.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

Los Algoritmos Greedy son juegos de niños

19 de septiembre de 2024

Alejo Fábregas	Camilo Fábregas	Juan Cruz Hernández
106160	103740	105711

Los Algoritmos Greedy son juegos de niños

1. Análisis del problema

El problema que estamos buscando resolver consiste en hallar un algoritmo que, dado un set de monedas de diferentes valores, indique qué monedas Sophia debe elegir en cada turno (el suyo y el de Mateo) para asegurarse de siempre ganar el juego.

Durante nuestro análisis, nos dimos cuenta que lo único que necesita hacer Sophia en cada turno para asegurarse la victoria es elegir la moneda más grande en sus turnos, y elegir la moneda más pequeña en los turnos de Mateo. Dado que Sophia siempre comienza el juego, esto algorítmicamente significa elegir la moneda más grande en los turnos pares (si el primero es el turno 0) y la más pequeña en los turnos impares.

Dado que el juego consiste en elegir monedas de los extremos hasta que no queden más, el problema se puede pensar como un arreglo que se recorre "de afuera hacia adentro" una sola vez. Además, sabiendo que en cada turno se hace siempre lo mismo (elegir una moneda), la regla simple que elegimos para nuestro algoritmo Greedy consiste en elegir la más grande si el turno es par, y la más pequeña si el turno es impar.

1.1. Seguimiento del juego

Por ejemplo, para el arreglo de monedas [10, 5, 3, 8, 1, 6]:

- Turno 0 (Sophia): entre [10, 6] elige 10. Nuevo arreglo [5, 3, 8, 1, 6]
- Turno 1 (Mateo): entre [5, 6] elige 5. Nuevo arreglo [3, 8, 1, 6]
- Turno 2 (Sophia): entre [3, 6] elige 6. Nuevo arreglo [3, 8, 1]
- Turno 3 (Mateo): entre [3, 1] elige 1. Nuevo arreglo [3, 8]
- Turno 4 (Sophia): entre [3, 8] elige 8. Nuevo arreglo [3]
- Turno 5 (Mateo): elige 3.

Los resultados finales entonces, son 24 para Sophia y 9 para Mateo.

2. Algoritmo Greedy

2.1. Solución con Algoritmo Greedy

Habiendo hecho este análisis al problema planteado, la solución que pensamos es la siguiente:

```python
def ganar_siempre(monedas):
    cant_turnos = len(monedas)

    i_izq = 0
    j_der = cant_turnos - 1
    turno_actual = 0
    puntaje_sophia = 0
    puntaje_mateo = 0

    while turno_actual < cant_turnos:
        if turno_actual % 2 == 0:
            # Turno Sophia -> agarramos la moneda mayor
            if monedas[i_izq] > monedas[j_der]:
                puntaje_sophia += monedas[i_izq]
                i_izq += 1
            else:
                puntaje_sophia += monedas[j_der]
                j_der -= 1
        else:
            # Turno Mateo -> agarramos la moneda menor
            if monedas[i_izq] < monedas[j_der]:
                puntaje_mateo += monedas[i_izq]
                i_izq += 1
            else:
                puntaje_mateo += monedas[j_der]
                j_der -= 1

        turno_actual += 1

    return puntaje_sophia, puntaje_mateo
```

2.2. Complejidad

El algoritmo propuesto tiene una complejidad $\mathcal{O}(n)$. Esto no es posible de demostrar mediante el Teorema Maestro, ya que la solución no es recursiva. De todos modos, dado que el juego consiste en elegir una moneda por turno hasta que no haya más, se puede ver que en todos los casos se debe recorrer una vez el arreglo de monedas. No es posible cortar el juego antes porque en ese caso quedarían monedas sin elegir. Esto significa que si tengo un arreglo de n monedas, habrá n turnos en el que en cada uno se elige una sola moneda. Esto es equivalente a recorrer un arreglo una sola vez haciendo operaciones $\mathcal{O}(1)$ en cada elemento, lo que tiene un costo de $\mathcal{O}(n)$.

Más adelante realizaremos un análisis empírico con mediciones de tiempo para verificar que el algoritmo efectivamente tiene esta complejidad teórica.

