



REPORTE:

# Ejercicio final

ASIGNATURA:

**Técnicas algorítmicas**

**Erick Daniel Reyes Torrecilla**

MATRÍCULA: 230300738

PROGRAMA EDUCATIVO: **ING EN Datos e inteligencia organizacional**

PRESENTADO A:

**Emmanuel Morales Saavedra**

---

## Elección de algoritmo: divide y vencerás

La técnica de divide y vencerás implica dividir el problema en subproblemas más pequeños, resolverlos de manera independiente y combinar los resultados para construir la solución global. Aunque otros enfoques como los algoritmos voraces o la programación dinámica pueden ser más directamente aplicables, el método de divide y vencerás es adecuado para el Sudoku, este es el método más eficiente pero no el más rápido.

## Complejidad Computacional

La complejidad computacional esperada de este enfoque es alta, ya que el problema del Sudoku es NP-completo. Sin embargo, con optimizaciones como el backtracking y heurísticas, se pueden mejorar significativamente los tiempos de ejecución para tableros comunes.

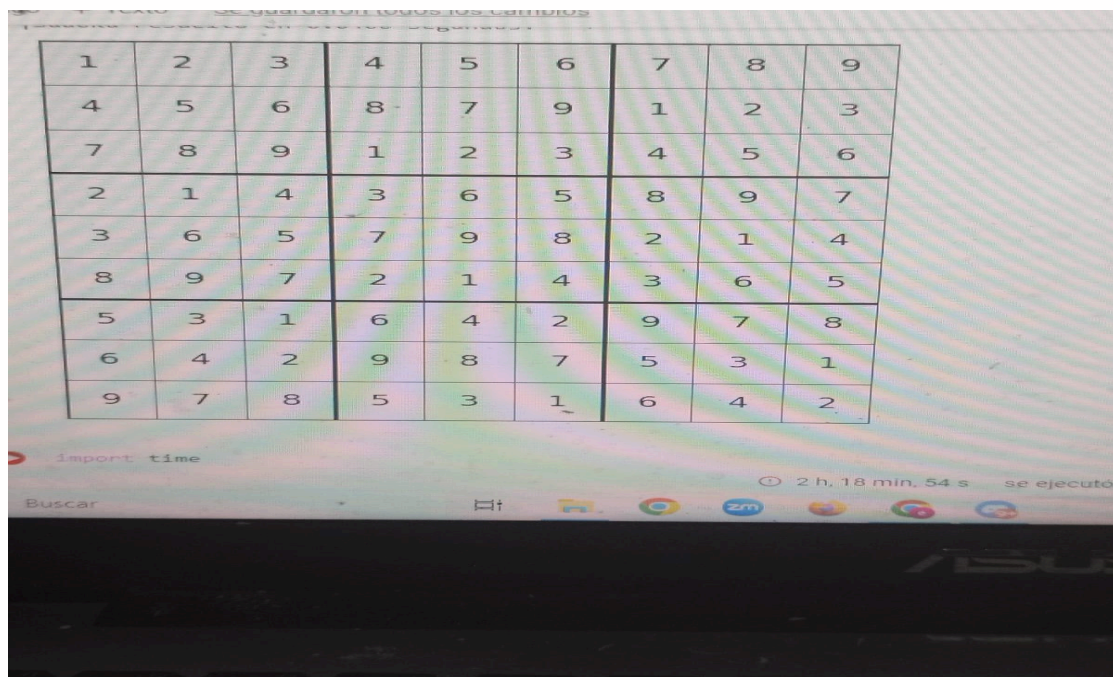
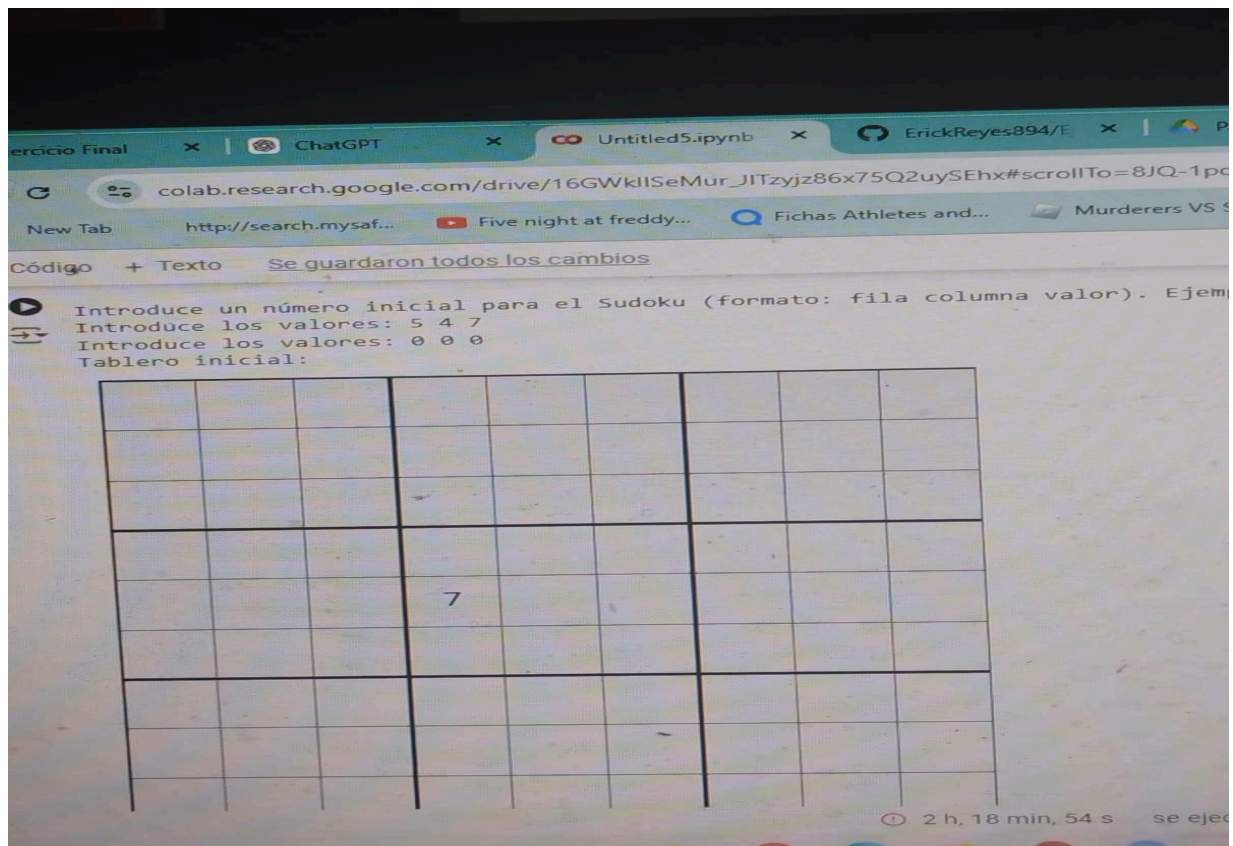
### Complejidad temporal:

- **Peor caso:**  $O(9n)$ , donde  $n$  es el número de celdas vacías. Sin embargo, con optimizaciones como forward checking, la búsqueda se reduce significativamente.

### Complejidad espacial:

- **Espacio adicional:**  $O(n)$ , debido a las llamadas recursivas en la pila.

## Demostración con ejemplo:



## Instrucciones para probar

- Copia el código en un archivo Python.
- Ejecuta el programa e ingresa el tablero de Sudoku fila por fila. Usa 0 para representar las celdas vacías.
- El programa resolverá el Sudoku y mostrará la solución junto con el tiempo de ejecución.

