+-----+
| 15 | 16 | 9 | 10 | 11 | 12 | 13 | 14 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 |
+-----+
| 16 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 |
+-----+

```

Ilustración 13: Ilustración 5: Ejemplo de salida caso 01 con doble llamada recursiva para $n = 16$

Para el caso con dos llamadas recursivas los valores explorados han sido: 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536.

Los datos resultantes han sido:

n	t (milisegundos)	log (n)	log (t)	tamaño (MB)
256	0,22222	2,40824	-0,65321	0,24905
512	0,22222	2,70927	-0,65321	0,99808
1024	0,44444	3,01030	-0,35218	3,99612
2048	3,00000	3,31133	0,47712	15,99222
4096	11,88889	3,61236	1,07514	63,98441
8192	46,44444	3,91339	1,66693	255,96878
16384	186,66667	4,21442	2,27107	1023,93753
32768	770,88889	4,51545	2,88699	4095,87503
65536	3175,55556	4,81648	3,50182	16383,75003

Tabla 6: Caso 1 con dos llamadas recursivas

La gráfica siguiente detallan las relaciones entre los valores:

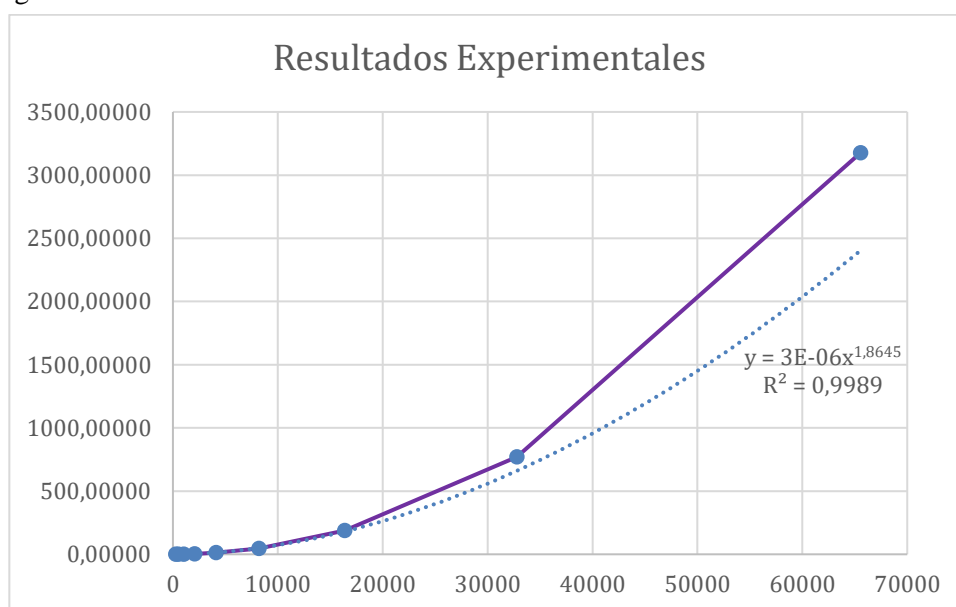


Ilustración 14: Resultados experimentales primer caso dos llamadas recursivas

El estudio teórico y experimental del algoritmo proporciona una visión integral de su rendimiento. Teóricamente, al analizar la estructura y operaciones del algoritmo, pudimos deducir su complejidad asintótica como $O(n^2)$, lo que sugiere un crecimiento cuadrático en relación con el tamaño de entrada n . Este análisis teórico nos brinda una comprensión fundamental de cómo el algoritmo escala con tamaños de entrada cada vez mayores. Por otro lado, el estudio experimental complementa esta comprensión al proporcionar una confirmación empírica del rendimiento del algoritmo. La ecuación $3 \times 10^{-6} x^{1,8645}$ obtenida experimentalmente coincide con la tendencia superlineal prevista teóricamente, respaldando la idea de que el algoritmo tiene una complejidad significativa y no lineal.