2.3. Por qué es Greedy

El algoritmo propuesto es de tipo Greedy porque para cada iteración (en este caso, cada turno) estamos eligiendo la mejor opción posible para ese momento, sin importar lo que pasó en el turno anterior ni tampoco lo que va a pasar en el turno siguiente. El único factor que tenemos en cuenta en nuestro estado actual para tomar la mejor decisión es si el turno actual corresponde a Sophia, o a Mateo.

Nuestra regla sencilla nos va a permitir llegar siempre al óptimo local: Sophia eligirá la moneda más grande de las dos que puede elegir, y Mateo eligirá la más pequeña. Ninguno "mira a futuro", es decir ninguno elige pensando en maximizar (o minimizar) su puntaje (elegir una moneda más pequeña en un turno para elegir una considerablemente más grande en el siguiente, o viceversa).

Nuestro estado actual no es modificado en ningún momento, la moneda que se elige permanece hasta el final del juego y no se modifica turno a turno. Sólo importa el estado actual del problema, de modo que aplicando iterativamente nuestra regla sencilla para cada momento del problema, vamos a obtener el óptimo local para cada caso y terminar llegando a obtener el óptimo global, que es que Sophia obtenga la victoria. Al aplicar la regla greedy en cada turno, le damos la ventaja a Sophia en todo momento, lo que le asegura ganar al final de la partida.

2.4. Particularidades

El algoritmo propuesto garantiza que Sophia siempre gana la partida, pero no garantiza que obtenga la máxima puntuación posible. Para esto, habría que tener una noción de como agarrar una moneda de la izquierda o de la derecha del arreglo puede afectar a los siguientes turnos y sus resultados, por lo que dejaría de ser greedy.

Notamos esta particularidad con los casos de empate. Si hay monedas del mismo valor a ambos lados del arreglo, decidir si tomamos la de la izquierda o la derecha afecta al resultado final, ya que cambia la moneda que es nueva opción en el siguiente turno.

Por eso, si uno hace las comparaciones con $\geq$ (mayor o igual), los resultados varían con respecto a usar $>$ (mayor), y sus contrapartes con el menor. En nuestro caso, utilizando mayor estricto obtenemos los mismos resultados esperados de los ejemplos de la cátedra, pero utilizando mayor o igual, obtenemos números distintos (en general mayor puntaje para Sophia).

3. Demostración de optimalidad

Vamos a demostrar por inducción que el algoritmo greedy planteado es óptimo, es decir, que Sophia siempre obtiene una mayor puntuación que Mateo al final del juego.

Antes de la demostración, incluímos un breve análisis que nos permitió llegar a la conclusión final.

Sea n la cantidad de monedas iniciales:

- Si $n = 1$, gana Sophia ya que siempre empieza jugando.
- Si $n = 2$, también gana Sophia ya que de las dos monedas (la del principio y la del final), elige la de mayor valor, obligando a que en el turno de Mateo se elija la de menor valor.
- Si $n = 3$, gana Sophia. Se repite la situación de $n = 2$. En su turno, Mateo va a poder elegir la moneda más chica de las 2 opciones que tuvo Sophia o, si la nueva opción es peor, va a elegir esa. Por lo tanto, Sophia ya ganó en el segundo turno, por lo que la tercer moneda solo agrega valor pero no modifica el participante que gana el juego.

En términos más generales, solo hay dos casos posibles:

- **Si la cantidad k es impar**, el jugador que agarra la moneda $k + 1$ en el último turno es Mateo, ya que siempre empieza Sophia. En este caso, Sophia eligió la moneda de mayor valor $k//2$ veces, y Mateo obtuvo la moneda de menor valor $k//2 - 1$ veces. La moneda $k + 1$ que elija Mateo va a significar que eligió $k//2$ veces, las mismas que Sophia. Pero como Sophia siempre eligió la moneda de mayor valor, y ambos eligieron la misma cantidad de veces, Sophia tiene la victoria asegurada.
- **Si la cantidad k es par**, el jugador que agarra la moneda $k+1$ en el último turno es Sophia, ya que siempre empieza Sophia. En este caso, Sophia eligió la moneda de mayor valor $k//2$ veces, y Mateo obtuvo la moneda de menor valor $k//2$ veces. De cualquier manera, como asumimos que para k monedas siempre gana Sophia, como Sophia tiene que volver a elegir en la moneda $k + 1$ sabemos que va a ganar, ya que simplemente va a sumar todavía más puntaje a la victoria que ya tenía asegurada para la moneda k.

