.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

Fecha de entrega: 18/09/2023

- Gauler, Ian Benjamin, Padrón: 109437
- Jauregui, Melina Belén, Padrón: 109524
- Tourne Passarino, Patricio, Padrón: 108725

Índice

1. Introducción

En el presente informe, se tiene como objetivo llevar a cabo un análisis del diseño implementado para abordar la problemática planteada por Scaloni, que consiste en la optimización máxima del tiempo total empleado en la realización de un análisis exhaustivo de sus rivales. A lo largo de este, se expondrán los criterios empleados y se proporcionará su justificación.

1.1. Información de la problemática

El enunciado propuesto, nos da la siguiente información:

- Cada compilado lo debe analizar Scaloni y alguno de sus ayudantes.
- El análisis del rival i le toma s_i tiempo a Scaloni y luego a_i tiempo al ayudante.
- Scaloni cuenta con n ayudantes, siendo n la cantidad de rivales a analizar. Además, cada ayudante puede ver los compilados completamente en paralelo a Scaloni y a los respectivos ayudantes.
- Al momento en que Scaloni haya terminado de analizar el i ésimo compilado, comenzará inmediatamente algún ayudante a analizarlo, para no desperdiciar ningún segundo.
- Sólo un ayudante verá el compilado, dado que no aporta mayor ganancia que dos ayudantes lo vean.

2. Análisis de diferentes soluciones propuestas

Vamos a proponer dos escenarios posibles para poder analizar sus soluciones. Estos dos escenarios los vamos a representar mediante un gráfico cada uno:

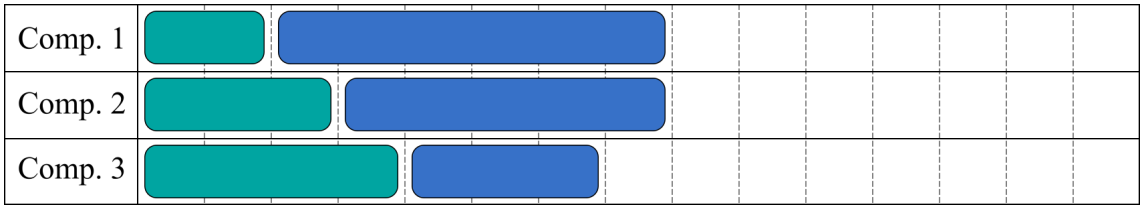


Figura 1: El tiempo que le tarda a los ayudantes ver cada compilado es inversamente proporcional a lo que le tarda a Scaloni.

