Teoría de Algoritmos

(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

9 de octubre de 2023

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

Índice

1. Introducción

En este informe haremos un estudio del problema planteado, qué días de entrenamiento descansar para así maximizar la ganancia de los mismos. En primer lugar, vamos a explicar el algoritmo al que llegamos, con su ecuación de recurrencia Luego analizar ciertas optimizaciones que podemos hacer del mismo, estudiar que sucede si variamos los distintos parámetros, y por último realizar un análisis de complejidad temporal del algoritmo planteado, con sus respectivas pruebas empíricas.

2. Análisis del problema

2.1. El problema y su algortimo

Para resolver el problema, se tiene que buscar la manera de descansar en los momentos más convenientes de forma que los entrenamientos posteriores se aprovechen al máximo. Inicialmente planteamos una versión de programación dinámica con memorización en un vector por días "recorridos", donde para cada día usábamos el mayor entre las opciones de haber descansado ayer o no. Esta memorización vectorial nos permite reutilizar los valores previamente calculados para los casos siguientes ahorrando costo computacional a cambio de complejidad espacial, en este caso vamos a poder reutilizar las ganancias máximas para n días en los días posteriores sin tener que rehacer toda la recurrencia. Para esto hacíamos las siguientes fórmulas:

```
entrenado_ayer = días[n-1] + min(e[i], s[días_consecutivos])
descansado_ayer = días[n-2] + min(e[i], s[0])
```

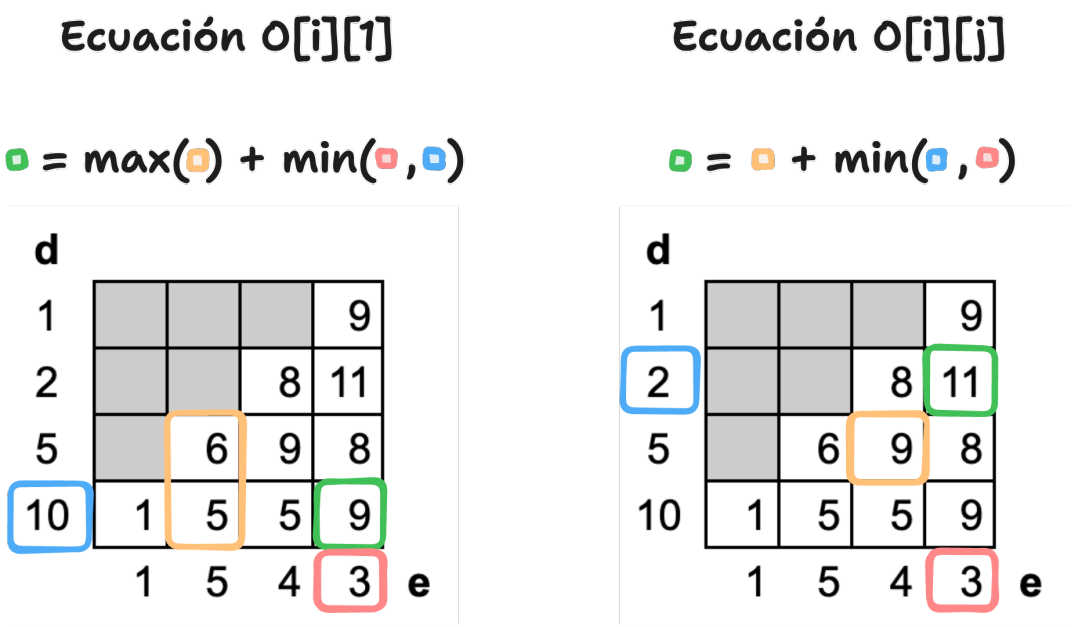
Aunque a priori esto parece una buena aproximación, nos encontramos con que no siempre aplica esta idea de que el máximo para 3 días sirva para los primeros 3 días de una planificación más larga. Si solamente se va a entrenar 3 días, el último de estos es conveniente gastar toda la energía restante, por mas poca que sea, antes que descansar, (de todas formas, después de este entreno finaliza la planificación), pero si mañana hubiera otro entrenamiento tal vez convenía descansar para aprovechar el siguiente. Es por esto que en la memorización falta una dimensión más, los días consecutivos entrenados. Como se ve en las fórmulas iniciales, el valor de días_consecutivos se iba actualizando sobre la marcha y no formaba parte de la memorización haciendo que cuando se consultaba el valor para un día previo, no se supiera que tanta energía tenía disponible el equipo.

