.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

23 de abril de 2024

Santiago Varga
110561

Morena Sandroni
110205

Nahuel Cellini Rotunno
103320

1. Introducción

En este trabajo, presentaremos un algoritmo Greedy diseñado con el propósito de ayudar al Señor del Fuego a administrar sus batallas de forma tal que estas se resuelvan de la manera más efectiva posible. En este caso, tendremos en cuenta el tiempo de duración y la importancia de cada una de las batallas. Siendo t_i el tiempo que dura la batalla i y b_i su importancia, podemos definir F_i como el momento en el que termina cada batalla. Si la primera batalla es la j, entonces $F_j = t_j$. En cambio, si la batalla j se realiza justo después de la batalla i, entonces $Fj = Fi + tj$. La efectividad radica entonces en minimizar la suma ponderada de los tiempos de finalización de las batallas en su totalidad, teniendo en cuenta también sus respectivos pesos (la importancia de cada una):

$$\sum_{i=0}^{n} b_i F_i$$

El algoritmo presentado debería dar una solución óptima para el problema dados los valores de n y todos los t y b. A lo largo de este trabajo, no solo analizaremos el funcionamiento del algoritmo, sino que también evaluaremos su complejidad temporal. Además, exploramos su capacidad de adaptación a diferentes escenarios, considerando variaciones en los tiempos y los pesos de las batallas. En las siguientes secciones, vamos a explorar como llegamos al algoritmo óptimo, teniendo en cuenta que atravesamos diferentes etapas antes de dar con la solución ideal. Analizaremos cómo funcionaban estos algoritmos, porque llegamos a ellos y como nos dimos cuenta que no eran suficientemente óptimos, evaluaremos su desempeño en varios contextos y, al final, sacaremos una conclusión general.

2. Análisis del problema

Para poder minimizar la suma ponderada de los tiempos de finalización de las batallas, incluyendo en esta su respectiva importancia, analizamos la posible relación existente entre los pesos y los tiempos de cada batalla y cómo estos modifican el resultado esperado.
Si analizamos los términos de la sumatoria podemos ver que F_i es siempre creciente, ya que en definitiva es la suma de todos los tiempos empleados en las diferentes batallas, mientras que b_i va variando y puede ser un número grande o chico. Como estos dos términos se multiplican entre sí, si queremos minimizar la sumatoria los términos tienen que ser lo más chicos posibles para que las adiciones sean lo más chicas posibles. Siguiendo esta lógica podríamos pensar que poniendo las batallas menos importantes al principio para tener términos chicos podría ser una opción, pero esto significa que las batallas más importantes son las que quedan para el final y esto va a hacer que los últimos términos que estemos sumando sean muy grandes por estar multiplicando entre si dos números grandes. Entonces lo mejor es ordenar por batallas de mayor a menor importancia, lo que haría que los términos se equilibren, ya que a pesar de tener sumandos que pueden ser grandes al principio, los valores chicos de la importancia de las batallas van a hacer que los últimos valores de la sumatoria no sean tan grandes.
Esto se puede ver en el siguiente ejemplo:

Supongamos las siguientes batallas, siendo $B_i = (\ b_i, t_i)$:

- B1 = (10 , 3)
- B2 = (8 , 15)
- B3 = (4 , 2)

Ordenando por importancia de menor a mayor:
B3 - B2 - B1 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 4*2 + 8*17 + 10*20 = 8 + 136 + 200 = 344

Ordenando por importancia de mayor a menor:
B1 - B2 - B3 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 10*3 + 8*17 + 4*20 = 30 +136 + 80 = 246

Bajo esta línea de pensamiento también podríamos pensar que solo controlando el crecimiento de la finalización de la batalla se podría también minimizar la sumatoria. Pero si no tenemos en cuenta la importancia, por más que se controle ese número ordenando por tiempos de mayor a menor, no podemos garantizar que la batalla más importante no quede para el final y haga que el último sumando sea un número muy grande, ya que se estaría multiplicando el valor de importancia máxima con el valor de tiempo de finalización máximo (Esto se puede ver al comparar los últimos sumandos del ejemplo anterior, 200 es mucho mayor que 80). Ante esto también debemos preguntarnos: ¿qué pasa cuando hay batallas con la misma importancia? ¿es indistinto pelearlas en el orden que sea? ¿o debemos garantizar que se peleen unas antes que otras dependiendo del tiempo?

