Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

19 de septiembre de 2023

Juan Pablo	Mateo Daniel	Lucas Rafael
Carosi Warburg	Vroonland	Aldazabal
109386	109635	107705

Índice

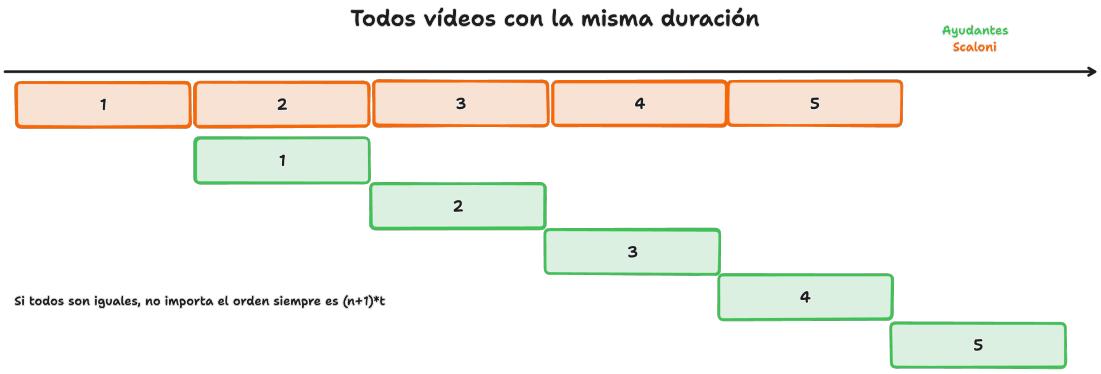
1. Introducción

En este informe haremos un estudio del problema planteado, como ordenar los vídeos a analizar por Scaloni y su equipo de ayudantes de tal forma que el tiempo total en verlo sea menor. En primer lugar, vamos a explicar el algoritmo al que llegamos y como llegamos a él, luego analizar porque este es efectivamente el óptimo, estudiar que sucede si variamos los distintos tiempos, y por último realizar un análisis de complejidad temporal del algoritmo planteado, con sus respectivas pruebas empíricas.

2. El camino hacia el Algoritmo

2.1. Todos Iguales

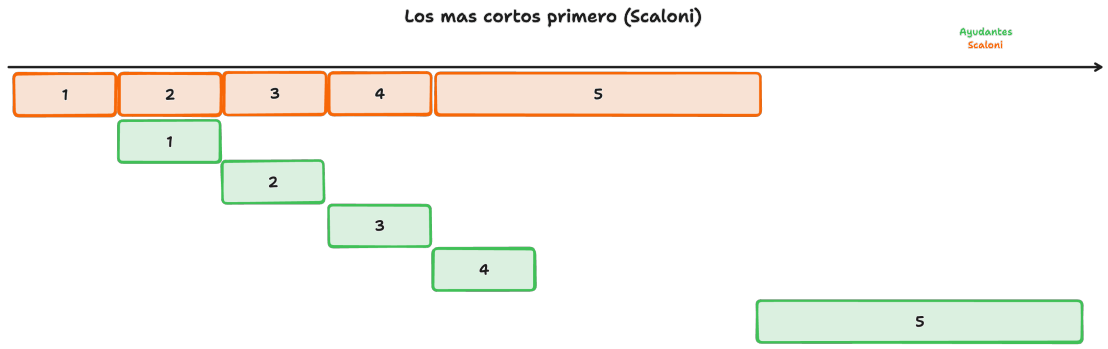
En primer lugar, planteamos el escenario más sencillo, todos los vídeos duran lo mismo independientemente de quien los vea.



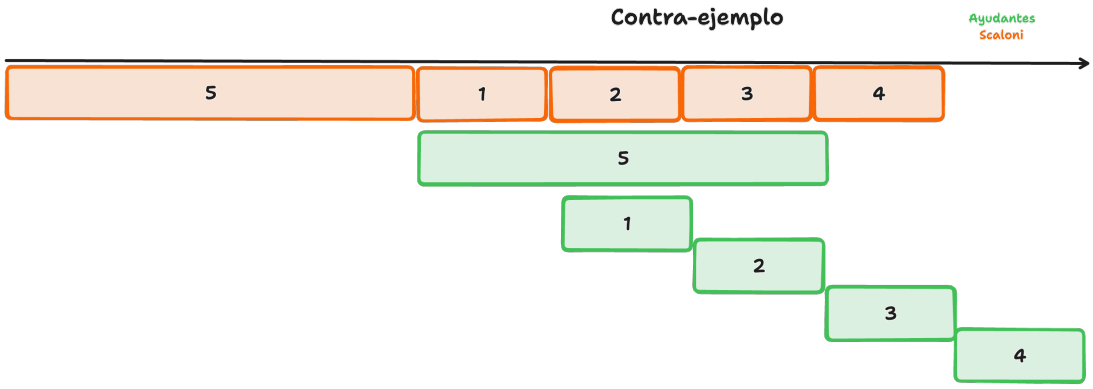
Queda claro al ver la figura que independientemente de cómo los ordenemos, siempre tardarán lo mismo, si tenemos n vídeos, donde cada uno tarda t, entonces la duración está determinada a siempre ser *(n+1)*t*.

