.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29 - 95.06) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

8 de octubre de 2023

Alexis Herrera
104639

Edgardo Francisco Saez
104896

Franco Gazzola
105103

1. Introducción

El objetivo de este trabajo es ayudar a Scaloni a definir un plan de entrenamiento que provea la ganancia óptima.

Para lograr esto, se plantea un algoritmo de programación dinámica que obtenga el máximo beneficio obtenido por los entrenamientos, teniendo en cuenta la energía disponible desde el último descanso $(s_1, s_2, ..., s_n)$ y el esfuerzo/ganancia de cada día e_i.

1.1. Análisis del problema

Resolver el problema a través de programación dinámica requiere plantear un caso base y una ecuación de recurrencia. Primero hay que hallar sobre que variable/s se va a hacer inducción, en este caso resulta lógico hacer inducción sobre la cantidad de días n.

Pero solo con esta variable resulta imposible poder determinar para un día i el óptimo solo teniendo en cuenta el óptimo de días anteriores. Para poder plantear la ecuación de recurrencia, necesitamos una variable más, que por la forma en que se plantea la energía disponible, conviene que sea "días desde el último descanso".

Planteamos la siguiente proposición:

$P(n, d)$: Máximo beneficio en n días de entrenamiento dado que el último descanso fue tomado hace d días.

Luego el máximo beneficio para los n días de entrenamiento es $\text{máx}_{0 \leq d \leq n} P(n, d)$.

Algunas observaciones:

- Si planteamos las posibles combinaciones n/d se obtiene una matriz nxn de las cuales solo son posibles los casos de la diagonal superior de la matriz. Esto porque no se pudo haber descansado hace más días de los que hay de entrenamientos. Es decir, siempre $n > d$.

Caso base $\forall d : P(1, d) = P(1, 0) = \text{mín}(e_1, s_1)$.

La primera igualdad sucede debido a que para el primer día no hubo descansos el día anterior $(d = 0)$.

Luego como en el primer día tengo s_1 de energía, siempre se va a elegir entrenar entonces el beneficio óptimo será el mínimo entre e_1 y s_1 como lo plantea el enunciado.

Paso inductivo Consideremos el caso para $P(k, d)$. La función $P(k, d)$ se define de la siguiente manera:

$$P(k,d) = \begin{cases} P(k-1,0) + \text{mín}(s_k, e_k) & \text{si } d = 0, \\ \text{máx}_{0 \leq d \leq 1} P(k-2,d) + \text{mín}(s_k, e_1) & \text{si } d = 1, \\ P(k-1,d-1) + \text{mín}(s_d, e_k) & \text{si } d > 1. \end{cases}$$

- **Caso $d = 0$:**
  Es el caso donde no hay descansos en k días, o lo que es lo mismo que se deben entrenar los k días. Entonces, $P(k-1, 0)$ es el máximo beneficio en $k-1$ días entrenando los $k-1$ días. Por lo tanto, para obtener $P(k, 0)$ solo basta con entrenar ese día k y sumar dicho beneficio con el de $P(k-1, 0)$.
  Es decir: $P(k, 0) = P(k-1, 0) + \text{mín}(s_k, e_k)$.

- **Caso $d = 1$:**
  Es el caso donde el día de ayer se descansó.
  Esto implica que el día k se va a entrenar pues no se obtendría ningún beneficio al descansar de nuevo, porque lo máximo de energía que se va a tener es s_1.

Luego el día k se va a generar un beneficio de $min(s_k, e_1)$.
Queda obtener, cuanto es el máximo beneficio que se pudo obtener hasta ese momento y eso es equivalente a $\text{máx}_{0 \leq d \leq 1} P(k-2, d)$.
Esto es, porque el máximo beneficio tras haber descansado el dia $k-1$, es equivalente al máximo beneficio del día $k-2$ en general. Y como estamos en el caso $d = 1$ entonces d será 0 o 1.
Por lo tanto: $P(k, 1) = \text{máx}_{0 \leq d \leq 1} P(k-2, d) + \text{mín}(s_k, e_1)$.