En resumen, Mateo solo va a poder elegir la opción que Sophia descartó o una incluso peor, pero nunca algo mejor, por lo que es imposible que en algún paso futuro, Mateo supere a Sophia (esto sumado a que Sophia siempre juega el primer turno, por lo que Matheo no podría tampoco superarla al jugar el último turno).

3.1. Demostración por Inducción

Sea x el puntaje de Sophia, y el puntaje de Mateo y m_i la moneda seleccionada en el turno i. Vamos a demostrar que, en todo momento, x es mayor que y (en otras palabras, que $x - y > 0$).

Sea i el turno actual, si consideramos como primer turno $i = 1$, entonces todos los turnos impares pertenecen a Sophia y todos los turnos pares pertenecen a Mateo. Por lo tanto:

$$x = \sum_{i \text{ impar}} m_i \quad \text{e} \quad y = \sum_{i \text{ par}} m_i$$

Entonces vamos a demostrar que, no importa el valor de n, la sumatoria de valor para i impar siempre va a ser mayor que la sumatoria de valor para i par:

$$x - y = \sum_{i \text{ impar}}^{n} m_i - \sum_{i \text{ par}}^{n} m_i > 0$$

Caso Base: para $n = 2$, Sophia gana ya que, de las dos opciones, va a seleccionar la mejor en su primer turno, dejando sin alternativa a Matheo. Por lo tanto:

$$x - y = \sum_{i \text{ impar}}^{n} m_i - \sum_{i \text{ par}}^{n} m_i = m_1 - m_2 > 0$$

Paso Inductivo:

$$x - y = \sum_{i \text{ impar}}^{h} m_i - \sum_{i \text{ par}}^{h} m_i > 0$$

$$\sum_{i \text{ impar}}^{h+1} m_i - \sum_{i \text{ par}}^{h+1} m_i > 0$$

$$(\sum_{i \text{ impar}}^{h} m_i + m_{h+1 \text{ impar}}) - (\sum_{i \text{ par}}^{h} m_i + m_{h+1 \text{ par}}) > 0$$

Como se partió de un caso base con n par, en el paso h +1 no existe $m_{h+1 \text{ par}}$. Aún si existiera, sería menor a $m_{h+1 \text{ impar}}$ ya que fue el paso siguiente a la elección de Sophia

$$(\sum_{i \text{ impar}}^{h} m_i + m_{h+1 \text{ impar}}) - (\sum_{i \text{ par}}^{h} m_i + m_{h+1 \text{ par}}) > 0$$

$$(\sum_{i \text{ impar}}^{h} m_i - \sum_{i \text{ par}}^{h} m_i) + (m_{h+1 \text{ impar}} - m_{h+1 \text{ par}}) > 0$$

El primer termino es positivo gracias al caso base y el segundo término también, por lo queda demostrado que Sophia siempre gana.

3.2. Variabilidad

La variabilidad de los valores de las monedas no afecta a la optimalidad del algoritmo. Se puede ver en la demostración que en ningún momento se considera los valores de las monedas. Esto es porque nuestro algoritmo tampoco considera los valores puntuales de las monedas, sólo le interesa ver cuál es mayor, si la moneda que está a la izquierda o la que está a la derecha. Al ser greedy, lo único que importa es cuál moneda es más beneficiosa para Sophia, sin importar si la diferencia entre las monedas es mínima o si es muy grande. Al hacer esto en cada turno, Sophia siempre tendrá la ventaja sin importar los valores, ni si gana por mucho o por poco. Es por eso que la variabilidad de los valores no importa, Sophia siempre eligirá el óptimo local. La única excepción es que las monedas son todas iguales, en ese caso la partida terminará en empate si hay una cantidad par de monedas, y si es impar ganará Sophia porque elige una cantidad mayor de monedas al empezar primera.

4. Análisis de complejidad

4.1. Introducción

Vamos a realizar mediciones de tiempo de nuestro algoritmo para comprobar empíricamente que tiene la complejidad teórica indicada anteriormente: $\mathcal{O}(n)$.