Para responder estas preguntas planteamos el siguiente ejemplo:

- B1 = (10 , 3)
- B2 = (8 , 15)
- B3 = (10 , 5)

Ordenando por importancia de mayor a menor las dos opciones que tenemos son las siguientes:

B1 - B3 - B2 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 10*3 + 10 * 8 + 8 * 23 = 30 + 80 + 184 = 294

B3 - B1 - B2 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 10*5 + 10 * 8 + 8 * 23 = 50 + 80 + 184 = 314

Se puede observar que, a igual importancia, si se pelean las batallas más cortas primero se minimiza la sumatoria, ya que de esta forma, tomando solo los empates, el primer sumando aporta un valor más chico que si no se toma en cuenta los empates o si se prioriza la batalla más larga primero. Entonces deberíamos buscar un algoritmo que no solo priorice las batallas más importantes primero, sino que además tenga en cuenta las batallas de menor duración. ¿Existe una manera de relacionar ambos parámetros de forma tal que se resuelva de manera óptima el problema?

3. Algoritmo propuesto

Finalmente, tomando como referencia el caso del problema de la mochila, podríamos establecer una relación entre la importancia de cada batalla y su duración de la forma b_i/t_i y luego ordenar los resultados de estas relaciones de mayor a menor. De este modo, se priorizan las batallas de mayor importancia y, a su vez, las de menor duración.

Volviendo al ejemplo anterior:

- B1 = (10 , 3)
- B2 = (8 , 15)
- B3 = (10 , 5)

Calculando los pesos segun la relacion b_i/t_i:

- B1 = 10/3 = 3,33
- B2 = 8/15 = 0,533
- B3 = 10/5 = 2

Ordenando de mayor a menor podemos garantizar que cuando se empata en la importancia de las batallas se prioricen las de menor tiempo:

B1 - B3 - B2 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 10*3 + 10 * 8 + 8 * 23 = 30 + 80 + 184 = 294

Para ver si este camino es el correcto vamos a probarlo en el primer ejemplo:

- B1 = (10 , 3)
- B2 = (8 , 15)
- B3 = (4 , 2)

La relación b_i/t_i es:

- B1 = 10/3 = 3,33
- B2 = 8/15 = 0,533
- B3 = 4/2 = 2

Ordenando de mayor a menor nos queda:

B1 - B3 - B2 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 10*3 + 4*5 + 8 * 20 = 30 + 20 + 160 = 210

En comparación a la solución planteada anteriormente, con este algoritmo da una suma ponderada menor. Pero también debemos preguntarnos qué pasa cuando dos batallas tienen exactamente la misma relación.

Para esto planteamos entonces un tercer ejemplo:

- B1 = (10 , 5)
- B2 = (8 , 15)
- B3 = (4 , 2)

La relación b_i/t_i es:

- B1 = 10/5 = 2
- B2 = 8/15 = 0,533
- B3 = 4/2 = 2

Siendo iguales sus relaciones, los dos ordenamientos posibles son:

B1 - B3 - B2 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 10*5 + 4*7 + 8 * 22 = 50 + 28 + 176 = 254

B3 - B1 - B2 $\rightarrow \sum_{i=0}^{n} b_i F_i$ = 4*2 + 10*7 + 8 * 22 = 8 + 70 + 176 = 254