- **Caso $d > 1$:**
  Esto sucede cuando el último descanso fue hace más de 1 día. En este caso se va a tener s_d energía disponible (por definición de s_i). Entonces el máximo beneficio que se puede generar por entrenar el día k es $min(s_d, e_k)$.
  Queda encontrar lo máximo que se pudo haber generado hasta ese momento, y esto corresponde a $P(k-1, d-1)$. La explicación es que considerando $k-1$ días y que el último descanso fue hace d días, entonces para el día k-1 el último descanso sucedió hace d-1 días. Además es el único caso de donde pudo haber llegado, pues para otro valor implicaría un último día de descanso distinto a d.

  Entonces: $P(k, d) = P(k-1, d-1) + \text{mín}(s_d, e_k)$.

2. Algoritmo y Complejidad

En esta sección se expondrá el código del algoritmo diseñado analizando y justificando su complejidad. Además se hará un análisis de si afecta la variabilidad de los valores a los tiempos y optimalidad del algoritmo planteado.

2.1. Código

A continuación se muestra el código de solución del problema.

```python
def optimo_entrenamientos(energia_demandada, energia_disponible):
    # Dadas las listas de energia demandada y disponible, devuelve la maxima
    # energia que se puede obtener entrenando
    matriz_optima = obtener_matriz_optima(energia_demandada, energia_disponible)

    return max(obtener_columna(matriz_optima, len(matriz_optima[0]) - 1))
```

```python
def obtener_matriz_optima(e_de, e_dis):
    # e_de: energia demandada
    # e_dis: energia disponible
    # m: matriz optima[descanse hace i dias][estoy en el dia i]

    m = [[0 for _ in range(len(e_de))] for _ in range(len(e_dis))]

    # Si solo tengo un dia lo mejor es hacer el entrenamiento con toda la energia
    disponible.
    m[0][0] = min(e_de[0], e_dis[0])

    # Si estoy en el dia 2 y el anterior descanse, tengo toda la energia para hacer
     el entrenamiento.
    m[1][1] = min(e_de[1], e_dis[0])

    # LLeno la matriz para el caso donde nunca se toma un descanso.
    for i in range(1, len(m[0])):
        m[0][i] = m[0][i - 1] + min(e_de[i], e_dis[i])

    # Lleno la matriz por columnas partiendo desde el dia 2 habiendo descansado
    # hace 1 dia.
    for j in range(2, len(m)):
        for i in range(1, j + 1):
            # Caso descanse el dia anterior.
            if i == 1:
                m[i][j] = min(e_dis[i - 1], e_de[j]) + max(obtener_columna(m, j -
    2))
            # Caso descanse hace i dias, con i > 1.
            else:
                m[i][j] = min(e_dis[i - 1], e_de[j]) + m[i - 1][j - 1]

    return m
```

2.2. Complejidad temporal

La complejidad total del algoritmo se debe calcular teniendo en cuenta que al ser un problema de programación dinámica todas las soluciones validas deben ser calculadas.

Si armamos una matriz de $n \cdot m$ siendo n la cantidad de días que se pueden entrenar y m la cantidad de días que pasaron desde el ultimo descanso. Los valores dentro de la misma serán la máxima ganancia que se puede obtener en ese día.

$$\begin{bmatrix} g_{0,0} & g_{0,1} & g_{0,2} \\ - & g_{1,1} & g_{1,2} \\ - & - & g_{2,2} \end{bmatrix} \tag{1}$$

La diagonal inferior izquierda no forma parte de la matriz ya que como se menciona en la sección 1.1 no se pudo haber descansado hace más días de los que hay de entrenamientos.

Teniendo en cuenta lo anterior como mínimo tendremos que recorrer $n * m$ casilleros donde m es igual a $\frac{n}{2}$. Ya con esto podríamos tener una cota de $\mathcal{O}(\frac{n^2}{2})$ que es equivalente a $\mathcal{O}(n^2)$.