2.2. Tiempo de Scaloni igual que el de los ayudantes

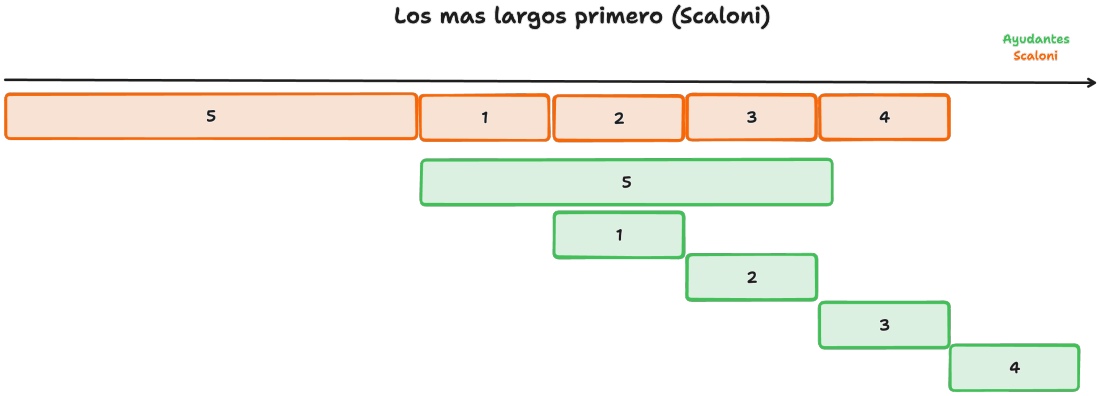
Luego que ver como se comportaba el problema donde todos los videos eran iguales, le incrementamos la complejidad, ahora haciendo variables los tiempos de los vídeos. Cabe destacar que por ahora vamos a considerar que cada video dura lo mismo para Scaloni que para los Ayudantes. Al plantear este nuevo escenario, en lo primero que pensamos fue ordenar por la duración de Scaloni ascendentemente, así este *libera* antes los videos para que los ayudantes puedan ir viendo los videos.



Sin embargo este no es el óptimo, ya que si vemos primero el video 5, tardaremos menos:

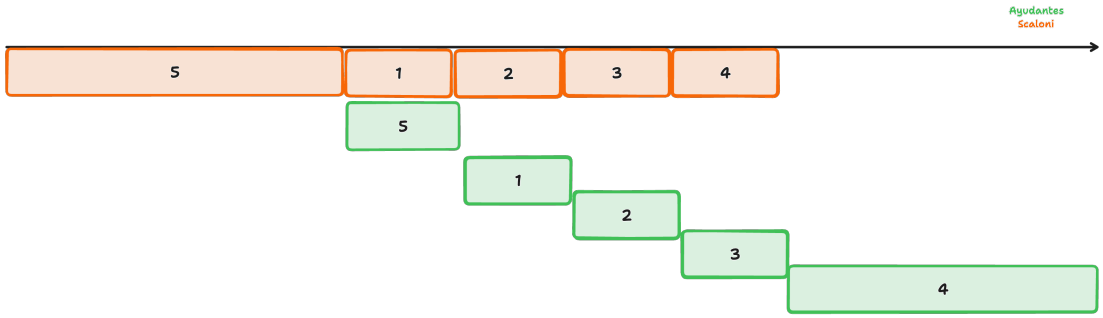


Aprendiendo del intento fallido, pudimos observar que ordenando ascendentemente por Scaloni, nos quedaba un bache donde este estaba mirando el video más largo y ningún ayudante estaba mirando nada, y no fue hasta que termino el ultimo video que el ayudante comenzó a ver el video más largo. Inspirándonos en eso decidimos ordenar por duración de Scaloni descendentemente. De esta manera, Scaloni primero mira los videos más largos, y mientras un ayudante mira el video largo, otros ayudantes pueden ir viendo los próximos videos más cortos, aprovechando de esta manera a los n ayudantes.