Como nuestra función *ganar_siempre* toma como parámetro un arreglo de n números (que representan monedas), podemos crear arreglos de distinto tamaño para crear simulaciones que nos servirán para medir el tiempo de ejecución de la función.

Correremos estas simulaciones para obtener los tiempos de ejecución de la función para distintas entradas, y luego haremos un ajuste por cuadrados mínimos para ver si nos alejamos de una tendencia lineal. Si el error es bajo, significa que la función es de complejidad lineal.

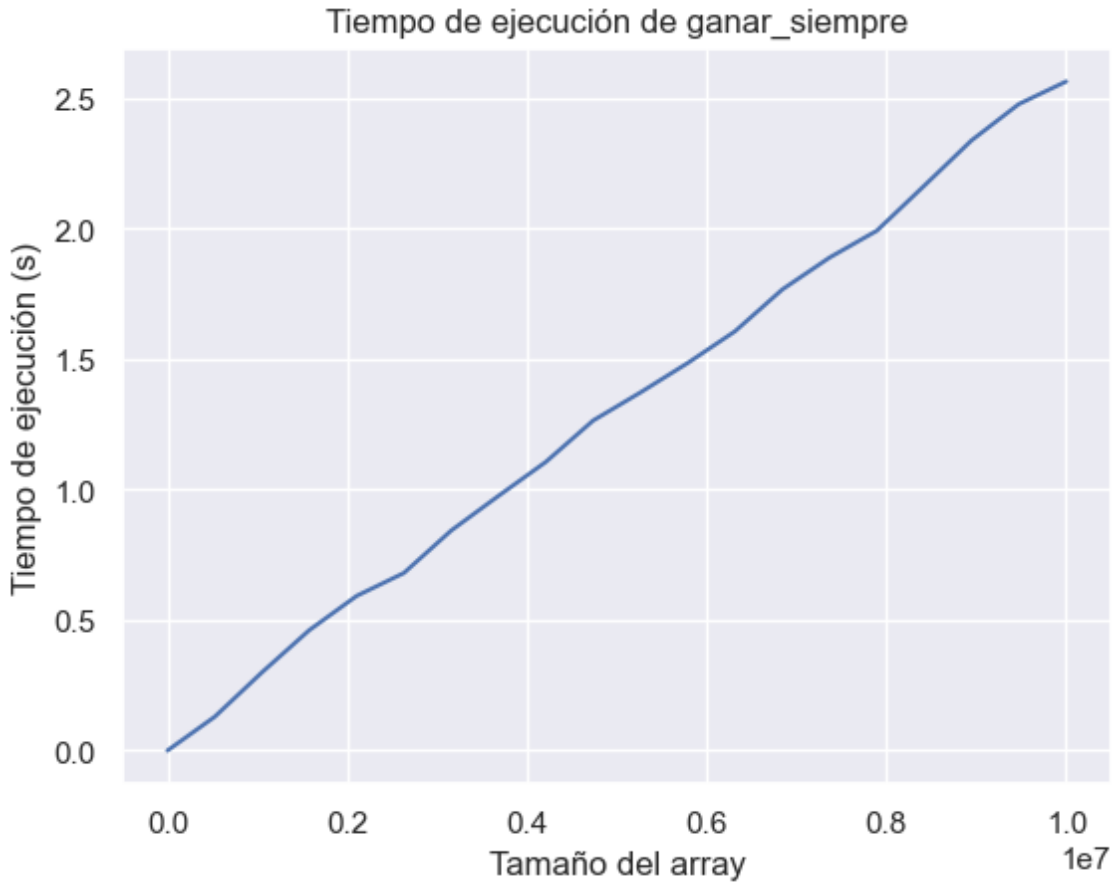
4.2. Dataset de prueba

Para probar y medir nuestra función utilizaremos distintos tamaños de entrada para evaluar cómo evoluciona el tiempo de ejecución. Tendremos 20 arreglos distintos, con tamaños variando entre 100 elementos y 10.000.000 de elementos, y los valores de esos arreglos serán números enteros entre 0 y 1000.

Más adelante realizaremos otro análisis con valores del arreglo más variados, para ver si la variabilidad de los valores de las monedas afecta a la complejidad.