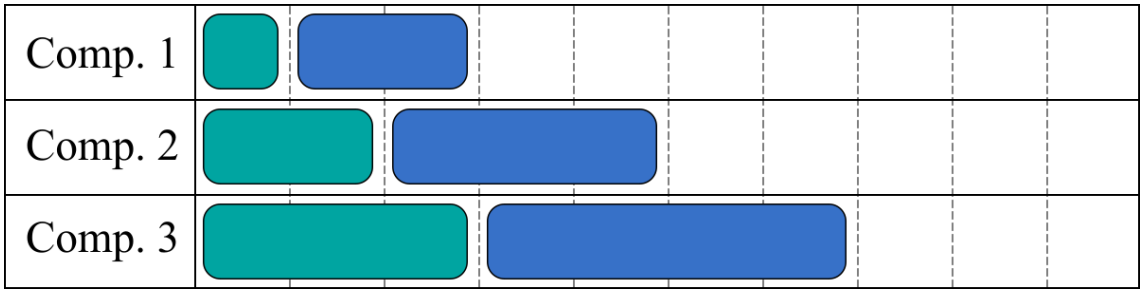
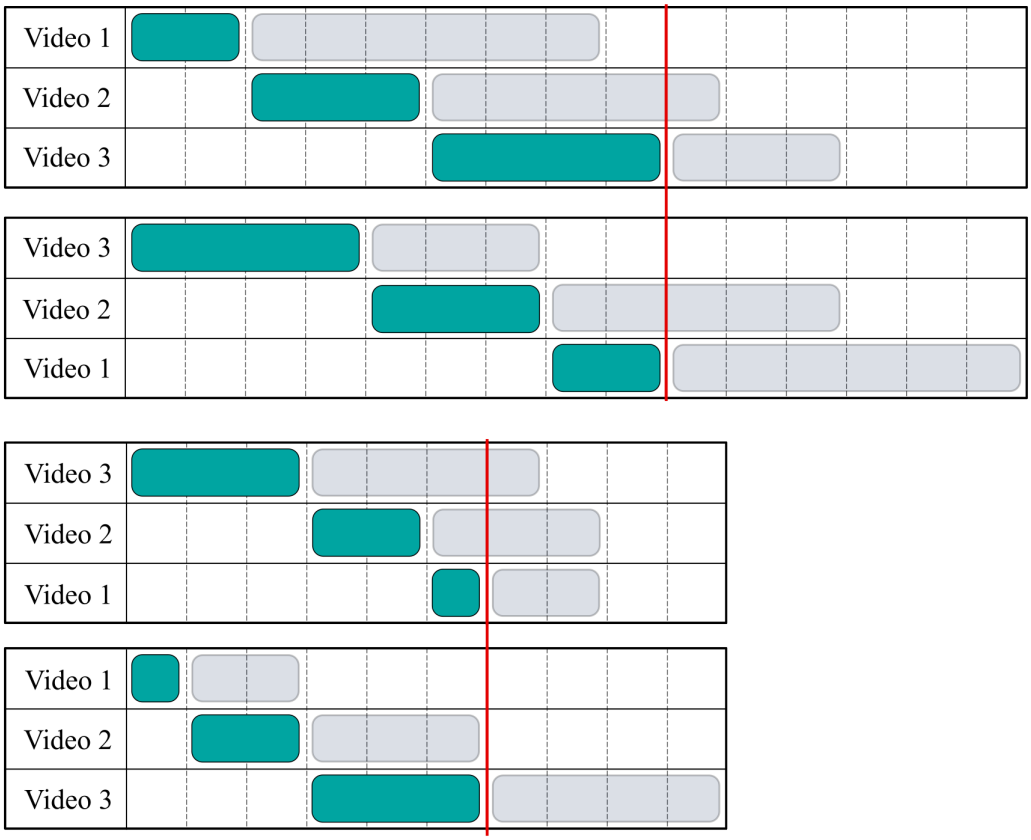


Figura 2: El tiempo que le tarda a los ayudantes ver cada compilado es proporcional a lo que le tarda a Scaloni.

Lo primero que notamos fue que el orden en que Scaloni visualiza los compilados no incide en el tiempo que le toma a él en finalizar la revisión de los mismos.



Tomando este análisis en consideración, concluimos que el mecanismo de ordenamiento debería únicamente depender de a_i y no de s_i.

3. Posibles algoritmos

Posterior a nuestro análisis, propusimos dos algoritmos:

3.1. Ordenando por a_i de forma creciente

Este algorítmo sigue las siguientes indicaciones:

- Ordenar los compilados de menor a mayor tiempo de análisis por parte de los ayudantes.
- Cada vez que termina Scaloni de ver un compilado, inmediatamente un ayudante es asignado a visualizarlo.

3.2. Ordenando por a_i de forma decreciente

- Ordenar los compilados de mayor a menor tiempo de análisis por parte de los ayudantes.

- Cada vez que termina Scaloni de ver un compilado, inmediatamente un ayudante es asignado a visualizarlo.