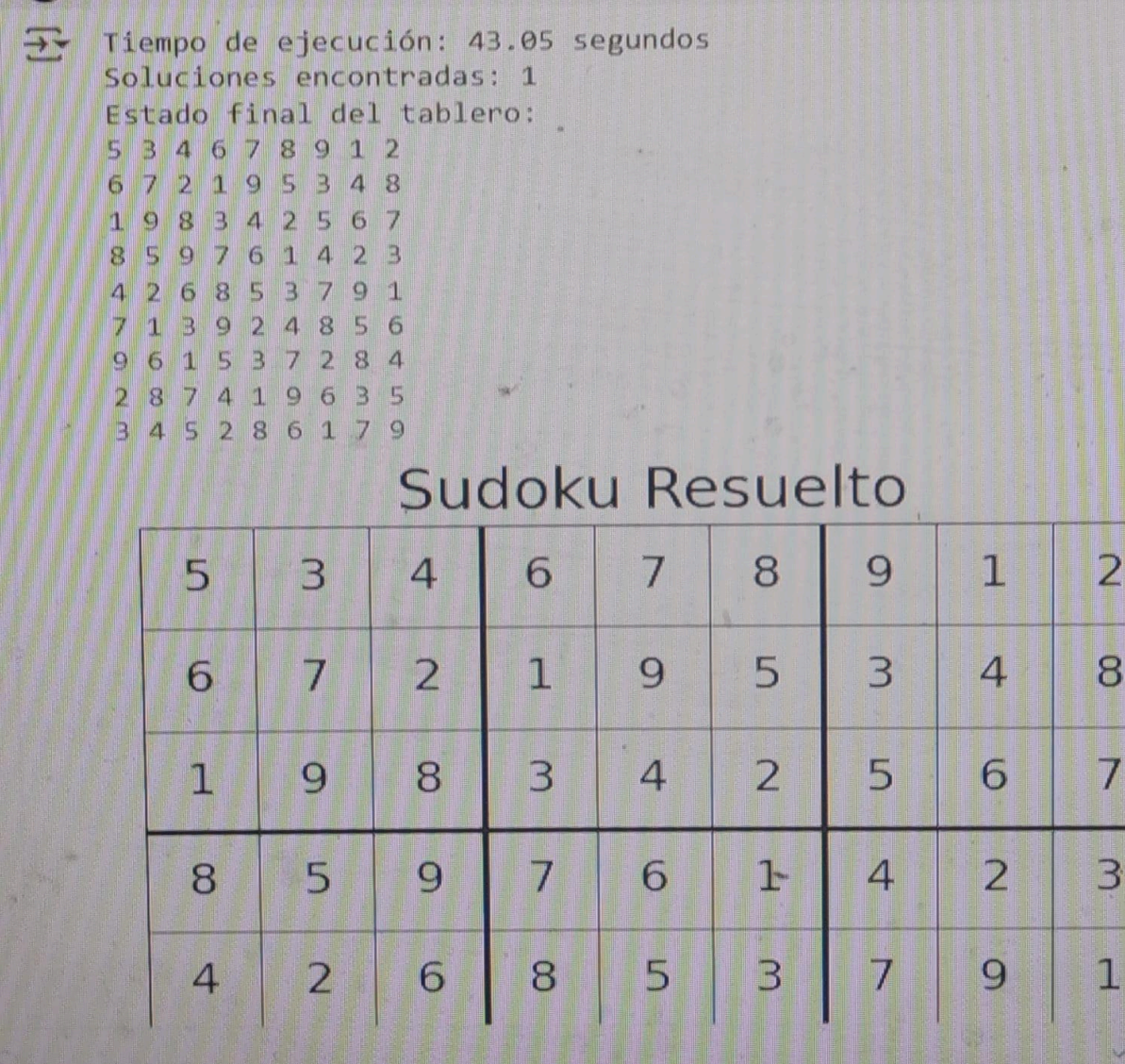
## Método rápido pero no óptimo: Algoritmos voraces

Los algoritmos voraces son rápidos y funcionan bien para problemas donde las decisiones locales conducen directamente a una solución global óptima. Sin embargo, el Sudoku no cumple con estas condiciones debido a su naturaleza altamente interdependiente

### Complejidad:

1. **Temporal:** En el peor caso, el algoritmo deberá explorar todas las celdas y probar cada número posible, lo que resulta en una complejidad aproximada de  $O(9n)$ , donde  $n$  es el número de celdas vacías.
2. **Espacial:** El espacio requerido es  $O(1)$ , ya que el algoritmo sólo necesita almacenar el tablero.