Cuando la cantidad de días descansados es igual a 0, es decir pararnos sobre la primer fila. Tendremos que recorrer n casilleros y en cada paso sumar la solución del día anterior, previamente calculada, al mínimo entre la ganancia disponible y la energía disponible. Ambas operaciones son de tiempo constante resultando en una complejidad de $\mathcal{O}(n)$.

Cuando la cantidad de días que pasaron luego del ultimo descanso es igual a 1, en nuestra matriz estos son todos los valores de la segunda fila. Para encontrar el óptimo de estos días, como se explicó en la sección 1.1, hay que referirse al día anterior al descanso y encontrar el máximo obtenido en ese día. Esto es equivalente a buscar el máximo en la columna correspondiente a ese día.

Si para cada columna tenemos en cuenta que a medida que n aumenta la cantidad de elementos en ella también lo hace, por estar la diagonal inferior vacía. La columna mas larga donde buscar el máximo tendría n - 2 elementos, ya que esta es la que vería el ultimo día cuando calcule su fila 1. Por lo tanto, al ser la operación de buscar el máximo lineal, su complejidad sería $\mathcal{O}(n-2)$. Ahora solo queda tener en cuenta cuantas veces se ejecutara la operación de obtener el máximo. Sabiendo que recién el tercer día puedo mirar hacia atrás dos días, la cantidad de veces que se ejecutara la búsqueda del máximo es n - 2. Ahora la complejidad resultante para este caso resulta en buscar el máximo n - 2 veces, lo que se traduce como: $\mathcal{O}(n-2) \cdot \mathcal{O}(n-2)$.

Finalmente cuando la cantidad de días de descanso es mayor a 1, para cada día hay que acudir a la ganancia generada en la posición diagonal izquierda superior y a esta sumarle el mínimo entre la energía disponible y la energía demandada. Ambas operaciones son de tiempo constante ya que una fue calculada previamente y la otra es una comparación. Como sabemos que la fila 1 y 2 no se recorre para este caso podemos decir qué se harán k operaciones de tiempo constante con $k < n \cdot m$.

Ahora analizando todo en conjunto la complejidad total debería calcularse como:

$$\mathcal{O}(f(n)) = \mathcal{O}(n-2) \cdot \mathcal{O}(n-2) + \mathcal{O}(n) + \mathcal{O}(k),\ k < n^2$$

Como

$$\mathcal{O}((n-2) \cdot (n-2)) = \mathcal{O}(n^2)$$

Si acotamos, podemos concluir que la complejidad del algoritmo es:

$$\mathcal{O}(f(n)) = \mathcal{O}(n^2)$$

2.3. Complejidad espacial

El algoritmo planteado usa una matriz de $n \cdot n$. Por lo tanto la complejidad espacial de nuestro algoritmo es de $\mathcal{O}(n^2)$.

2.4. Variabilidad de los valores

Para hacer una análisis de la variabilidad de los valores hay que tener en cuenta que al haber diseñado un algoritmo de programación dinámica el mismo esta sujeto al principio de optimalidad que establece que una solución óptima al problema global se puede construir a partir de soluciónes óptimas a subproblemas más pequeños. Esto significa que es necesario evaluar todas las posibles combinaciones de subproblemas para determinar cuál es la mejor solución global.

Siendo que hay que explorar todas las soluciones la complejidad temporal del algoritmo se mantendría constante sin importar la variabilidad de los valores.

3. Casos de prueba

A continuación se muestran los ejemplos mencionados anteriormente, verificando que el programa devuelve lo esperado. También se muestran los ejemplos dados por la cátedra.

