.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

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.

2. Solución óptima

Para realizar de forma óptima el trabajo propuesto, consideramos dos cuestiones:

- 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 esto en cuenta, el ordenamiento que empleamos ignora el tiempo que le tarda al DT ver cada compilado.
- 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.

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.

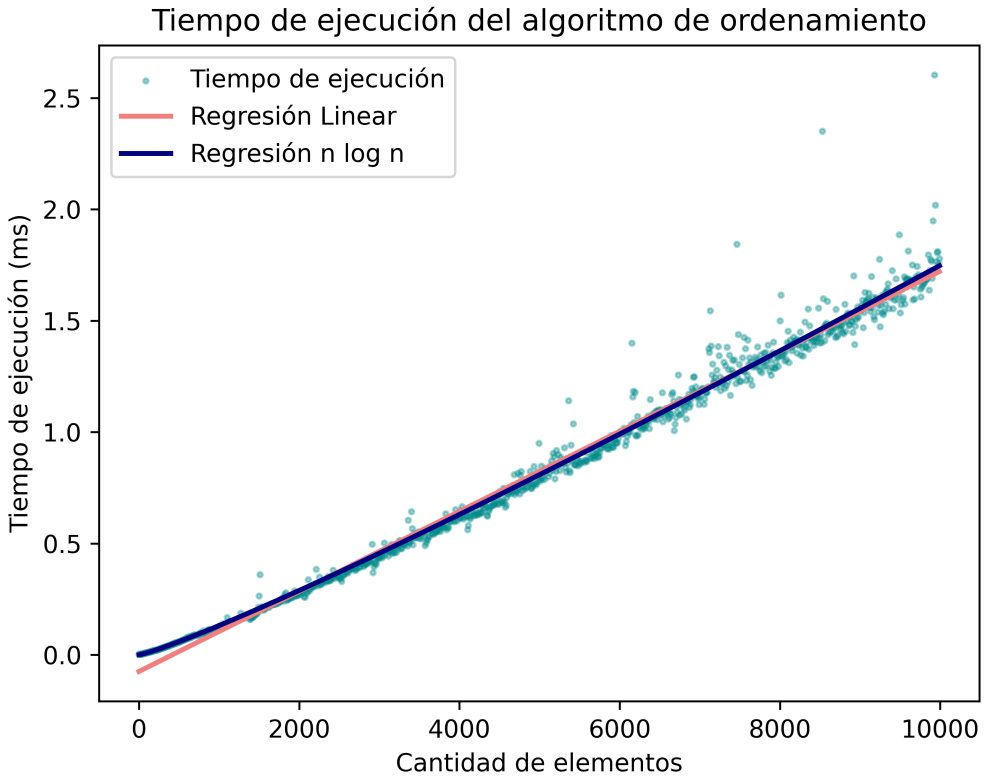
```
def compilados_ordenados_de_forma_optima(compilados):
    return sorted(compilados, key=lambda compilado: compilado.tiempo_ayudante,
    reverse=True)
```

2.1. 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 $O(n \log n)$.

3. 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 listas de 10 hasta 10'000 elementos y luego los graficamos.



Calculamos una regresión lineal y una regresión lineal logaritmica y graficamos sus curvas sobre los datos. De esta forma pudimos comprobar empíricamente que la complejidad tiende a $O(n \log n)$.

4. 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 videos y permite que el tiempo invertido por Scaloni y sus asistentes se administre de manera óptima.