En el caso del orden espacial podemos detallar en la siguiente ilustración como es $O(n^2)$.

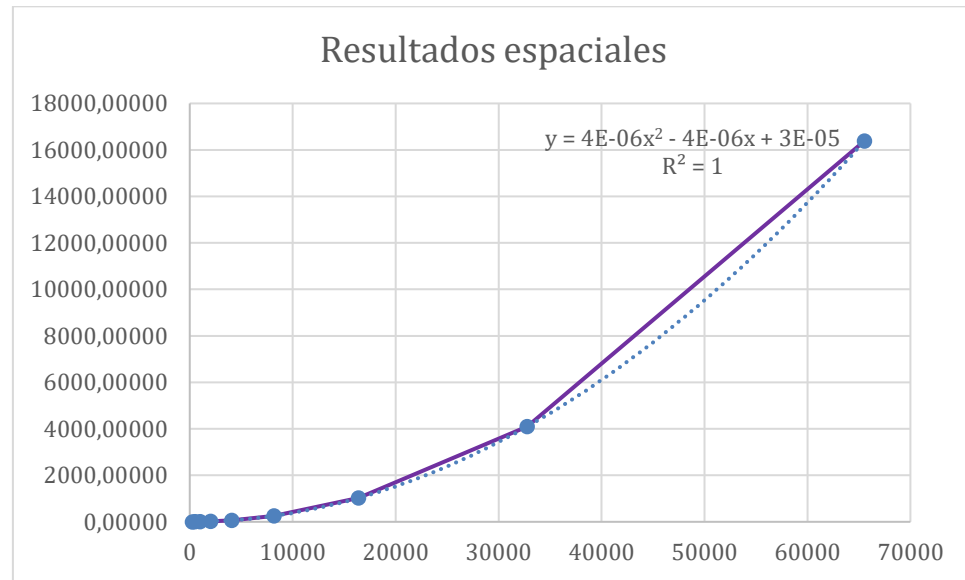


Ilustración 15. Orden espacial caso 1 doble llamada recursiva.

A. Anexo.

A.1. Diseño del código.

Diagramas de clases.

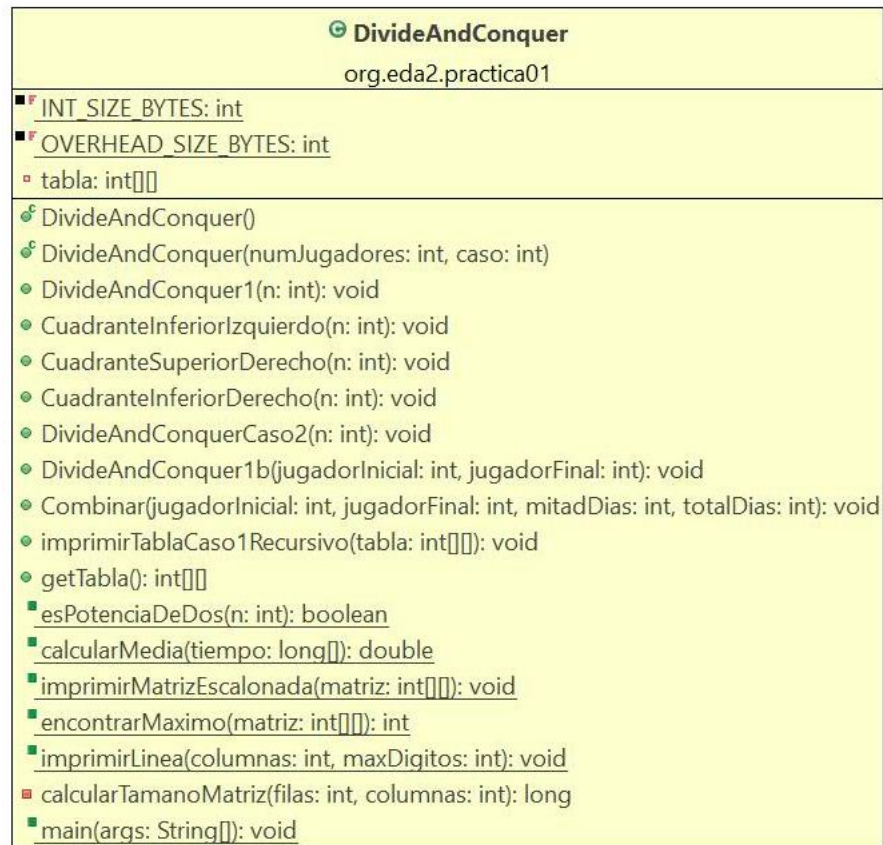


Ilustración 16. Diagrama de Clases.