En esta ocasión utilizamos la librería PyTest para ejecutar los casos de prueba, por lo que para ejecutarlos ingresamos el siguiente comando: `pytest -v`.

```
(env) ✘ alexis@alexis-pc ~/Documents/fiuba/tda/TDA-TP1-Greedy/TP2-PD  main pytest -v
============================================================================================================
platform linux -- Python 3.10.12, pytest-7.4.1, pluggy-1.3.0 -- /home/alexis/Documents/fiuba/tda/
cachedir: .pytest_cache
rootdir: /home/alexis/Documents/fiuba/tda/TDA-TP1-Greedy/TP2-PD
plugins: anyio-4.0.0, timeout-2.1.0
collected 23 items

test_tp2.py::test_3_elementos PASSED
test_tp2.py::test_10_elementos PASSED
test_tp2.py::test_10_bis_elementos PASSED
test_tp2.py::test_10_todo_entreno PASSED
test_tp2.py::test_50_elementos PASSED
test_tp2.py::test_50_bis_elementos PASSED
test_tp2.py::test_100_elementos PASSED
test_tp2.py::test_500_elementos PASSED
test_tp2.py::test_1000_elementos PASSED
test_tp2.py::test_5000_elementos PASSED
test_tp2.py::test_3_elementos_lista PASSED
test_tp2.py::test_10_elementos_lista PASSED
test_tp2.py::test_10_bis_elementos_lista PASSED
test_tp2.py::test_10_todo_entreno_lista PASSED
test_tp2.py::test_50_elementos_lista PASSED
test_tp2.py::test_50_bis_elementos_lista PASSED
test_tp2.py::test_100_elementos_lista PASSED
test_tp2.py::test_500_elementos_lista PASSED
test_tp2.py::test_1000_elementos_lista PASSED
test_tp2.py::test_5000_elementos_lista PASSED
test_tp2.py::test_no_descansa PASSED
test_tp2.py::test_descansa_1er_dia PASSED
test_tp2.py::test_descansa_2do_dia PASSED
```

Figura 1: Casos de Prueba

A continuación se harán seguimientos de ejemplo donde el óptimo de la solución varía según el caso:

- **Se entrena todos los días**

Energia demandada	75	60	54
Energia disponible	80	59	58
dias vs ultimo_descanso	Día 1	Día 2	Día 3
0	75	134	188
1		60	129
2			114

Figura 2: Entreno todos los días

En este primer escenario, vamos a analizar por qué la mejor solución es entrenar todos los días posibles.

Para comprender por qué es óptimo entrenar todos los días, podemos observar que las cantidades de energía disponibles a lo largo de todos los días de entrenamiento son muy cercanas a las demandas de energía requeridas por cada sesión de entrenamiento. Esto nos lleva a deducir que no sería conveniente descansar un día para recuperar la energía máxima.

Comenzando el primer día con una energía disponible de 80 unidades, obtendremos una ganancia de 75 unidades. Al continuar al segundo día, disponemos de 59 unidades de energía y una sesión de entrenamiento que demanda 60 unidades, lo que nos da un total de 134 unidades, esto nos muestra que no valdría aumentar la energía disponible, dado que solo obtendríamos 1 punto mas de ganancia, pero perderíamos lo entrenado en el primer día. Luego, en el tercer y último día, obtenemos una ganancia de 54 unidades, alcanzando así una ganancia máxima de 188 unidades en total.

- **Se descansa el primer día**

Energia demandada	30	100	50
Energia disponible	120	69	65
dias vs ultimo_descanso	Día 1	Día 2	Día 3
0	30	99	149
1		100	80
2			150

Figura 3: Descanso el primer día

Primero, analicemos de manera analítica por qué la mejor solución consiste en descansar el primer día y luego entrenar los dos días siguientes.

Para comprender por qué debemos descansar el primer día, es importante observar que la demanda de energía del primer día es la más baja en comparación con los otros entrenamientos. Además, la energía inicial es aproximadamente el doble que la del segundo día. Por lo tanto, para obtener la solución óptima del problema, debemos esforzarnos por obtener la mayor ganancia posible en los últimos dos entrenamientos.