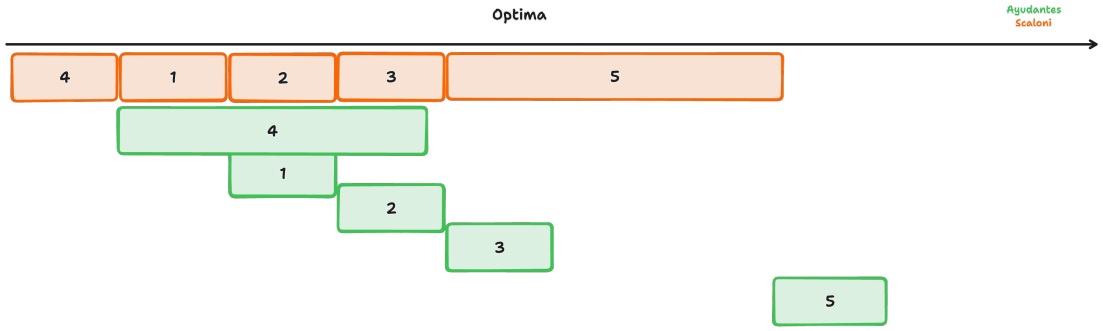


2.3. Tiempos Independientes

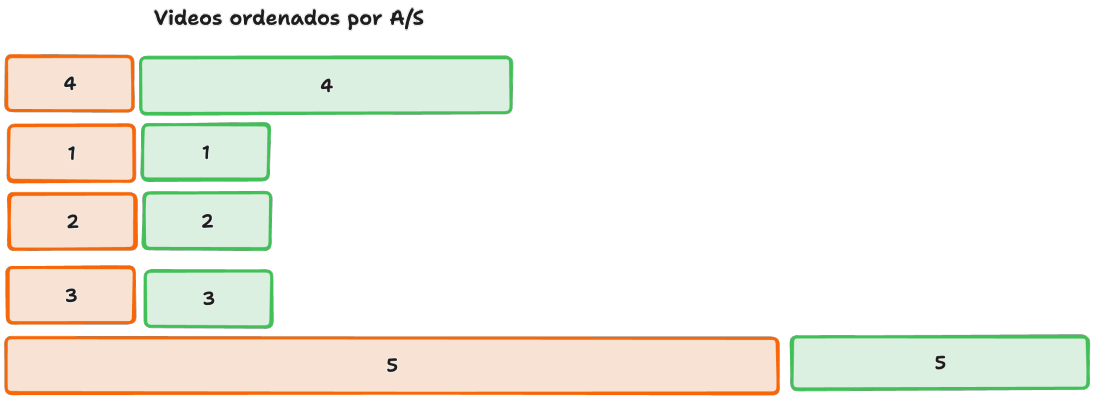
Ordenar descendentemente por Scaloni funciono perfectamente hasta ahora. ¿Pero qué pasa si ahora consideramos el problema completo, donde los tiempos de los ayudantes y de Scaloni son independientes?



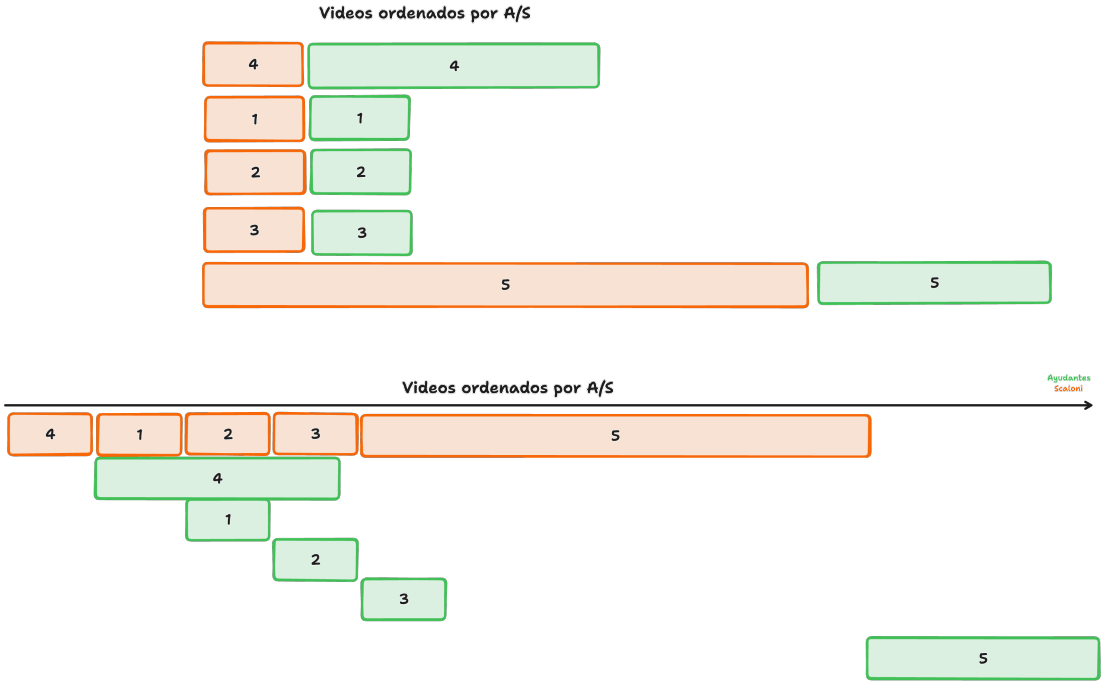
Si seguimos la lógica que teníamos hasta este punto, los videos nos quedarían como se ve en la figura. Sin embargo volvemos a ver un problema que creíamos haber solucionado, donde los ayudantes no realizan nada de trabajo mientras Scaloni mira su video más largo, y no es hasta el final que los ayudantes pueden ver el último video, el cual es el más largo de ver para ellos. La solución óptima al problema recién planteado seria:



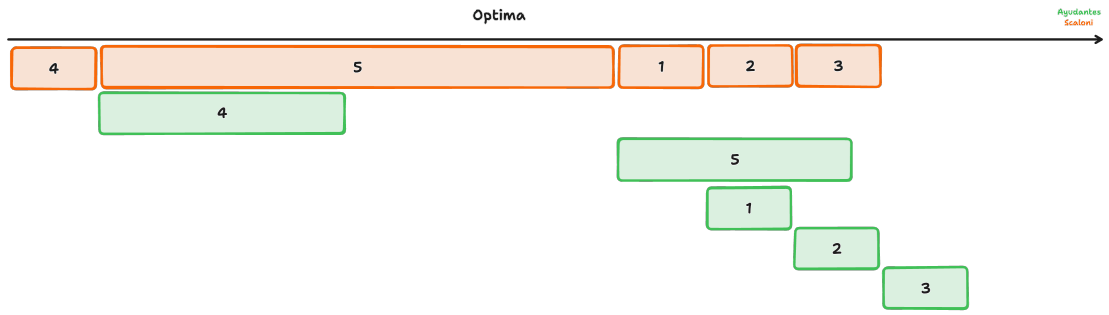
Con este ejemplo empezamos a ver que al tener tanto el tiempo de Scaloni como el de los ayudantes variable, deberíamos tomar ambas en consideración. Teniendo eso en mente se nos ocurrió la idea de hacer un coeficiente que sea $\frac{tiempo\ de\ ayudante}{tiempo\ de\ Scaloni}$, así podemos considerar ambas variables a la vez. Empezamos viendo si el ejemplo anterior efectivamente se resuelve, así que calculamos los coeficientes y ordenamos los videos descendentemente, llegando al siguiente orden:



Nuevamente vemos que el sistema pareciera funcionar ya que busca el equilibro de las tareas donde Scaloni rápidamente deje videos listos para los ayudantes y que a su vez los ayudantes tengan videos relativamente largos al principio con respecto al final, pero igualmente hay fallas. Cuando un video es demasiado largo tanto para Scaloni como para los ayudantes, este puede tener una relación baja (o sea fácil para A en comparación con S) e ir al final, pero sobresalir mucho en el tiempo total de Scaloni. Como imaginábamos pudimos encontrar un contraejemplo:



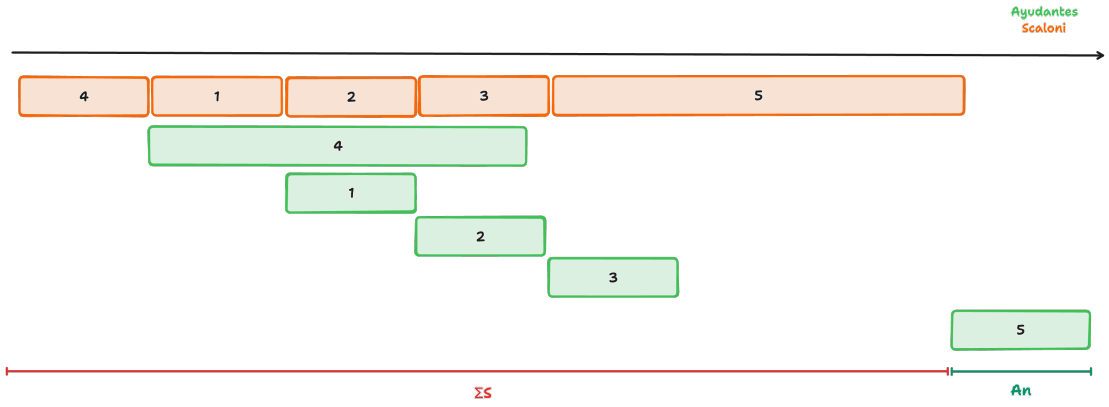
Una vez más vemos que luego de un largo bache donde solo Scaloni está viendo su ultimo video y no es hasta que termina que comienza uno de los más largos de los ayudantes, entonces podríamos optimizar esta solución:



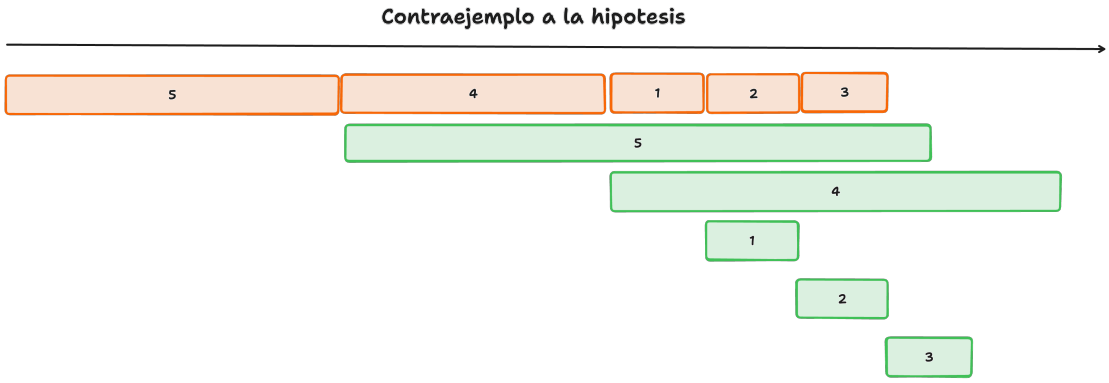
Llegado este punto, pudimos sacar ciertas hipótesis basándonos en los intentos erróneos.

- En primer lugar, concluimos que el tiempo total que tarda Scaloni siempre es el mismo independiente del orden, ya que siempre que termina con uno arranca con otro sin baches.
- La segunda hipótesis fue, que, al tener tantos videos como ayudantes, estos últimos son virtualmente ilimitados, por lo que cada vez que Scaloni termine un video siempre habrá un ayudante disponible para verlo.
- La última de estas hipótesis consistía en que los ayudantes siempre van a terminar después que Scaloni, consecuencia directa de mirar los videos solo después que Scaloni los haya visto.

Considerando estas hipótesis construimos una cuarta hipótesis: El tiempo total siempre es el tiempo que tarda en ver Scaloni todos los videos más el ultimo video visto por un ayudante:



Si esta hipótesis fuese correcta, entonces el tiempo óptimo de será ordenar a los ayudantes de forma descendente, de tal manera que $tiempo\ total = \sum s\_i + min\{a\_i\}$. Esta sería la óptima ya que como dijimos previamente, bajo las hipótesis que establecimos, el tiempo total es todo el tiempo que tarda Scaloni sumándole el del el ultimo ayudante. Teniendo en cuenta que el tiempo de Scaloni es independiente del ordenamiento, el tiempo mínimo posible es aquel que minimiza el tiempo que depende de los ayudantes.
Sin embargo, esta última hipótesis que planteamos no es correcta, demostrado con el contraejemplo:



Esta hipótesis resultó ser falsa con ejemplos donde los videos más largos de los ayudantes sean tanto más largos que los demás videos, que aunque comiencen antes, sigan luego de que estos últimos hayan terminado. De todas formas este algoritmo sigue siendo el óptimo, en caso donde $tiempo\ total = \sum s\_i + a\_n$ ya quedo mostrado (teniendo en cuenta que $a\_n = min\{a\_i\}$), pero para el caso donde un tiempo de un ayudante se sobrepase del último video, sigue siendo el óptimo. Esto es porque si tenemos un video el cual es mucho más largo para los ayudantes que el resto, queremos que este arranque lo antes posible, para así poder evitar que un video que no es el último termine después que Scaloni termine de ver todos los videos, y si llega a suceder, que sea el menor tiempo posible. Entonces en ambos casos ordenar por duración de los ayudantes descendentemente es lo óptimo para terminar lo antes posible todos los videos.

3. Algoritmo planteado

Nuestro algoritmo comienza primero por leer todos los videos de nuestro set de videos, y guardarlos en una lista para su posterior uso.

Luego, ordenamos esta lista de videos por orden decreciente de duración utilizando como referencia los videos de los ayudantes. Utilizamos la función de ordenamiento provista por el lenguaje Python, que utiliza el algoritmo `TimSort` el cual va a ser explicado posteriormente.

Por último, utilizamos la siguiente función para calcular el tiempo óptimo del set de videos. Para lograr esto, recorremos todos los videos sumando el tiempo de S_i a tiempo Scaloni y verificando dos posibles casos.