<div>  DivideAndConquerTest org.eda2.practica01 </div>
<ul style="list-style-type: none"> • testCaso1Base(): void • testCaso1ConCuatroJugadores(): void • testCaso1ConOchoJugadores(): void • testCaso1ConDieciseisJugadores(): void • testCaso1ConSesentayCuatroJugadores(): void • testCaso2BasePar(): void • testCaso2ImparConTresJugadores(): void • testCaso2ParConSeisJugadores(): void • testCaso2ImparConSieteJugadores(): void • testCaso2ImparConNueveJugadores(): void • testCaso2ParConTreceJugadores(): void • testCaso2ConQuinceJugadores(): void • testCaso1Baseb(): void • testCaso1ConCuatroJugadoresRecurso(): void • testCaso1ConOchoJugadoresRecurso(): void • testCaso1ConDieciseisJugadoresRecurso(): void • imprimirTabla(tabla: int[]): void • imprimirTablaCaso1Recurso(tabla: int[]): void

Ilustración 17. Diagrama de Clases para las pruebas.

A.2. Esquema archivos fuente.

El esquema de archivos fuente es el siguiente:

DivideAndConquer.java

Ruta: main\java\org\eda2\practica01\DivideAndConquer.java

Descripción: Este archivo contiene la implementación del algoritmo "Divide y Vencerás". Incluye diferentes casos teniendo en cuenta especificaciones concretas. Además, contiene el menú principal con los cálculos necesarios para medir los tiempos de ejecución del algoritmo, así como una función para mostrar el correcto funcionamiento de este.

DivideAndConquerTest.java

Ruta: test\java\org\eda2\practica01\DivideAndConquerTest.java

Descripción: Este archivo es un banco de pruebas diseñado para comprobar el correcto funcionamiento del algoritmo "Divide y Vencerás". Contiene casos de prueba que verifican que el algoritmo produce los resultados esperados bajo diversas condiciones y entradas.

Memoria.docx

Ruta: docs\practica01\Memoria.docx

Descripción: Este archivo contiene la memoria de la práctica. Probablemente incluya una descripción detallada del problema abordado, el enfoque utilizado para resolverlo, los resultados obtenidos y cualquier otra información relevante relacionada con la práctica.

PR1 Estudio Experimental.xlsx

Ruta: docs\practica01\PR1_Estudio_Experimental.xlsx



Descripción: Este archivo Excel alberga los cálculos y datos recopilados durante el estudio experimental realizado como parte de la práctica. Puede incluir tablas, gráficos u otros elementos visuales que representen los resultados de las pruebas realizadas.

PR1 Tareas a Repartir.xlsx

Ruta: docs\practica01\PR1_Tareas_a_Repartir.xlsx

Descripción: Este archivo Excel contiene la distribución de tareas asignadas a cada uno de los compañeros que trabajan en la práctica. Incluye una lista de tareas específicas y las personas responsables de completar cada tarea.



B. Bibliografía.

Temario asociado a la asignatura.

<https://dis.um.es/~nmarin/transparencias-divide-y-venceras-AED-II.pdf>