Si comenzamos descansando el primer día, podremos entrenar el segundo día con una energía disponible de 120 unidades, lo que nos permitirá obtener una ganancia de 100 unidades. Luego, continuando con el tercer día, podemos obtener una ganancia adicional de 50 unidades, lo que nos lleva a un máximo de 150 unidades de ganancia en total.

En cambio si decidimos entrenar todos los días no podremos aprovechar el beneficio del día 2 que son 100 unidades, solo podremos aprovechar 69 de esos 100, haciendo que al final del día 3 solo obtengamos 149 de beneficio.

- **Se descansa el segundo día**

Energia demandada	50	20	65
Energia disponible	60	10	5
dias vs ultimo_descanso	Día 1	Día 2	Día 3
0	50	60	65
1		20	110
2			30

Figura 4: Descanso el segundo día

En este caso, analizaremos por qué es crucial descansar en el segundo día de entrenamiento. Hay dos aspectos clave a considerar: la energía disminuye rápidamente y el segundo día de entrenamiento genera ganancias menores en comparación con los otros días. Por lo tanto, es esencial aprovechar al máximo los entrenamientos del primer y tercer día, ya que estos proporcionarán las mayores ganancias totales.

Si comenzamos el primer día con una energía disponible de 60 unidades, obtendremos una ganancia completa de 50 unidades. Sin embargo, si continuamos entrenando los dos días siguientes, solo conseguiremos una ganancia total de 65 unidades debido al rápido agotamiento de la energía disponible. En cambio, si decidimos descansar en el segundo día, podremos entrenar el tercer día con la energía inicial, maximizando así el provecho de los entrenamientos del primer y tercer día. Esto resultará en una ganancia total de 110 unidades, lo cual es significativamente mayor.

A continuación se presentan capturas del código de los ejemplos mencionados con nuestro algoritmo.

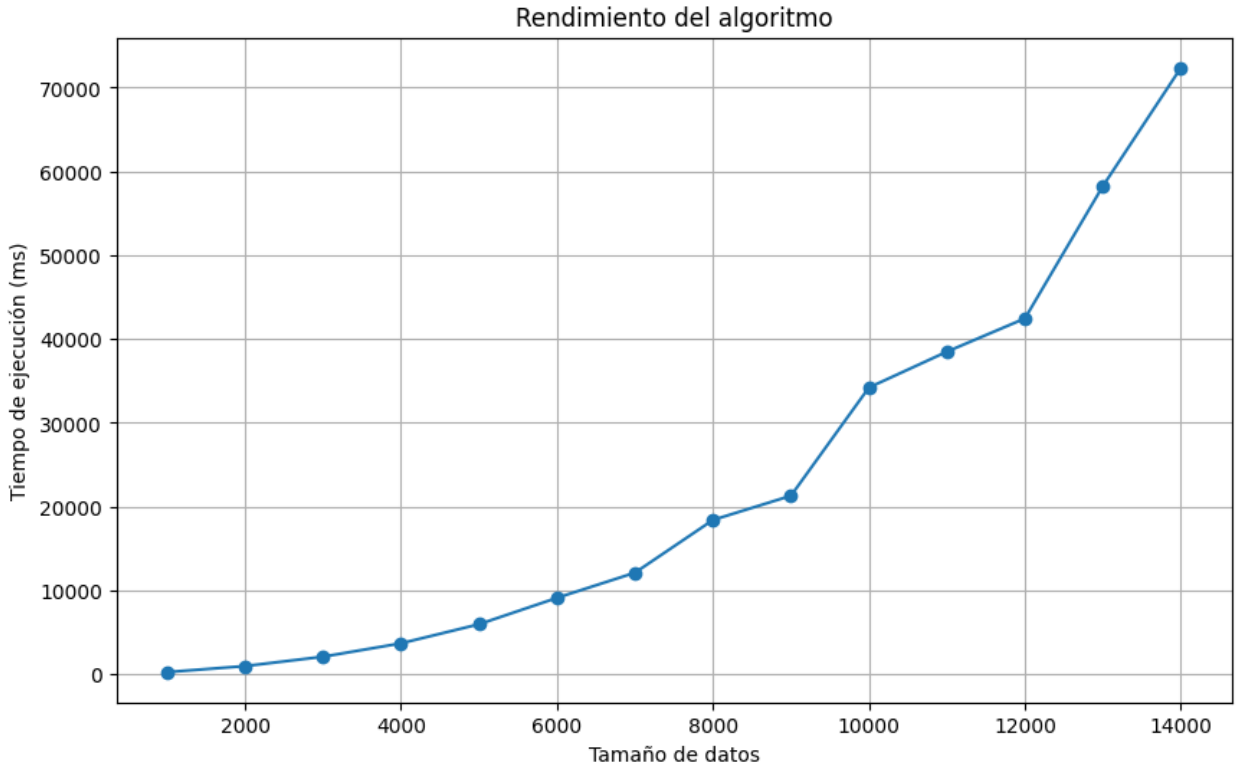
```
Alexis Herrera
def test_no_descansa():
    energia_demandada = [75, 60, 54]
    energia_disponible = [80, 59, 58]

    assert optimo_entrenamientos(energia_demandada, energia_disponible) == 188
    assert obtener_lista_entrenamientos(energia_demandada, energia_disponible) == ['Entreno', 'Entreno', 'Entreno']

Alexis Herrera
def test_descansa_1er_dia():
    energia_demandada = [30, 100, 50]
    energia_disponible = [120, 69, 65]

    assert optimo_entrenamientos(energia_demandada, energia_disponible) == 150
    assert obtener_lista_entrenamientos(energia_demandada, energia_disponible) == ['Descanso', 'Entreno', 'Entreno']

Alexis Herrera
def test_descansa_2do_dia():
    energia_demandada = [50, 20, 65]
    energia_disponible = [60, 10, 5]

    assert optimo_entrenamientos(energia_demandada, energia_disponible) == 110
    assert obtener_lista_entrenamientos(energia_demandada, energia_disponible) == ['Entreno', 'Descanso', 'Entreno']
```