**Demostración con ejemplo:**

→ Tiempo de ejecución: 43.05 segundos  
Soluciones encontradas: 1  
Estado final del tablero:

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

**Sudoku Resuelto**

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1

**Instrucciones para probar**

- Copia el código del archivo .txt con el nombre de la técnica deseada.
- Ejecuta el programa de 30 segundos a 3 minutos .
- El programa resolverá el Sudoku y mostrará la solución junto con el tiempo de ejecución

**Método no óptimo y algo lento:** La programación dinámica no es la técnica más común para resolver Sudokus, ya que este problema no se descompone fácilmente en subproblemas independientes con solapamiento de soluciones. Además el proceso es lento y tardado pero se puede adaptar la técnica para aprovechar la idea de construir soluciones parcialmente, almacenando y reutilizando información sobre decisiones previas.

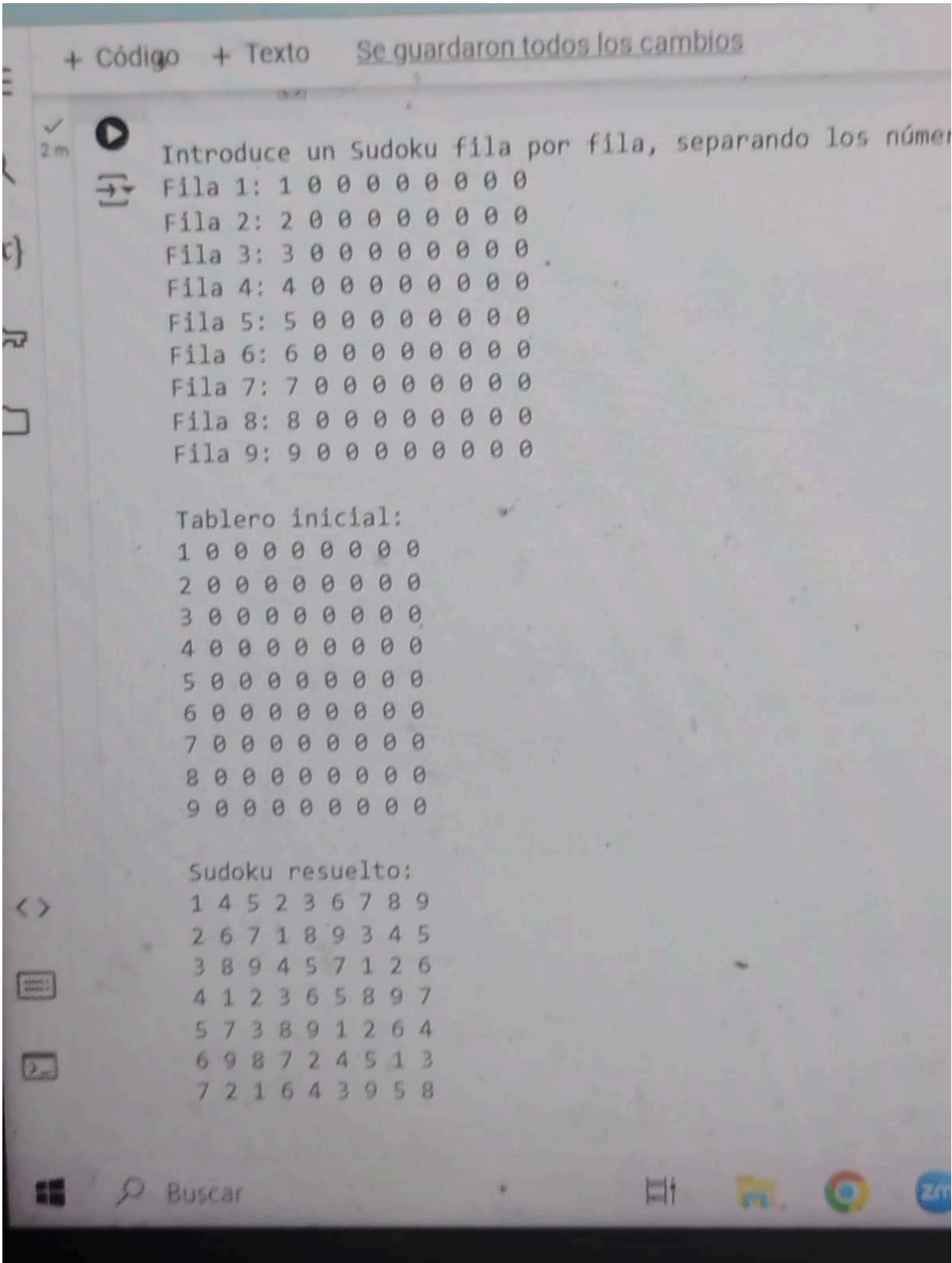
## Complejidad Computacional

### Complejidad Temporal

1. **Exploración del tablero:**
  - El algoritmo recorre todas las celdas del tablero (9x9) para identificar las vacías. En el peor caso, esto implica 81 celdas.
2. **Elección de valores:**
  - Por cada celda vacía, se prueban todos los números posibles (1 a 9). Esto introduce un factor multiplicativo de 999 por cada celda vacía.
3. **Total:**
  - El número máximo de celdas vacías es 81, y para cada una, se prueban 999 números y se valida cada intento. Esto da una complejidad aproximada:  $O(81)O(9^{81})O(81)$
  - Sin embargo, gracias a las restricciones del Sudoku, muchas combinaciones son rechazadas temprano, lo que reduce significativamente el espacio de búsqueda en la práctica.

### Complejidad Espacial

- La complejidad espacial depende de la profundidad de la pila de recursión.
- En el peor caso, la pila puede alcanzar una profundidad de 81, una por cada celda vacía.
- Por tanto, la complejidad espacial es  $O(81)$ , que es constante  $O(1)$  para tableros de tamaño fijo.