Se puede observar que el orden entre B1 y B3 es irrelevante ya que la sumatoria da igual, por lo que podemos decir que cuando hay un empate entre relaciones no importa el orden. Entonces podriamos afirmar que el algoritmo planteado es el que ordena de mayor a menor según la relación b_i/t_i.

```python
def ordenar_por_proporcion(batallas):
    batallas.sort(key = lambda x: x[1]/x[0], reverse=True)
    return batallas

def calcular_tiempo_total(batallas):
    batallas = ordenar_por_proporcion(batallas)
    coef_de_impacto = 0 # Suma acumulada de los tiempos de finalizacion
    t_total = 0 # Tiempo total
    for t_act, b_act in batallas:
        coef_de_impacto += (t_total + t_act) * b_act
        t_total += t_act
    return coef_de_impacto
```

4. Optimalidad

El objetivo principal de esta demostración es verificar que nuestra solución propuesta para ordenar las batallas, basada en el criterio de la relación entre la importancia y la duración de cada batalla, es óptima. Para ello, emplearemos inversiones sobre la solución óptima previamente obtenida y demostraremos que estas inversiones no alteran la optimalidad de la solución, manteniendo la suma ponderada de los tiempos de finalización igual o menor que la solución original.
Al demostrar que podemos realizar inversiones sobre la solución óptima sin afectar su eficiencia, validaremos la validez y efectividad de nuestro algoritmo propuesto para optimizar el orden de las batallas.

- Caso 1: B_i es más importante que B_{i+1}, siendo $B = (b, t)$.

  Supongamos que tenemos las siguientes batallas con sus respectivas importancias y tiempos de duración:

  - B_i = (10,3)
  - B_{i+1} = (8,15)

  A partir de la secuencia original [B_i,B_{i+1}] obtenemos que la suma ponderada es de ($10*F_i$ + $8*F_{i+1}$), y luego al intercambiar a [B_{i+1},B_i] obtendremos que la suma ponderada es de ($8*F_{i+1}$ + $10*F_i$).

  Obteniendo finalmente que si F_i es mayor que F_{i+1}, la suma ponderada luego de la inversión disminuye, pero si F_{i+1} es mayor que F_i, entonces esta podrá mantenerse o disminuir.

- Caso 2: B_i y B_{i+1} tienen la misma importancia, pero B_i tiene una duración menor que B_{i+1}, siendo $B = (b, t)$.

  Supongamos que tenemos las siguientes batallas con sus respectivas importancias y tiempos de duración:

  - B_i = (8,5)
  - B_{i+1} = (8,15)

  A partir de la secuencia original [B_i,B_{i+1}] obtenemos que la suma ponderada es de ($8*F_i$ + $8*F_{i+1}$), y luego al intercambiar a [B_{i+1},B_i] obtendremos que la suma ponderada es de ($8*F_{i+1}$ + $8*F_i$).

  Como t_i ¡t_{i+1}, la suma ponderada después de la inversión será menor que la original.

- Caso 3: B_i y B_{i+1} tienen la misma importancia y duración, siendo $B = (b, t)$.

  Supongamos que tenemos las siguientes batallas con sus respectivas importancias y tiempos de duración:

  - B_i = (8,10)
  - B_{i+1} = (8,10)

  A partir de la secuencia original [B_i,B_{i+1}] obtenemos que la suma ponderada es de ($8*F_i$ + $8*F_{i+1}$), y luego al intercambiar a [B_{i+1},B_i] obtendremos que la suma ponderada es de ($8*F_{i+1}$ + $8*F_i$).

  Entonces, dado que las duraciones y las importancias son iguales, la suma ponderada se mantendrá igual después de la inversión.

Finalmente, hemos demostrado que en cada caso la inversión no aumenta la suma ponderada, lo que confirma que se pueden realizar inversiones sobre la solución óptima sin alterar su optimalidad.