La solución al problema la planteamos con una matriz de memorización donde por cada día calculado guardamos los valores que iría acumulando según los entrenamientos consecutivos, para esto las ecuaciones de recurrencia quedan de la siguiente manera, donde $O[i][j]$ equivale a la matriz de soluciones óptimas en la fila i, columna j:

$$O[i][j] = 0 \ \forall \ \{i, j : j = 0 \vee i = 0\}$$

$$O[i][1] = Max(O[i-2][j] \ \forall \ 0 > j > i) + Min(e[i], s[0]) \ \forall \ i > 0$$

$$O[i][j] = O[i-1][j-1] + Min(e[i], s[j]) \ \forall \ \{i, j : j > 1, i > j\}$$



Lo que hacemos en este caso es, por cada día, guardar todos los posibles casos de días consecutivos (resulta en una matriz triangular inferior porque es imposible tener más días consecutivos que días totales). Cada día existe la posibilidad de que sea el primer día consecutivo entrenando, o sea ayer fue descanso, [segunda ecuación] donde lo óptimo sería a la opción máxima de anteayer más el entrenamiento de hoy con la energía disponible de no tener entrenamientos acumulados. Las otras opciones para cada día es haber entrenado, y para esto se tiene que sumar a cada opción de ayer el entrenamiento de hoy con el cansancio siguiente al que tenía esa opción ayer [tercera ecuación].

Una vez tenemos la matriz completa con todas las posibilidades calculadas, para encontrar el óptimo para n días solo hace falta ir a la columna n y buscar la opción con mayor ganancia, algo similar a lo que se hace cuando se va a descansar el día siguiente, ya que en ambos casos la energía restante de los jugadores deja de importar (porque se acaba o porque se reiniciará) y solo importa ganancia total.

Una vez sabemos que opción es la indicada, para reconstruir el cronograma y marcar cada día como "Entreno" o "Descanso" lo que hacemos es ver cuantos días consecutivos llevaba en esta opción elegida según su posición en el eje d y de esta forma conocemos como entreno los últimos "i" días, sabemos que el día anterior a ese fue un descanso y que para ese descanso habíamos elegido como día previo la opción más alta, por lo que nuevamente buscamos cual era y sabemos cuantos entrenos consecutivos habían llegado hasta ahi. De esta forma continuamos rearmando el camino desde el final "trazando" las diagonales desde el día óptimo anterior a cada descanso.

2.2. Optimización del problema

Ahora que ya sabemos como calcular todas las opciones de la matriz, y cómo reconstruir el cronograma ideal, notamos que llevar a cabo todas las posibilidades no tenía mucho sentido, porque aunque sea imposible saber en un día n que opción seria mejor entre una que tiene más entrenamientos consecutivos pero lleva más ganancia y otro más descansado pero con una ganancia inferior debido a que esto dependerá de cómo sean los siguientes días, si podemos saber que hay casos donde nunca podrán "recuperar" lo perdido contra la mejor opción del momento. Esto se da cuando hay una opción con menor ganancia pero mayor cansancio.

Para demostrar este supuesto, cubrimos los posibles días siguientes de dos opciones que cumplan

con el mismo.

- A: energía = 10, ganancia = 5
- B: energía = 12, ganancia = 6

Para el caso donde el día siguiente se entrene un valor menor a la energía de ambos, se mantienen las diferencias tanto de energía como de ganancia, haciendo que para el día posterior a ese el caso tenga el mismo comportamiento al que se está cubriendo

- esfuerzo = 5
- A: energía = 5, ganancia = 10
- B: energía = 7, ganancia = 11

Para el caso donde el día siguiente se entrene un valor mayor a la energía de A pero menor a la de B, la energía de A se agota mientras que B sigue teniendo algo a la vez que la diferencia en ganancia aumenta. En cualquier caso posterior a este, esto no va a hacer más que incrementarse ya que hasta que A no descanse, este no podrá nunca sumar más ganancia

- esfuerzo = 11
- A: energía = 0, ganancia = 15
- B: energía = 1, ganancia = 17

Para el caso donde el entrenamiento sea mayor a la energía de ambos, estos se quedaran igualados en esta, pero la diferencia en ganancia se aumentara aun mas que en el caso anterior, dejando una paridad en energía pero clara ventaja en ganancia para el B