Comparemos cómo se desempeñan con nuestros dos algorítmos en cada escenario propuesto:

3.3. Resultados obtenidos del Caso 1

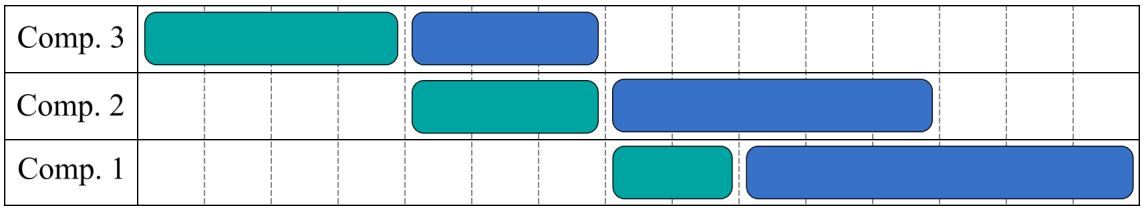


Figura 3: Caso 1 ordenado de forma creciente por a_i

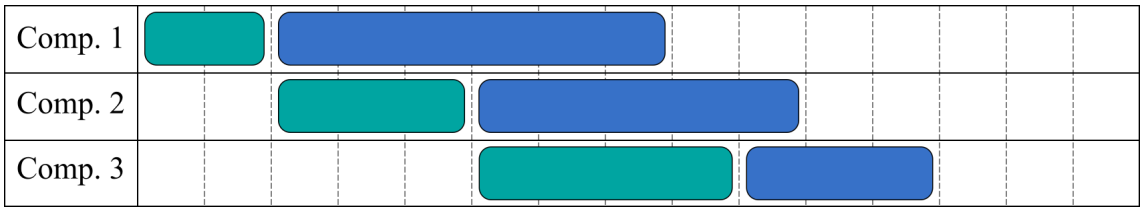


Figura 4: Caso 1 ordenado de forma decreciente por a_i

3.4. Resultados obtenidos del Caso 2

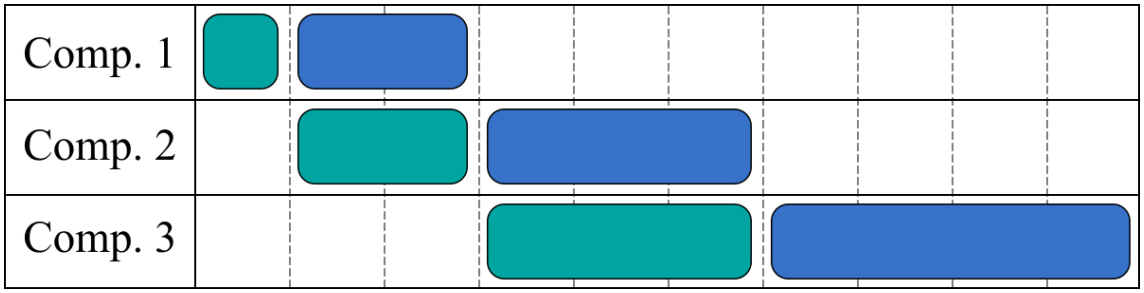


Figura 5: Caso 2 ordenado de forma creciente por a_i

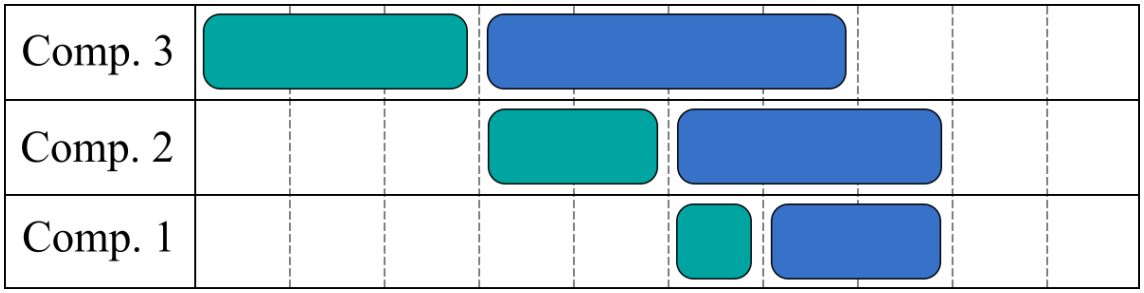


Figura 6: Caso 2 ordenado de forma decreciente por a_i

Como se puede observar, para ambos casos el ordenamiento decreciente resulta en un tiempo total menor.

4. Demostración

Procederemos a demostrar que ordenar los compilados por a_i de forma decreciente minimiza el tiempo total.