5. Variabilidad de los tiempos del algoritmo dados diferentes valores de t_i y b_i

Si hablamos de los valores particulares que podrían tomar t_i y b_i, estos no afectan a los tiempos del algoritmo, ya que las operaciones que se hacen son siempre las mismas. Es decir, si tenemos una lista 1 cuyos valores de t_i y b_i varían entre 1 y 100, y una lista 2 cuyos valores de t_i y b_i varían entre 100 y 10000 el tiempo de ejecución del algoritmo para esas dos listas va a ser el mismo, y si hay algún tipo de variación no depende del algoritmo en sí, sino de factores externos que en principio están fuera de nuestro control. Si hablamos de la cantidad de elementos en la lista de batallas, el algoritmo va a tardar más en ejecutar a medida que la cantidad de elementos aumenta. Esto se debe a la complejidad del algoritmo de ordenamiento utilizado.

Para probar esto, hicimos varias pruebas de ejecucion con sets de datos de hasta 10.000 elementos:

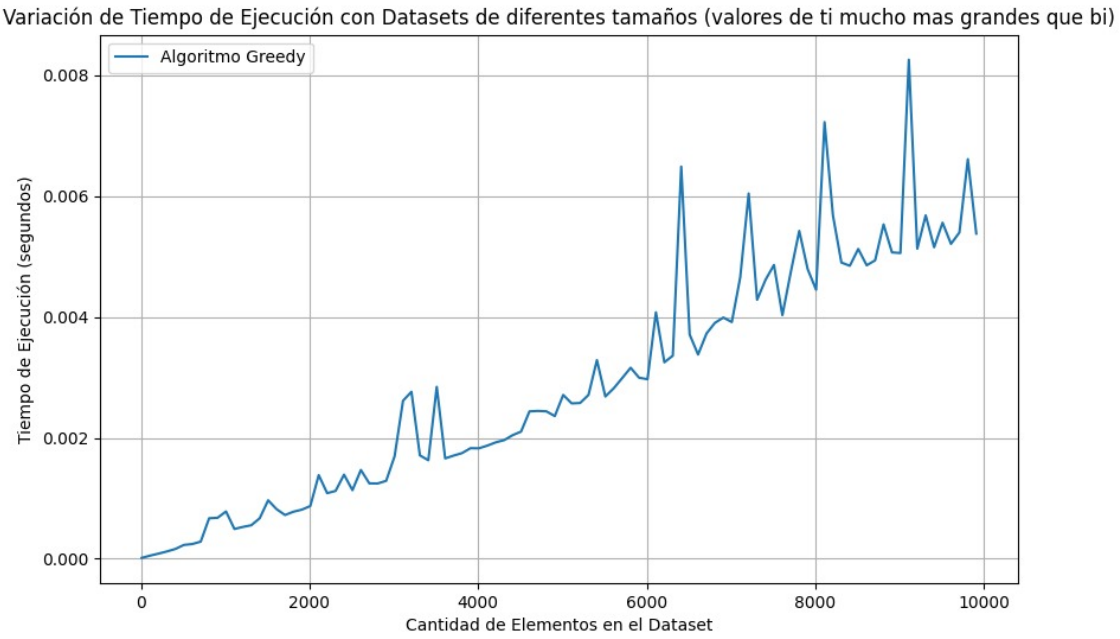

Grafico 1. Variación de Tiempo de Ejecución con Datasets de diferentes tamaños (valores de t_i mucho mas grandes que b_i)