- Tomando el video_i, si $s\_i + a\_i + tiempo\ hasta\ video\_i$ supera la duración máxima total en ese momento, sobrescribimos la duración total con el nuevo valor. Caso contrario, mantenemos la duración total anterior.

```
def calcular_tiempo(videos):
    duracion_total = 0
    tiempo_scaloni = 0
    for video in videos:
        duracion_total = max(duracion_total, tiempo_scaloni + video['S_i'] + video['A_i'])
        tiempo_scaloni += video['S_i']
    return duracion_total
```

3.1. Complejidad temporal de nuestro algoritmo

Nuestro algoritmo comienza primero realizando una operación $\mathcal{O}(n)$, ya que realizamos n operaciones $\mathcal{O}(1)$ para guardar todos los videos del set. En la última parte de nuestro algoritmo también realizamos otra operación $\mathcal{O}(n)$ ya que volvemos a recorrer todos los videos. Sin embargo, bajo

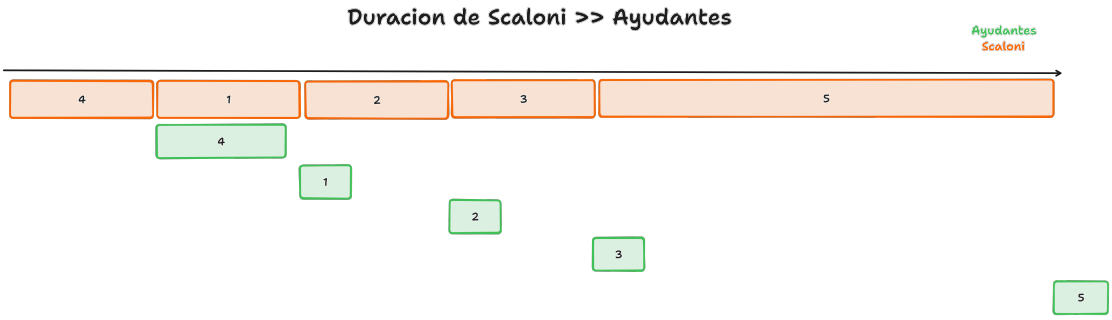
la notación Big O, esto no es muy relevante ya que la complejidad de nuestro algoritmo la va a determinar nuestro método de ordenamiento `TimSort`, dado que tiene un complejidad temporal mayor.

El método de ordenamiento `TimSort` consiste en una combinación de insertion sort y mergesort. Su mejor escenario en términos de complejidad temporal es $\mathcal{O}(n)$, pero esto solo lo logra cuando el arreglo de elementos ya viene ordenado en su totalidad. Mientras que en su peor caso termina siendo $\mathcal{O}(nlogn)$, el cual es bastante eficiente para un método de ordenamiento.
El factor clave de este ordenamiento que lo hace tan eficiente, es que si este algoritmo detecta partes ordenadas en el arreglo no las divide como pasa en mergesort sino que, ya las deja disponibles para mergear ahorrándose varios llamados recursivos innecesarios.
Otro atributo de este ordenamiento es que si la cantidad de elementos totales es menor a 64 se decanta por utilizar el método de insertion sort, el cual si bien tiene una complejidad temporal de $\mathcal{O}(n^2)$, para relativamente pocos elementos es muy eficiente.

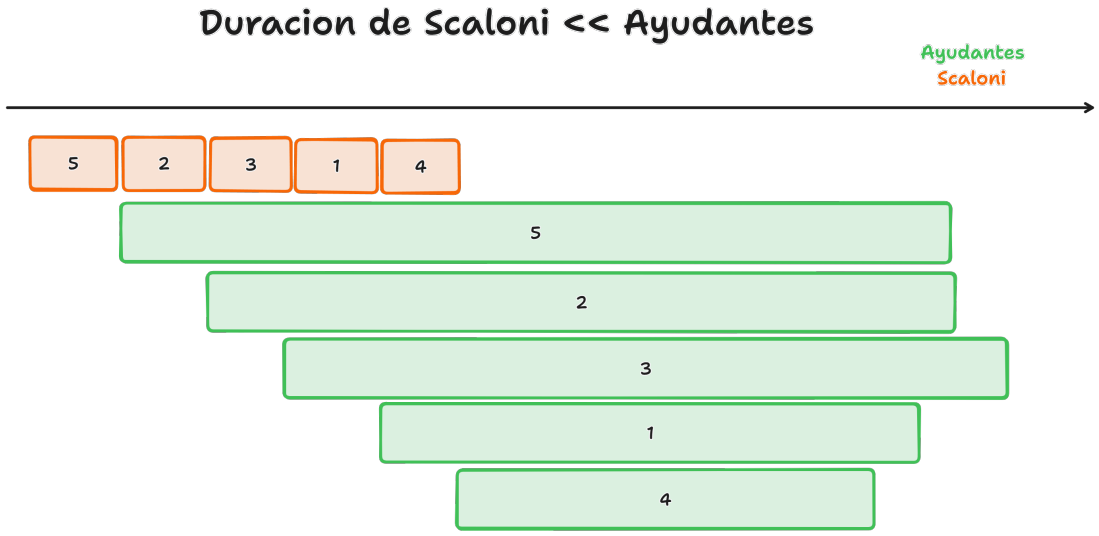
Para concluir, la complejidad temporal de nuestro algo rimo sería de $\mathcal{O}(nlogn + 2n)$, pero bajo la notación Big O, eliminamos los términos no determinantes que fueron nombrados anteriormente, quedándonos con una complejidad temporal de $\mathcal{O}(nlogn)$.