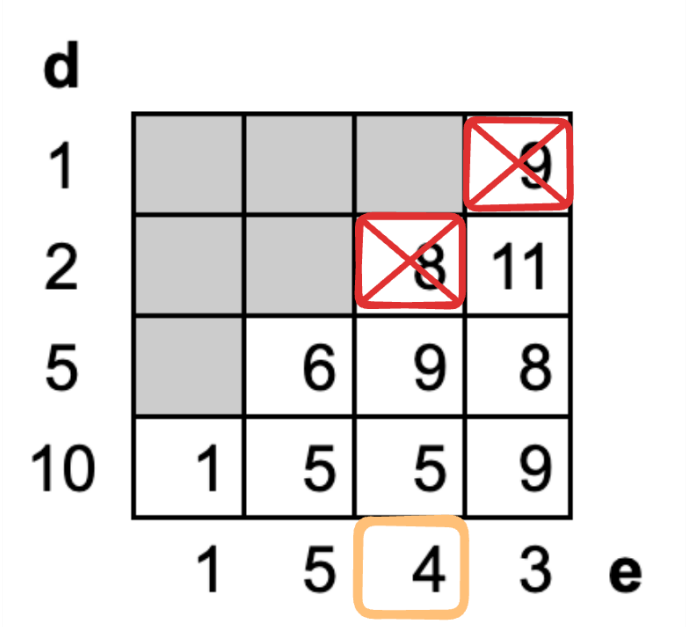
- esfuerzo = 15
- A: energía = 0, ganancia = 15
- B: energía = 0, ganancia = 18

El último caso posible es el de que descansen, acá se "resetea" la energía manteniendo la misma ganancia, haciendo que la ventaja la siga manteniendo el B.

- A: energía = 20, ganancia = 5
- B: energía = 20, ganancia = 6

Habiendo comprobado por inducción que para todo día n+1 la opción B sigue siendo más óptima, podemos afirmar que toda opción con más días acumulados que la de mayor ganancia es descartable, por lo que en el código no guardamos las opciones posteriores a esta.

El resultado termina siendo en que solo continúan siendo completadas las diagonales que tienen posibilidades bajando sustancialmente la cantidad de operaciones y de memoria utilizadas. En la imagen de abajo se ve como en el tercer día, se descarta la opción de ir 3 días consecutivos haciendo que en el día siguiente también te ahorres ese cálculo. (Más información de esto en el apartado de complejidad)



3. Algoritmo planteado

3.1. Análisis del código

Primero comenzamos por leer los respectivos esfuerzos y energías disponibles para cada día del cronograma de entrenamiento y guardarlos separadamente para su posterior uso.

Luego, comenzamos con la operación para calcular la ganancia máxima posible dado un cronograma de entrenamiento. Para ello, vamos a utilizar el principio de memoization para guardar los resultados de los cálculos que fuimos haciendo previamente, y así, evitar tener que repetirlos en el futuro. En otras palabras, estamos ahorrando grandes cantidades de tiempo a cambio de más espacio de almacenamiento.

Nuestro método de almacenamiento va a ser una matriz donde las filas son la energía, la cual varía según la cantidad de días seguidos entrenando, y las columnas son los esfuerzos de cada día. Cada celda representa la ganancia obtenida por esa secuencia de entrenamientos/descansos. En la matriz vamos a guardar todas las posibles combinaciones para cada día. Cabe recalcar que en realidad solo utilizamos media matriz, la sección triangular inferior. Esto se debe a que por ejemplo en el primer día, no va a ser necesario calcular toda la columna de energías disponibles ya que en el primer día como máximo entrenó solo un día.

Comenzamos agregando el primer día a la matriz tomando el mínimo entre el esfuerzo y la energía disponible, ya que es la única opción posible para ese día. Luego, pasamos a los días donde hay múltiples caminos y opciones. La primera opción a calcular sería la base de la columna donde significa que descansó el día anterior, por lo tanto, tomamos el máximo valor de antes de ayer y le sumamos el mínimo entre el esfuerzo de ese día y el primer elemento de nuestra lista de energías.