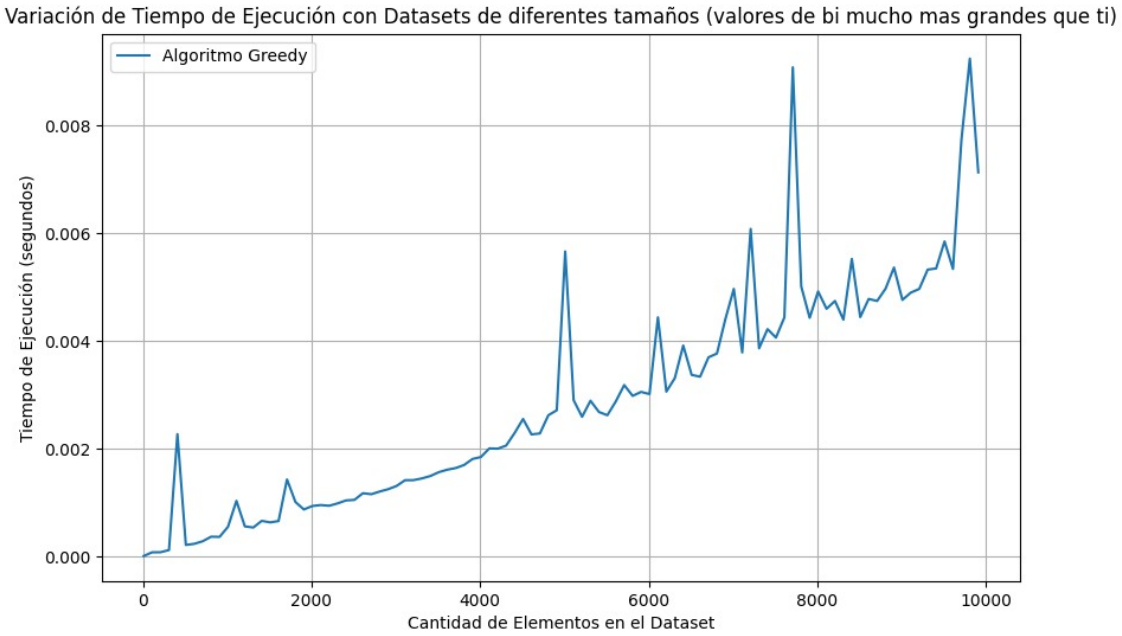

Grafico 2. Variación de Tiempo de Ejecución con Datasets de diferentes tamaños (valores de b_i mucho mas grandes que t_i)

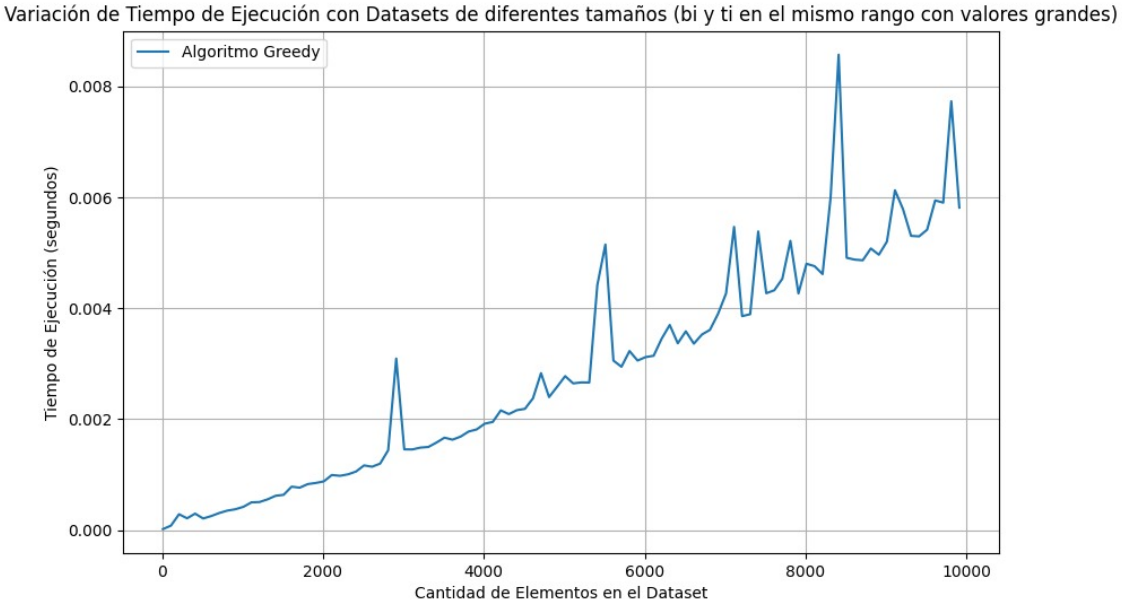

Grafico 3. Variación de Tiempo de Ejecución con Datasets de diferentes tamaños (b_i y t_i en el mismo rango con valores grandes)