4.3. Tiempo en función de la entrada

A continuación graficamos el tiempo de ejecución de *ganar_siempre* en función al tamaño de la entrada que probamos anteriormente. Podemos ver que para 5.000.000 de elementos tarda aproximadamente 1,2 segundos, y que para 10.000.000 de elementos tarda aproximadamente 2.5 segundos, lo que nos estaría indicando que el algoritmo es lineal.



4.4. Ajuste por cuadrados mínimos

Ahora vamos a ajustar por cuadrados mínimos a los resultados que obtenimos de los tiempos de ejecución de nuestra función. En nuestro caso vamos a ajustar contra la siguiente recta, para comprobar la linealidad del algoritmo:

$$y = c_1 x + c_2$$

Podemos ver que el error es bajo, por lo que ajusta muy bien a la recta:

- Valores de la pendiente y la ordenada al orígen:

$$c\_1 = 2{,}5495991986148807e-07$$

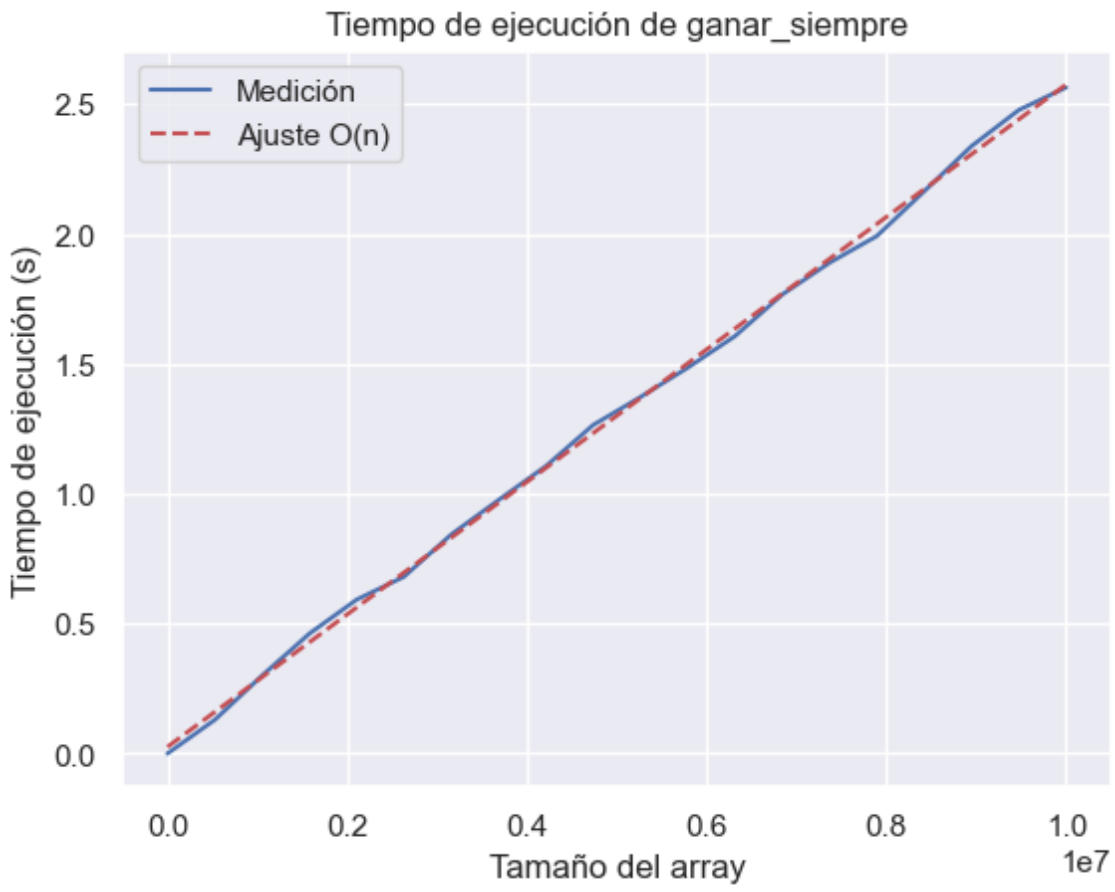
$$c\_2 = 0{,}025683563953890605$$

- Error cuadrático total:

$$0{,}011200328799622086$$

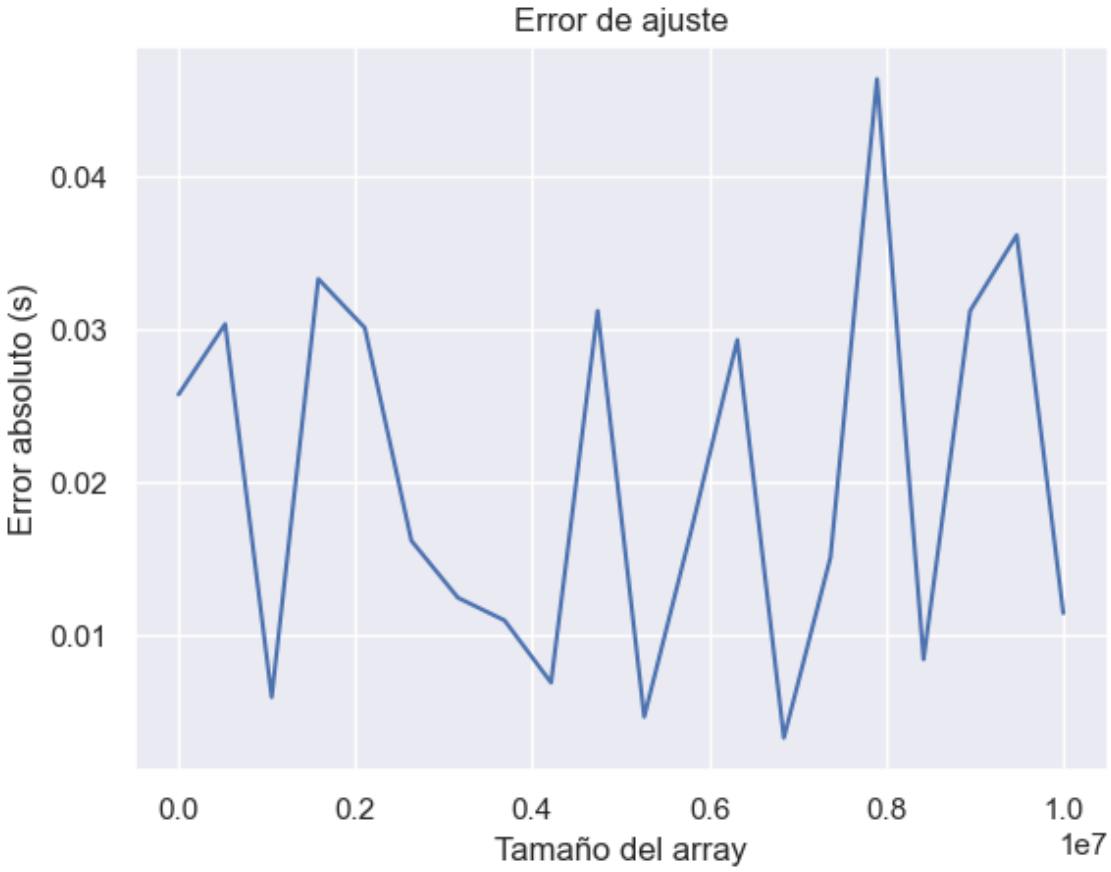
4.5. Gráfico de los datos y el ajuste

Se observa que los datos ajustan muy bien contra la recta.



4.6. Error de ajuste en función del tamaño de la entrada

En este gráfico se observa que el error de ajuste según el tamaño de la entrada es muy bajo en todos los casos, y es parejo a lo largo de todos los tamaños. Esto es otro indicio de que el algoritmo es lineal.