Después, para el día actual, tomamos cada posibilidad del día anterior y le sumamos el mínimo entre el esfuerzo de ese día y la energía disponible teniendo en cuenta cuantos días seguidos llevamos entrenando. En otras palabras, tomamos todos los caminos creados anteriormente los extendimos entrenando un día más. Con caminos nos referimos a las secuencias de entrenamientos o descanso hasta llegar a ese día con cierta ganancia, tal como dijimos previamente.

Una vez terminado esto, significa que ya ampliamos todos los caminos para ese día, entonces, nos guardamos esos caminos y continuamos iterando hasta llegar al último día.

Por otro lado, queremos recalcar ciertas partes de nuestro código que pusimos para optimizar este algoritmo. A medida que extendíamos los posibles caminos, nos guardábamos una referencia al mejor camino de ese día para luego solo guardar la columna hasta ese elemento, incluyéndolo. Esto genera que se utilice menos espacio en memoria ya que la columna es menor, y, además, evita que el algoritmo continúe extendiendo caminos que no van a ser óptimos.

```
def ganacias_por_dia(energia, descanso, optimizar=True):
  dias = []

  for dia in range(len(energia)):
    if dia == 0:
      dias.append([min(energia[0], descanso[0])])
      continue

    posibilidades_hoy = []

    ante_ayer = dias[dia-2][-1] if dia >= 2 else 0
    ganacia_hoy_descansado = min(energia[dia], descanso[0])
    posibilidades_hoy.append(ante_ayer + ganacia_hoy_descansado)

    mejor_posibilidad = 0

    posibilidades_ayer = dias[dia-1]
    for dias_acumulados, posibilidad_ayer in enumerate(posibilidades_ayer):
      nueva_posibilidad = posibilidad_ayer + min(energia[dia], descanso[
    dias_acumulados + 1])
      posibilidades_hoy.append(nueva_posibilidad)

      if posibilidades_hoy[mejor_posibilidad] < nueva_posibilidad:
        mejor_posibilidad = dias_acumulados + 1

    if optimizar:
      posibilidades_hoy = posibilidades_hoy[:mejor_posibilidad+1]
    dias.append(posibilidades_hoy)

  return dias
```

3.2. Complejidad temporal de nuestro algoritmo

Considerando que el tamaño del problema es n, para una n cantidad de días a entrenar entonces tanto nuestro vector 'e' como nuestro vector 's' tienen el tamaño n. Partiendo de esa base, nuestro algoritmo comienza recorriendo todos los días, luego realizamos unas operaciones $\mathcal{O}(1)$ (comparaciones, sumas y asignaciones) , y para cada día recorre todas las posibilidades del día anterior. Como explicamos anteriormente, terminamos recorriendo de n días, $n^2/2$ posibilidades de distintos entrenamientos, ya que solo llenamos diagonal inferior de la matriz. Al recorrer todas las posibilidades del día anterior, realizamos todas operaciones $\mathcal{O}(1)$ (comparaciones, sumas y asignaciones). Por lo tanto, terminamos realizando $n^2/2$ veces operaciones $\mathcal{O}(1)$, en consecuencia nuestro algoritmo tendría una complejidad de $\mathcal{O}(n^2/2)$, que por la notación Big-O esta acotada por $\mathcal{O}(n^2)$. Luego, la optimización que realizamos en términos de notación Big-O, su cota superior sigue siendo $\mathcal{O}(n^2)$ ya que en el peor de los casos no podemos realizar ningún tipo de poda, sin embargo en la vida real el algoritmo es mucho mas eficiente, en complejidad algorítmica y espacial, ya que para cada rama que cortamos, son menos días a recorrer en los días siguientes. Además, para obtener el máximo de hace 2 días en el caso de tener descanso, ya no tenemos que hacer una búsqueda lineal para el máximo sino que siempre sera el ultimo elemento del vector del día, y lo mismo para saber cual opción del ultimo día es efectivamente el mas óptimo. Tomamos medidas de la complejidad espacial que reduce nuestro algoritmo, y estimamos que la complejidad algorítmica que se reduce es similar, ya que al tener menos posibilidades en la matriz tenemos menos posibilidades para calcular a futuro. A continuación están las métricas de tests que realizamos del algoritmo con y sin la optimización. A medida que crecen los datos vemos como la optimización toma un rol mas importante.

```
=== Calcular operaciones ahorradas en los archivos de la catedra ===
Archivo: tests/cronograma_3.txt, optimizado: 4, sin optimizar: 6, diferencia: 33.33333333333333%
Archivo: tests/cronograma_9.txt, optimizado: 25, sin optimizar: 45, diferencia: 44.44444444444444%
Archivo: tests/cronograma_27.txt, optimizado: 176, sin optimizar: 378, diferencia: 53.43915343915344%
Archivo: tests/cronograma_81.txt, optimizado: 787, sin optimizar: 3321, diferencia: 76.30231857874135%
Archivo: tests/cronograma_243.txt, optimizado: 4777, sin optimizar: 29646, diferencia: 83.88652769344937%
Archivo: tests/cronograma_729.txt, optimizado: 25870, sin optimizar: 266085, diferencia: 90.27754289042976%
Archivo: tests/cronograma_2187.txt, optimizado: 148477, sin optimizar: 2392578, diferencia: 93.79426710435355%
Archivo: tests/cronograma_6561.txt, optimizado: 671192, sin optimizar: 21526641, diferencia: 96.88204025885878%
Archivo: tests/cronograma_19683.txt, optimizado: 4003983, sin optimizar: 193720086, diferencia: 97.93310901173149%
```