**Ejemplo:**

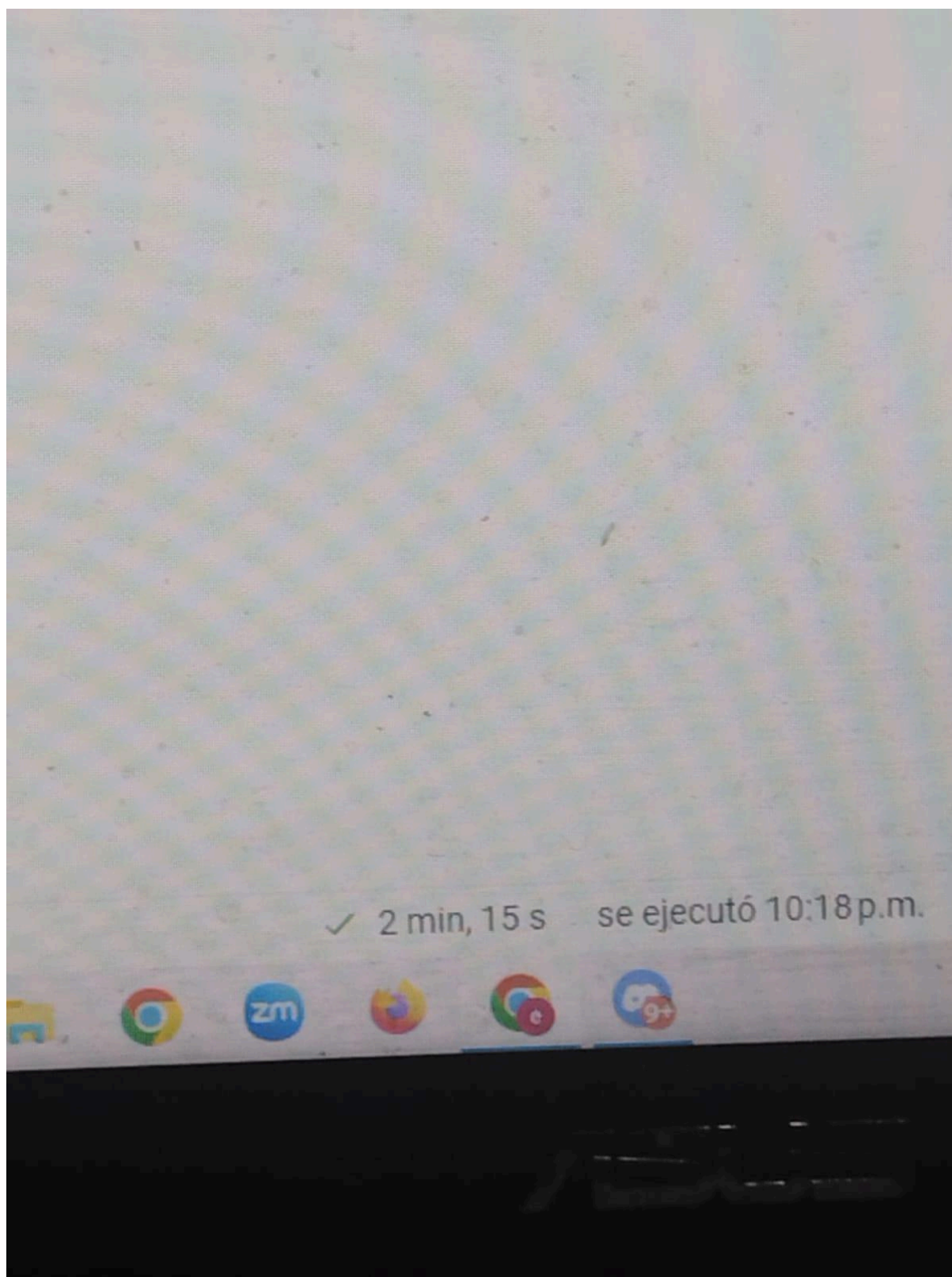
```
+ Código + Texto Se guardaron todos los cambios

2 m
Introduce un Sudoku fila por fila, separando los números
Fila 1: 1 0 0 0 0 0 0 0 0
Fila 2: 2 0 0 0 0 0 0 0 0
Fila 3: 3 0 0 0 0 0 0 0 0
Fila 4: 4 0 0 0 0 0 0 0 0
Fila 5: 5 0 0 0 0 0 0 0 0
Fila 6: 6 0 0 0 0 0 0 0 0
Fila 7: 7 0 0 0 0 0 0 0 0
Fila 8: 8 0 0 0 0 0 0 0 0
Fila 9: 9 0 0 0 0 0 0 0 0

Tablero inicial:
1 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0
8 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0

Sudoku resuelto:
1 4 5 2 3 6 7 8 9
2 6 7 1 8 9 3 4 5
3 8 9 4 5 7 1 2 6
4 1 2 3 6 5 8 9 7
5 7 3 8 9 1 2 6 4
6 9 8 7 2 4 5 1 3
7 2 1 6 4 3 9 5 8
```







## **Instrucciones para probar**

- Copia el código del los archivos.txt
- Ejecuta el programa e ingresa el tablero de Sudoku fila por fila. Usa 0 para representar las celdas vacías.
- El programa resolverá el Sudoku y mostrará la solución junto con el tiempo de ejecución