Figura 5: Tests

4. Mediciones

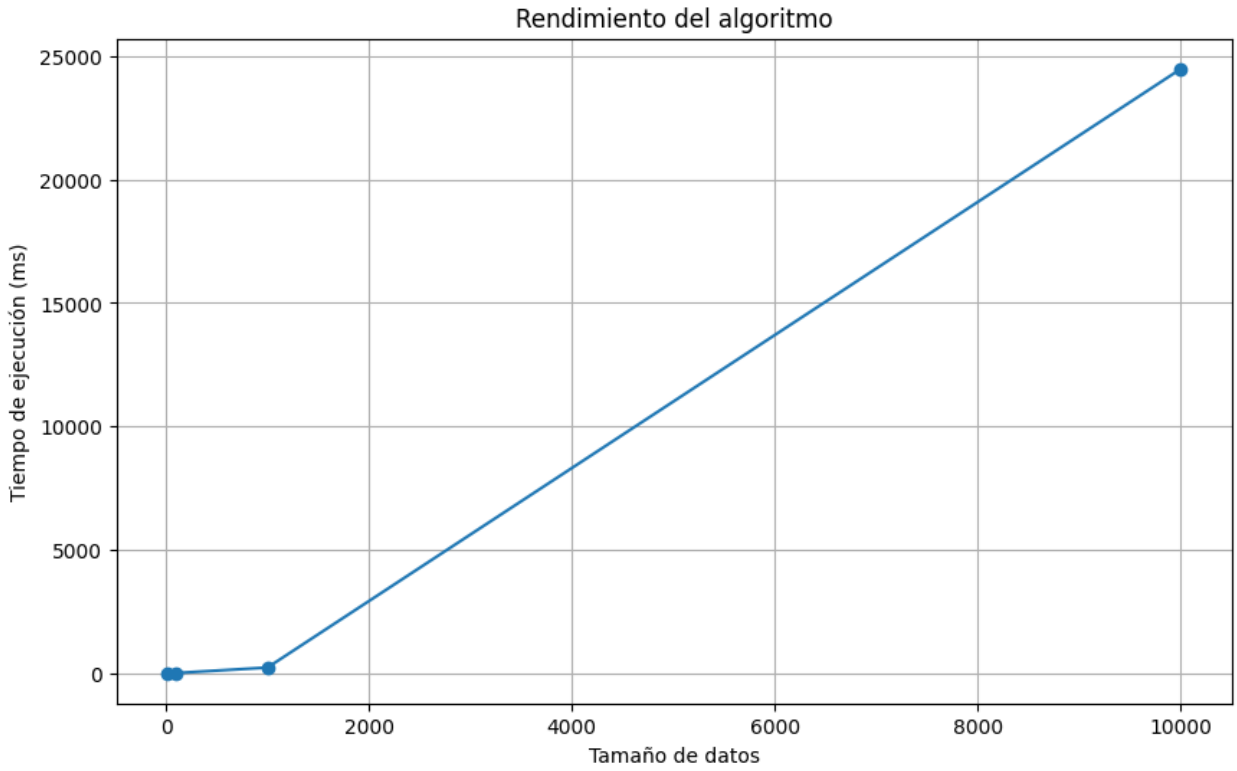
Llevamos a cabo mediciones de tiempo utilizando arreglos de diferentes longitudes. En cada uno de estos casos, repetimos el proceso en múltiples ocasiones para obtener valores de tiempo de ejecución consistentes y estables.