4.7. Variabilidad

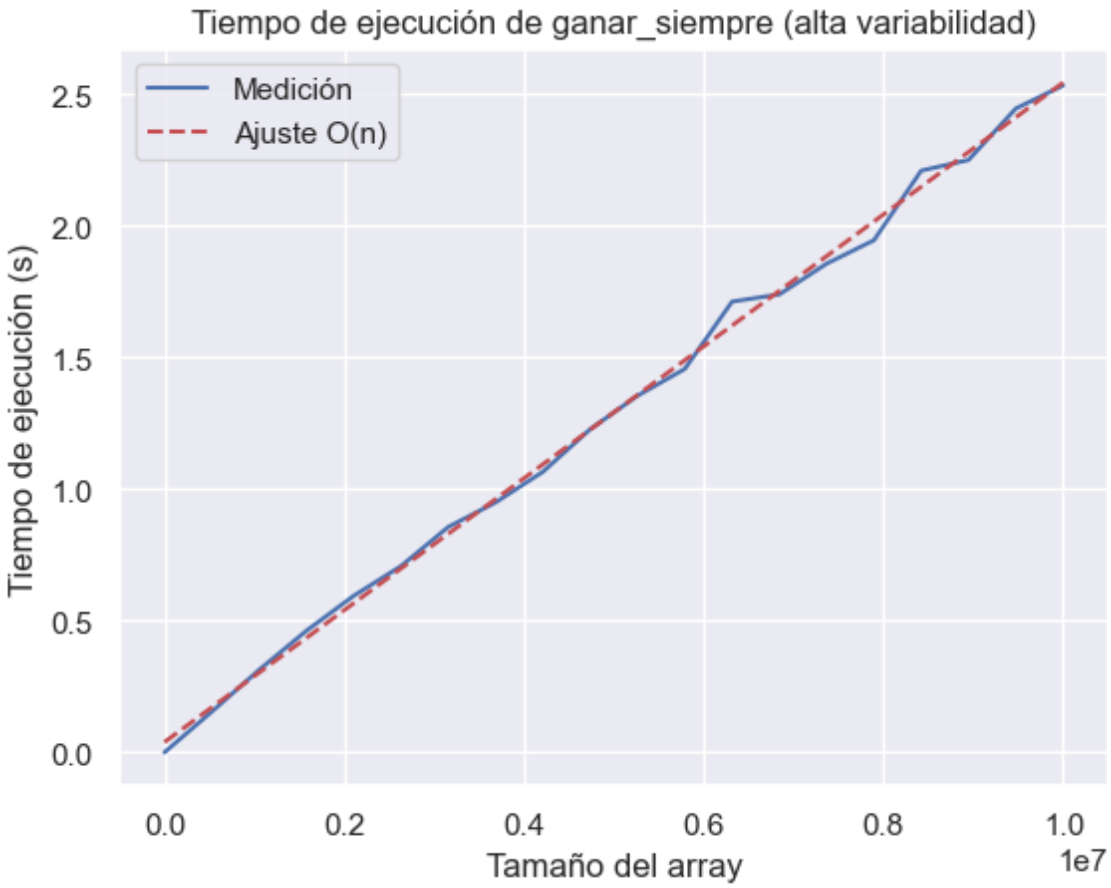
Ahora vamos a realizar un análisis extra para observar si la variabilidad de los datos que utilizamos afecta a la complejidad del algoritmo. Es decir, si cómo varían los valores de las monedas unos respecto a otros modifica el tiempo de ejecución de la función.

Para esto, vamos a repetir las mismas mediciones que hicimos antes pero con datasets de prueba distintos. Tendremos un dataset con alta variabilidad de los valores de las monedas, y otro con baja variabilidad.

4.7.1. Alta variabilidad

En este caso utilizamos arrays de monedas cuyos valores varían aleatoriamente entre 0 y 10.000.000, para testear una alta variabilidad de valores.

A continuación podemos ver el resultado del ajuste por cuadrados mínimos:



Podemos observar que el ajuste sigue siendo muy bueno, y el error cuadrático total es un poco mayor pero similar a nuestras mediciones anteriores:

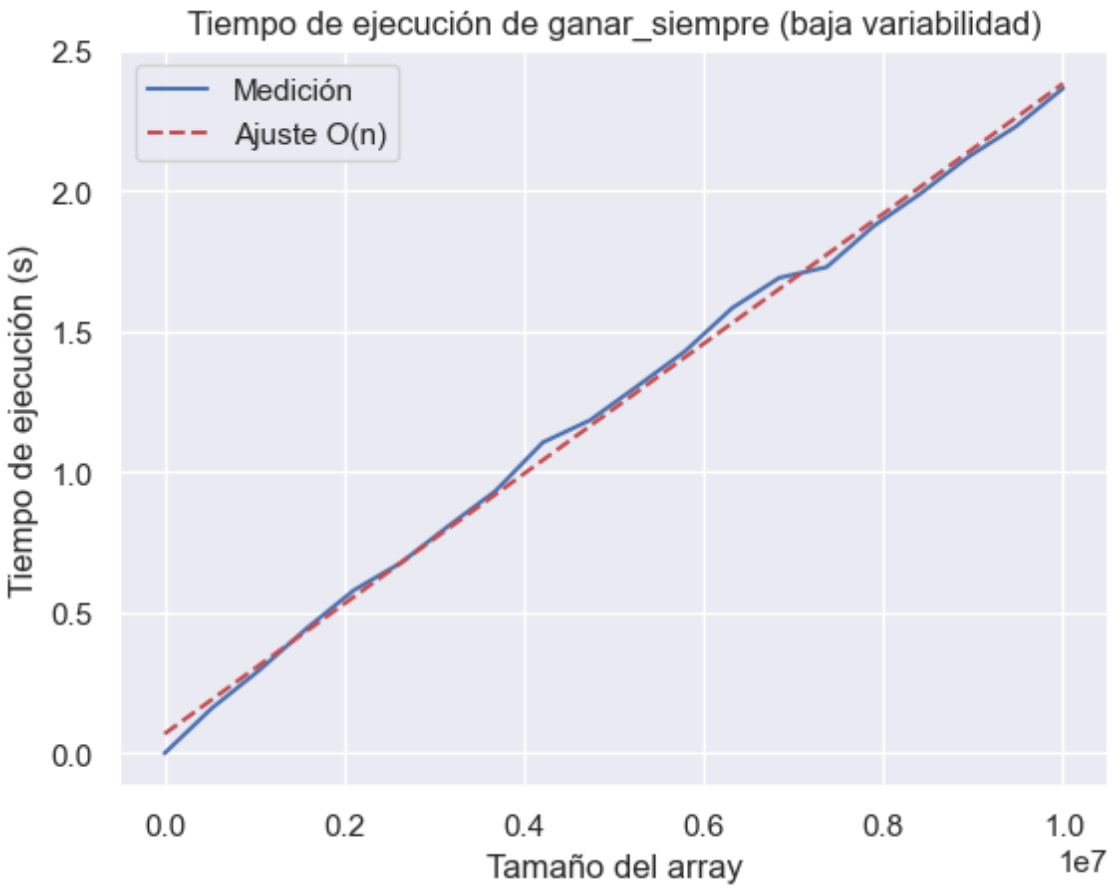
- Error cuadrático total:

  0,026536051206929232

4.7.2. Baja variabilidad

En este caso utilizamos arrays de monedas cuyos valores varían aleatoriamente entre 0 y 10, para testear una alta variabilidad de valores. Como son valores enteros, significa que sólo hay 10 valores posibles para las monedas.

A continuación podemos ver el resultado del ajuste por cuadrados mínimos:



Observamos que el ajuste también es muy bueno, y el error cuadrático total es similar:

- Error cuadrático total:

  0,02100523381599273

4.8. Resultados del análisis

Basándonos en nuestro análisis teórico de la complejidad temporal, y en este análisis empírico, podemos afirmar categóricamente que nuestro algoritmo greedy es lineal, de complejidad $\mathcal{O}(n)$.

Además, por nuestro estudio de la variabilidad de los valores de las monedas, podemos sostener que la complejidad no se ve afectada por las diferencias en la variabilidad.

5. Conclusiones

Nuestra conclusión a este trabajo práctico es que el uso de un algoritmo Greedy se adapta muy bien a las necesidades del problema en particular. Fuimos capaces de encontrar una regla básica que se asemeja al problema real, lo que resultó en un algoritmo simple y legible que además nos permitió llegar a los óptimos locales y globales de una manera muy eficiente. Nuestro análisis del problema fue correcto ya que habiendo entendido el enunciado y lo que se pedía, la solución inicial fue prácticamente la que terminamos por elegir como la definitiva.

Con las demostraciones hechas pudimos demostrar tanto la optimalidad del algoritmo propuesto (es decir, que Sophia siempre gane el juego), como la complejidad del mismo. Mediante la demostración por inducción, pudimos asegurar que nuestro algoritmo llega siempre a la solución óptima. Y con nuestras mediciones de tiempo, pudimos comprobar empíricamente que nuestro algoritmo llega a la solución de manera lineal con un error muy bajo.