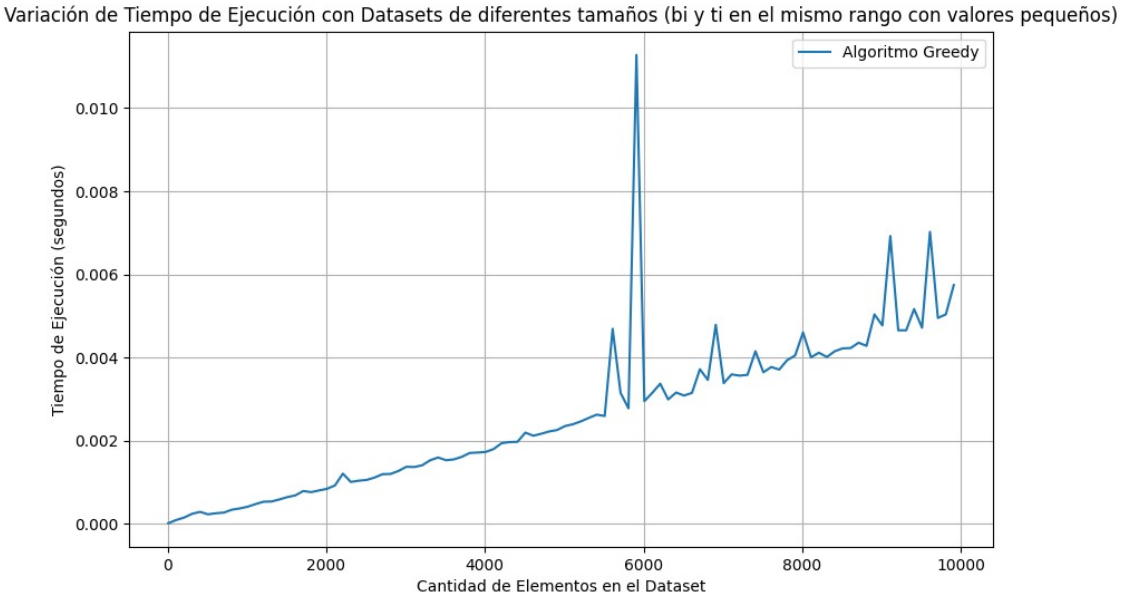

Grafico 4. Variación de Tiempo de Ejecución con Datasets de diferentes tamaños (b_i y t_i en el mismo rango con valores pequeños)

Como se puede observar en los diferentes graficos, el tiempo de ejecucion no depende de los valores de bi y ti ya que en todos los casos los graficos son practicamente iguales.

6. Variabilidad de la optimalidad del algoritmo dados diferentes valores de t_i y b_i

La variabilidad de t_i y b_i no afecta directamente a la optimalidad del algoritmo Greedy. Al estar utilizando una relación b_i / t_i, si los valores de t_i y b_i son muy parecidos van a tener una relación cercana a 1, mientras que si son muy diferentes van a estar más cerca de cero si t_i es mayor a b_i, y van a tender crecer si b_i es mayor a t_i.
Para ilustrar veamos el siguiente ejemplo:

- $B_1 = (1000, 2) \rightarrow b_i$ / $t_i = 0{,}002$
- $B_2 = (99, 100) \rightarrow b_i$ / $t_i = 1{,}01$
- $B_3 = (4, 900) \rightarrow b_i$ / $t_i = 225$

Las diferencias entre valores pueden seguir creciendo, pero siempre vamos a tener la misma tendencia, y el algoritmo siempre va a "priorizar" batallas de mucha importancia y poca duración