Llamando C al conjunto ordenado de n compilados, s_i el tiempo que le tardar visualizar a Scaloni el i-ésimo compilado, y a_i lo que le tarda a un ayudante. Llamaremos al tiempo que tarda Scaloni analizar los compilados hasta el k-ésimo S_k.

$$S_k = \sum_{i=1}^{k} s_i$$

Luego, el tiempo total que se tarda en analizar los compilados hasta el k-ésimo, en el orden en el que se los presenta, sigue la siguiente formula:

$$T_k = \begin{cases} s_1 + a_1 & k = 1 \\ \text{máx}\,(T_{k-1}, S_k + a_k) & k > 1 \end{cases}$$

Ordenar C para minimizar T_n requiere minimizar $\text{máx}\,(T_{n-1}, S_n + a_n)$. Como S_n es independiente del orden de C, $S_n + a_n$ se minimiza colocando último al compilado con el tiempo del ayudante más pequeño. Luego, T_{n-1} se minimiza de la misma manera, resultando en un algoritmo recursivo. Solo nos falta justificar por qué el orden que minimiza $S_n + a_n$ también minimiza $\text{máx}\,(T_{n-1}, S_n + a_n)$. Haremos esto partiendo de la asumpción de que se respeta el orden propuesto y de que modificarlo resulta en un T_n más grande.

- Si $\text{máx}\,(T_{n-1}, S_n + a_n) = S_n + a_n$ ($\implies S_n + a_n \geq T_{n-1}$)
  Es el caso más trivial, ya que minimizar $S_n + a_n$ también minimiza $\text{máx}\,(T_{n-1}, S_n + a_n)$.
- Si $\text{máx}\,(T_{n-1}, S_n + a_n) = T_{n-1}$ ($\implies T_{n-1} \geq S_n + a_n$)
  Este caso implica que existe un compilado j tal que $a_j \geq \sum_{i=j+1}^{n} s_i + a_n \implies a_j > a_n$. Colocar a este compilado último causaría que $T'_n = S_n + a_j$, y

$$T'_n = S_n + a_j, \quad T_n = S_j + a_j$$

$$\begin{gathered} S_n > S_j \\ S_n + a_j > S_n + a_j \\ T'_n > T_n \end{gathered}$$

Esto demuestra que el orden de C que minimiza T_n es con los tiempos a_i decrecientes.