3.3. Variabilidad de los valores de S y E

Si tenemos en cuenta el algoritmo original, la variabilidad de los valores no altera el tiempo de ejecución ya que siempre se tiene que llenar toda la matriz diagonal inferior. En cuanto al algoritmo optimizado, si podemos encontrar un mayor impacto. Podemos notar que a medida que los datos van resultando en que pasen muchos días acumulados de entrenamiento, nuestra matriz va creciendo y nuestra optimización deja de ser tan útil. Esto sucede cuando la cantidad de energía disponible luego de varios días acumulados sigue siendo alta, por lo que nunca podemos podar la cantidad de posibilidades. Realizamos una prueba manteniendo los mismos valores que en la figura anterior, solo que el máximo valor del vector S en la simulación era mas elevado y estos fueron los resultados:

```
=== Calcular operaciones ahorradas en los archivos de la catedra ===
Archivo: tests/cronograma_3.txt, optimizado: 6, sin optimizar: 6, diferencia: 0.0%
Archivo: tests/cronograma_9.txt, optimizado: 44, sin optimizar: 45, diferencia: 2.2222222222222223%
Archivo: tests/cronograma_27.txt, optimizado: 346, sin optimizar: 378, diferencia: 8.465608465608465%
Archivo: tests/cronograma_81.txt, optimizado: 3131, sin optimizar: 3321, diferencia: 5.721168322794339%
Archivo: tests/cronograma_243.txt, optimizado: 27378, sin optimizar: 29646, diferencia: 7.650273224043716%
Archivo: tests/cronograma_729.txt, optimizado: 221315, sin optimizar: 266085, diferencia: 16.825450513933518%
Archivo: tests/cronograma_2187.txt, optimizado: 2051786, sin optimizar: 2392578, diferencia: 14.243715356406353%
Archivo: tests/cronograma_6561.txt, optimizado: 18840816, sin optimizar: 21526641, diferencia: 12.476749159332384%
Archivo: tests/cronograma_19683.txt, optimizado: 162564150, sin optimizar: 193720086, diferencia: 16.082966223750283%
```

Como era de esperarse, ahora las optimizaciones no pudieron explotarse tanto como lo hacían antes, donde los datos de E y S se distribuían uniformemente en el mismo intervalo.

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.