7. Complejidad Temporal

La complejidad del algoritmo es de $\mathcal{O}(n \log n + n)$, ya que es la complejidad del algoritmo de ordenamiento Timsort, que es el que utiliza la función .sort() de python, y la complejidad de recorrer la lista de elementos, haciendo operaciones $\mathcal{O}(1)$ en el medio. Como se cumple qué $O(nLog(n)) > O(n)$, finalmente decimos que la complejidad del algoritmo es $\mathcal{O}(n \log n)$.
En el gráfico se puede observar que el algoritmo tiene una tendencia a $\mathcal{O}(n \log n)$:

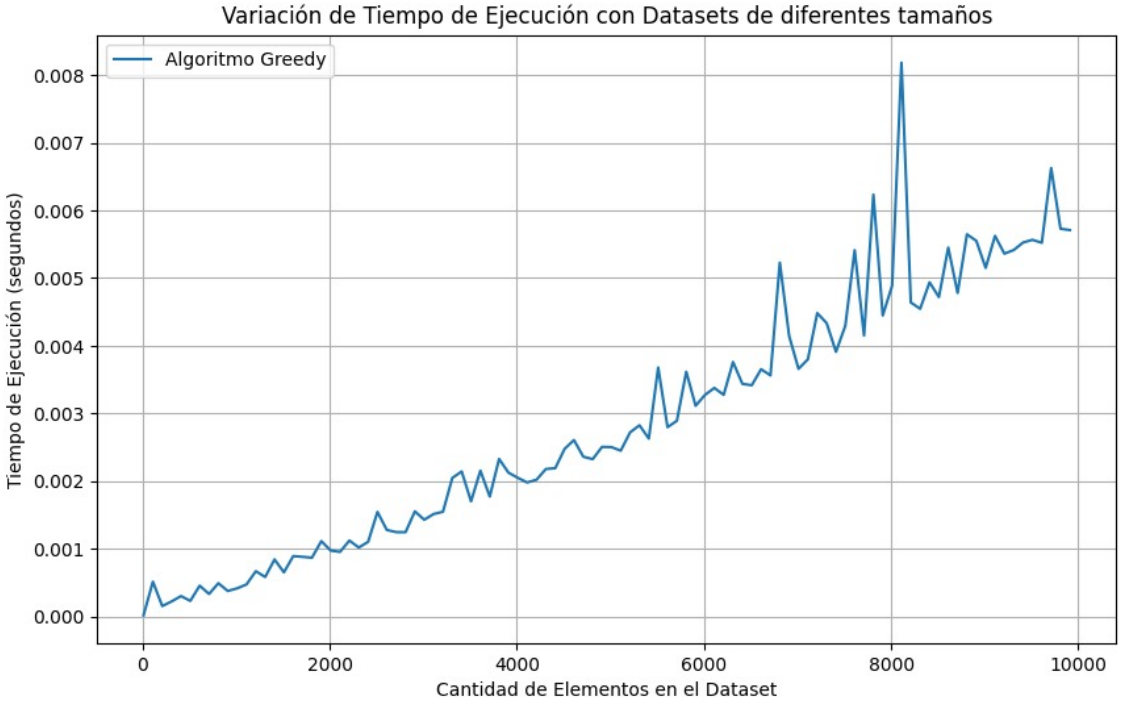

Grafico 5. Variación del tiempo de ejecución con diferentes tamaños de sets de datos.

8. Conclusión

Como conclusión, en este trabajo pudimos ver cómo los algoritmos Greedy pueden acercarse a soluciones óptimas en problemas de optimización de tiempo. A través de diferentes enfoques, buscamos ayudar al Señor del Fuego a administrar sus batallas de forma tal que estas se resuelvan de la manera más efectiva posible. Aunque creemos que el algoritmo basado en la relación b_i/t_i es óptimo, reconocemos que podría existir un algoritmo no Greedy que ofrezca mayor precisión a cambio de más tiempo de ejecución.