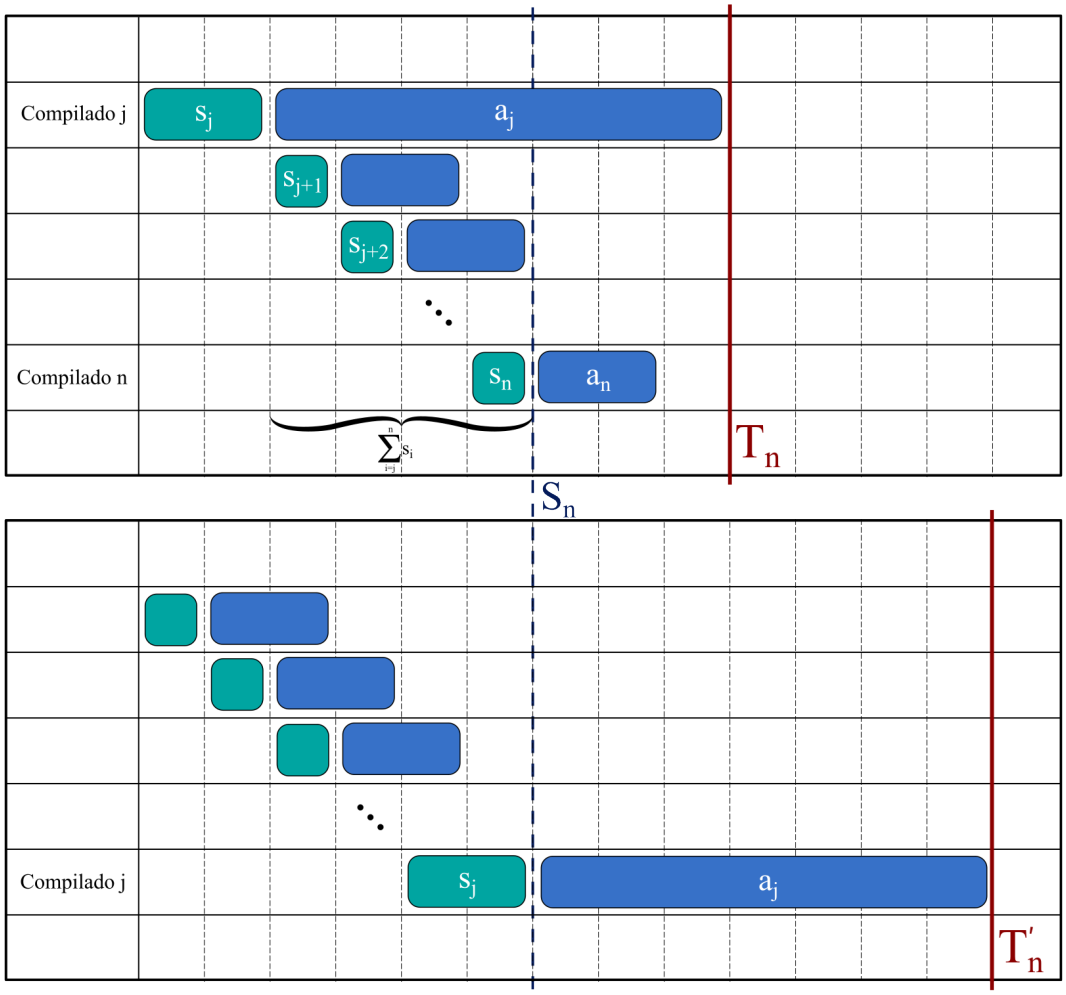


Figura 7: Ilustración de la demostración

Para ello propusimos el siguiente algoritmo:

En primer lugar, hemos definido una clase llamada `Compilado` para modelar el compilado de cada oponente, con los atributos `tiempo_scaloni` y `tiempo_ayudante`, que almacenan el tiempo que le lleva analizarlo a Scaloni y a algún ayudante, respectivamente.

```
class Compilado:
    def __init__(self, scaloni, ayudante):
        self.tiempo_scaloni = scaloni
        self.tiempo_ayudante = ayudante
```

De esta forma, y teniendo en cuenta los criterios previamente detallados, definimos la función `compilados_ordenados_de_forma_optima` que recibe como parámetro un arreglo con elementos de la clase `Compilado`. Esta ordena el arreglo en función del tiempo requerido por los asistentes para visualizar cada compilado, en orden descendente.

```
def compilados_ordenados_de_forma_optima(compilados):
    return sorted(compilados, key=lambda compilado: compilado.tiempo_ayudante,
    reverse=True)
```

- Los asistentes realizan el análisis de cada uno de los compilados asignados en paralelo. Por lo tanto, el tiempo que se invierte en la revisión de un compilado específico de máxima duración, puede ser aprovechado de manera tal que este sea visto mientras Scaloni se dedica a la revisión de otros compilados. De esta forma, nos aseguramos que se minimice el tiempo que suman los ayudantes en la resvisión total.
- También en esta solución tiene en consideración el análisis 2 descripto previamente, ya que en el mejor de los casos, la $\sum_{k=1}^{n} s_k + a_n$ termina siendo la mínima ya que, por la forma del ordenamiento del algorítmo, a_n es la duración del compilado más corto.

4.1. Cota de complejidad temporal

El algoritmo presentado tiene una complejidad temporal de $O(n \log n)$. Esto se debe a que seguimos los siguientes pasos: Ordenamos los compilados según el tiempo de análisis de los ayudantes. Para ello utilizamos sorted de la librería de python, que usa por detrás el algoritmo de Timsort, teniendo este una complejidad de $O(n \log n)$.