3.2. Variabilidad de los valores de Scaloni y sus ayudantes

En primer lugar, la variabilidad en los valores de duración de Scaloni no van afectar el tiempo total de ninguna manera. Esto se debe a que por más de que las duraciones de los videos de Scaloni sean muy elevados o difieran mucho video a video, este si o si va a tener que verlos todos y va a tardar lo mismo sin importar el orden en el que los vea.
Bajo este caso, notamos un patrón que se cumple en la gran mayoría de las veces, por no decir todas, el cual es que el tiempo óptimo se conforma por $tiempo\ total = \sum s\_i + min\{a\_i\}$, tal como sucede en el siguiente ejemplo.



Por otro lado, la variabilidad en los valores de duración de los ayudantes genera que el patrón enunciado anteriormente ya no sea válido en la gran mayoría de los casos, debido a que puede suceder que uno de los videos de los ayudantes tenga duración mayor a la sumatoria de los videos de Scaloni mas el video mas corto de los ayudantes. Además, notamos que en estos casos es cuando tenemos a la mayor cantidad de ayudantes trabajando en paralelo. Estas dos observaciones se pueden observar en la siguiente imágen.



Ninguna de estas variabilidades van a arruinar la optimalidad de nuestro algoritmo ya que cambios en los tiempos de Scaloni no brindan diferencias en nuestro algoritmo y, diferencias en el tiempo de los ayudantes tampoco porque con el ordenamiento esto no va a tener relevancia. Lo único que generaría es que el tiempo total del set de videos sea elevado, pero seguirá siendo el mejor caso posible.

4. Mediciones

Se realizaron múltiples mediciones al algoritmo desarrollado con dos finalidades principales, ver su optimalidad y performance.

Para verificar que obtenemos los resultados esperados, utilizamos los archivos provistos por la cátedra logrando exitosamente replicar el tiempo óptimo de los mismos. Para estos casos, hicimos uso únicamente del tiempo total calculado pero nuestro desarrollo también puede devolver los videos ordenados si fuese eso lo que se busca.

Una vez verificada la optimalidad, desarrollamos un sistema de tests aleatorios para verificar su performance con distintas cantidades de videos y variaciones de duración tanto para Scaloni como para los ayudantes. Para esto planteamos crear múltiples simulaciones con cantidades de vídeos creciendo exponencialmente con duraciones generadas aleatoriamente, tanto la exponencialidad como rango de duraciones es configurable mediante constantes en el código para los tests.

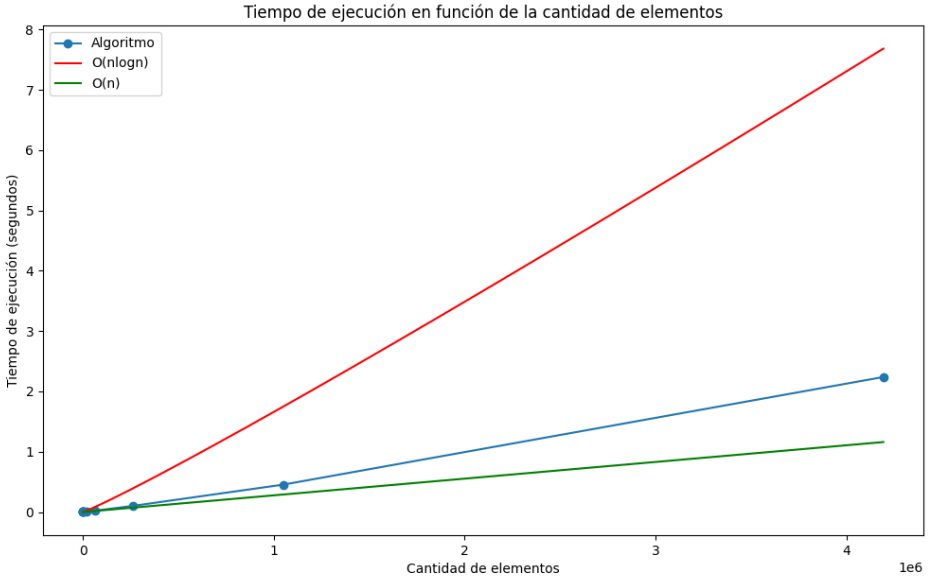
Los parámetros por defecto son hacer simulaciones con una cantidad de vídeos base 4 con exponente de 1 a 11 (4, 16, 64, ..., 4.194.304 vídeos). donde los videos tienen duraciones entre 0 y 10 * cantidad de videos (depende de la cantidad de videos para evitar que en cantidades grandes haya demasiados repetidos)

Una vez definidos los parámetros, por cada simulación se genera un archivo en el mismo formato que los provistos por la cátedra (S_i, A_i), los cuales quedan en una carpeta /datos por si luego se quiere consultar y/o replicar la simulación. Luego, estos archivos son interpretados y ordenados con nuestro algoritmo guardando tiempo total de ejecución.