4.1. Medición 1



En el primer escenario, hemos creado diversas series de datos con incrementos de 1000 elementos, abarcando un rango desde 0 hasta 15.000 elementos. Al observar el gráfico resultante, notamos como, al comienzo de la serie, se evidencia una tendencia cuadrática en el rendimiento del algoritmo la cual se va haciendo mas evidente a medida que la cantidad de elementos aumenta. Debido a esto podemos decir que la complejidad del algoritmo es $\mathcal{O}(n^2)$.

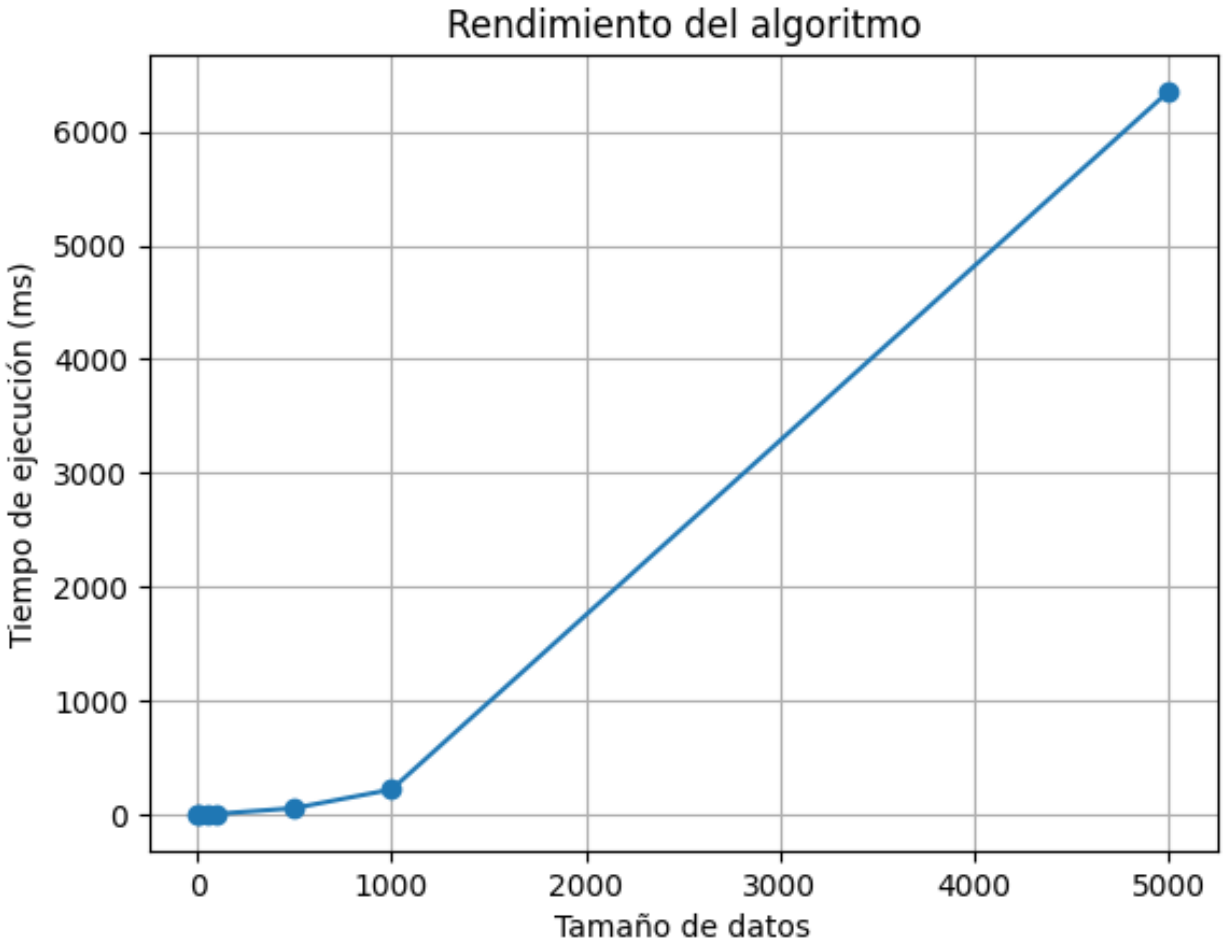
4.2. Medición 2



En este escenario, hemos generado series de datos de longitudes diversas, aumentando de manera exponencial desde 10 hasta 10.000 elementos.

En el gráfico, se aprecia un cambio abrupto positivo en la pendiente entre el punto donde se calculó el tiempo con 1000 elementos y el punto con 10.000 elementos. Este fenómeno es coherente con lo planteado en el gráfico previo, donde se observa un crecimiento con complejidad $\mathcal{O}(n^2)$.

4.3. Medición 3



Para concluir, llevamos a cabo las mediciones utilizando las muestras proporcionadas por la cátedra, las cuales contenían 3, 10, 50, 100, 500, 1000 y 5000 elementos para su análisis.

En el eje x, podemos observar el gran incremento en la pendiente de la recta entre el punto con 1000 y 5.000 elementos

4.4. Análisis de Complejidad

Para concluir, a partir de los casos planteados, podemos afirmar de manera concluyente que el algoritmo propuesto presenta una complejidad evidente de $\mathcal{O}(n^2)$.

4.5. Generación de datos

En ambos ejemplos los elementos fueron generados por los valores pseudoaleatorios del lenguaje (el módulo `random`).

A continuación se muestra cómo generamos los diferentes conjuntos de datos para diferentes cantidades de elementos.

```
s = [random.uniform(1, 100) for _ in range(n)]
s_descendente = sorted(s, reverse=True)
e = [random.uniform(1, 100) for _ in range(n)]
```

5. Conclusiones

Después de un análisis minucioso del problema, llegamos a la conclusión de que no era posible resolverlo únicamente teniendo en cuenta la cantidad de días, n. Para encontrar la solución óptima, introducimos otra variable llamada "días desde el último descanso", obteniendo así una ecuación de recurrencia que encontraba la solución óptima al problema explorando todas las combinaciones de los subproblemas posibles para determinar la ganancia máxima.

Además, pudimos concluir que la complejidad del algoritmo es de orden $\mathcal{O}(n^2)$, y corroboramos este resultado con un análisis teórico y múltiples gráficos que lo confirman.