5. Mediciones

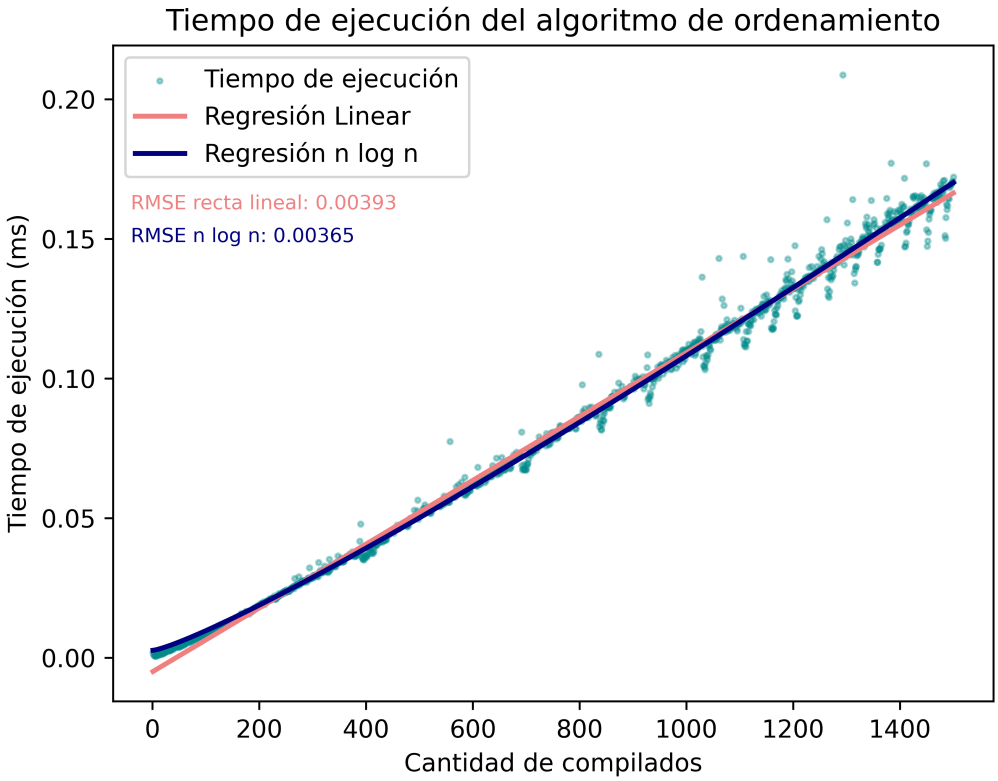
Para las mediciones, creamos listas de compilados de ejemplo de forma aleatoria. Decidimos que el tiempo de análisis de un compilado fuera un valor aleatorio entre 1 y 99'999, basándonos en los datos de ejemplo provistos por la cátedra. Medimos el tiempo de ejecución de nuestro algoritmo para ciertas cantidades de compilados.