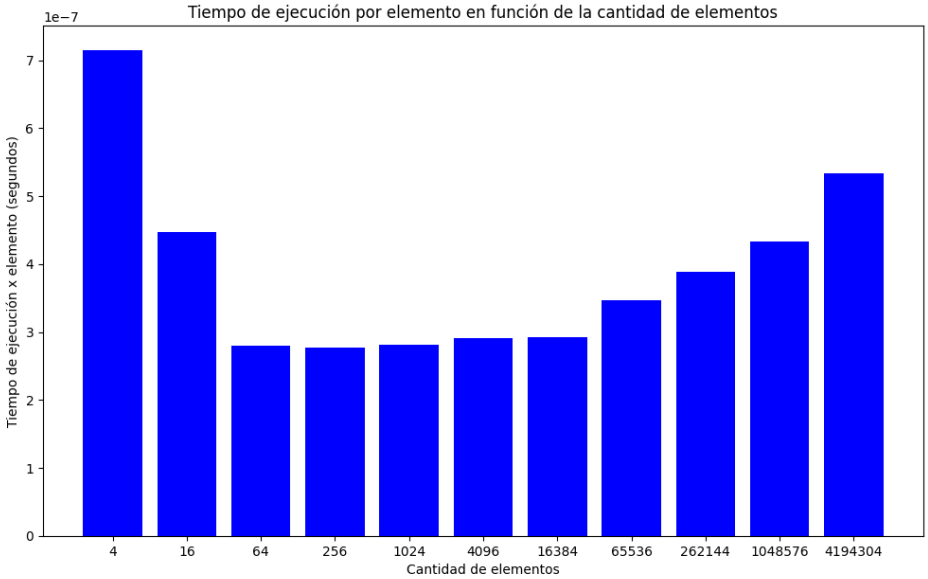
Una vez finalizadas las simulaciones, generamos dos gráficos con sus análisis:

El primero donde se gráfica una línea representando el tiempo total de ejecución por cantidad de elementos, comparándolo con una simulación de performance $\mathcal{O}(n)$ y $\mathcal{O}(nlogn)$. Estas dos simulaciones surgen de obtener la menor duración por video de los tests y hacer los respectivos cálculos para estimar cómo tendría que evolucionar por la cantidad de elementos. Los resultados de este gráfico son acordes a lo esperado, con una curvatura superior a $\mathcal{O}(n)$, pero igualmente por

debajo de la cota superior teórica del algoritmo ($\mathcal{O}(nlogn)$).



El segundo gráfico consiste en un gráfico de barras donde se muestra cómo evoluciona el tiempo de ejecución sobre la cantidad de videos totales. En este caso podemos ver cómo con pocos videos el tiempo por video es alto debido a que acá tienen mucha más relevancia factores $\mathcal{O}(1)$ como la declaración de variables o inicialización de memoria, pero a luego cuando la cantidad crece esto pierde relevancia y el tiempo por video baja para luego volver a aumentar acorde a su complejidad superior a $\mathcal{O}(n)$.



5. Conclusiones

En conclusión, el algoritmo para determinar el menor tiempo que tardaría el equipo en ver la totalidad de videos en menor tiempo de forma óptima consiste en simplemente ordenar de forma descendente los videos por su duración de los ayudantes (s_i), independientemente si el tiempo de duración total es $tiempo\ total = \sum s\_i + min\{a\_i\}$ o no, por lo analizado previamente, si hay un video que se excede del tiempo del último video, queremos que este arranque lo antes posible. En notación Big O nuestro algoritmo tiene complejidad de $\mathcal{O}(nlogn)$, pero como se detalló previamente con ciertas optimizaciones suele ser más óptimo que $(nlogn)$, y fue demostrado empíricamente, donde se mantuvo considerablemente por debajo de su cota superior.

Este es un algoritmo de tipo Greedy ya que mediante la solución de óptimos locales se puede generar una solución al óptimo global. Bajo nuestro caso, luego del ordenamiento, vamos recorriendo cada video haciendo solamente una verificación, la cual nos determina nuestro óptimo local. Repitiendo esto hasta el total de los videos nos lleva a encontrar la solución óptima y global del problema. Otro factor a tener en cuenta es que los algoritmos Greedy están directamente relacionados con ordenamientos o heaps.