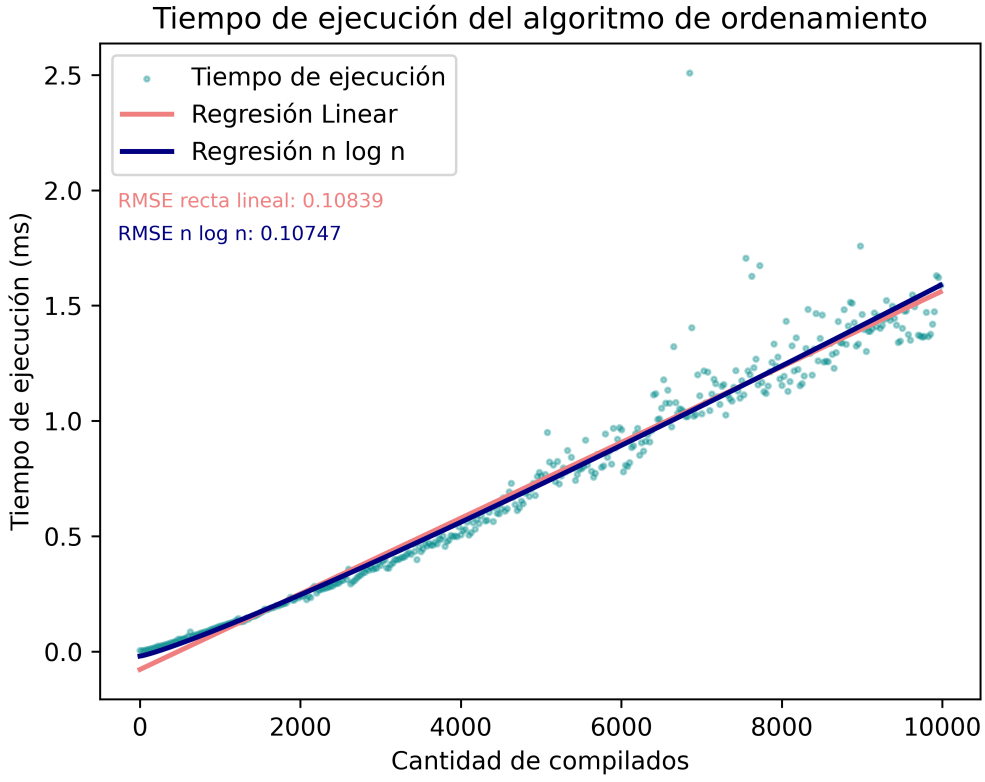
Calculamos una regresión lineal y una regresión lineal logaritmica que se ajuste a nuestros datos y graficamos sus curvas. Ahora, debemos comprobar cuál de las dos curvas se ajusta mejor a nuestros datos. Para ello, usamos la raíz del error cuadrático medio, que nos da una idea de cuán cerca están los valores predichos de los valores reales.

Al calcular el error cuadrático medio para ambas curvas, observamos que el error era prácticamente similar. En consecuencia, decidimos llevar a cabo un análisis más detallado, enfocándonos en diversos intervalos del gráfico con el objetivo de examinar el comportamiento de los datos en mayor profundidad. Esta investigación nos permitió concluir que, especialmente en el caso de muestras caracterizadas por valores reducidos, la curva lineal logarítmica se ajusta mejor a los datos observados.

Sin embargo, al considerar valores más elevados, se observó que las diferencias en los tiempos tendieron a adquirir una naturaleza lineal. De la misma manera, la función logarítmica se asemeja a una lineal para valores grandes, por lo que el error cuadrático medio tendía a ser el mismo para ambas funciones. Dado este análisis, decidimos incrementar el número de muestras para cantidades de compilados menores, y disminuir el número de muestras para cantidades de compilados mayores.

De esta manera, mediante el gráfico y la validación a través del error cuadrático medio, hemos comprobado que la curva lineal logarítmica se adapta de manera más precisa a nuestros datos.


Tiempo de ejecución del algoritmo de ordenamiento
Tiempo de ejecución
Regresión Linear
Regresión n log n
RMSE recta lineal: 0.00393
RMSE n log n: 0.00365
Tiempo de ejecución (ms)
0.00
0.05
0.10
0.15
0.20
0
200
400
600
800
1000
1200
1400
Cantidad de compilados


Tiempo de ejecución del algoritmo de ordenamiento
Tiempo de ejecución
Regresión Linear
Regresión n log n
RMSE recta lineal: 0.10839
RMSE n log n: 0.10747
Tiempo de ejecución (ms)
0.0
0.5
1.0
1.5
2.0
2.5
0
2000
4000
6000
8000
10000
Cantidad de compilados

De esta forma pudimos comprobar empíricamente que la complejidad tiende a $O(n \log n)$.

6. Conclusiones

Finalmente, consideramos que la solución óptima para abordar la problemática de Scaloni sería que éste analizara los compilados en función del tiempo requerido por los asistentes para visualizar cada uno, organizándolos en orden descendente. Esta estrategia permitiría resolver el problema con una complejidad algorítmica de orden $O(n \log n)$. Este enfoque garantiza la máxima eficiencia en la visualización de los compilados y permite que el tiempo invertido por Scaloni y sus asistentes se administre de manera óptima.