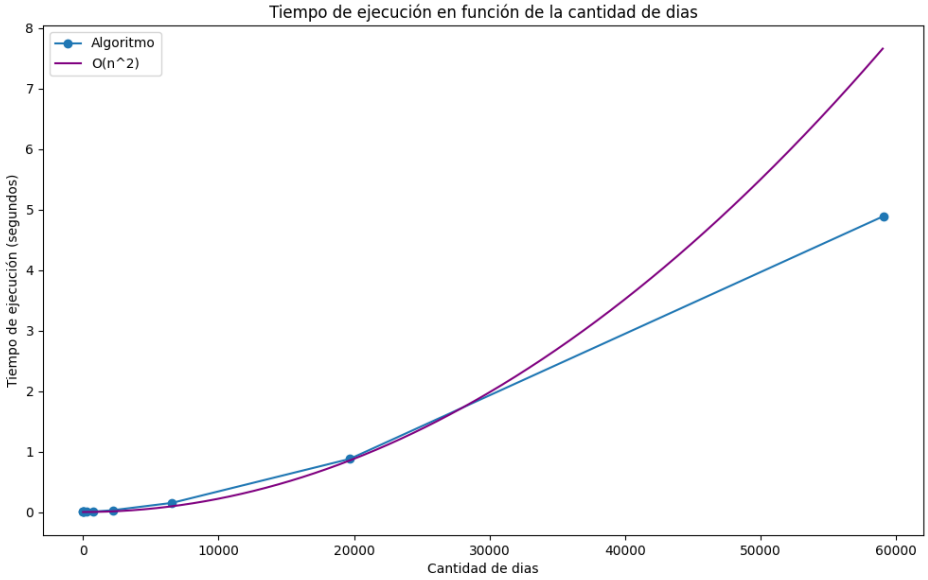
Una vez verificada la optimalidad, desarrollamos un generador de datos, el cual crea archivos con el mismo formato que el de la cátedra. Lo utilizamos para crear archivos con distintos tamaños y, también, tiene la opción para agregarle variabilidad a los esfuerzos o energías, según elección. Esto fue de suma utilidad para simular cómo se comporta nuestro algoritmo en la práctica con diferentes escenarios, utilizando diferentes cantidades de días y variando la diferencia los valores de las energías y esfuerzos.

Cabe recalcar que para generar las energías decidimos ordenar nuestro set para que queden de forma decreciente, ya que, si íbamos acotando el randint con la energía anterior, podía suceder el escenario donde el primer valor tenga un valor sumamente bajo, dejando el resto de días con posiblemente muchos valores repetidos.

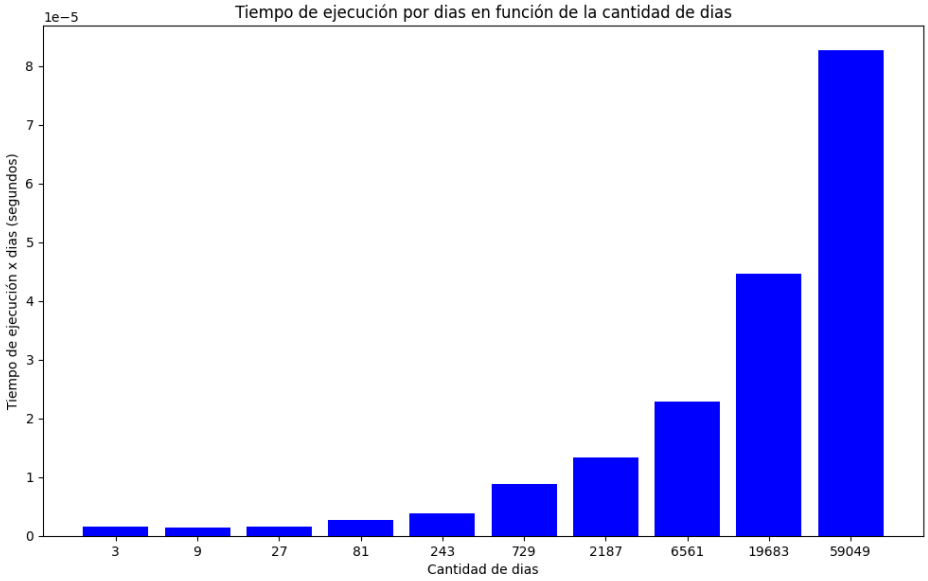
Una vez definidos los parámetros, por cada simulación se genera un archivo en el mismo formato que los provistos por la cátedra. Luego, estos archivos son pasados a nuestro algoritmo y se calcula el tiempo que tardó en ejecución para luego graficarlo.
Una vez finalizadas las simulaciones, generamos dos gráficos con sus análisis:

Este primer gráfico es un lineplot que muestra cuanto tiempo tardó el algoritmo variando la cantidad de días en el cronograma de entrenamientos. Además, agregamos una simulación de

performance de $\mathcal{O}(n^2)$ para comparar con la de nuestro algoritmo. Podemos ver que, tal como esperábamos, el algoritmo posee una curvatura levemente inferior a la de su cota de $\mathcal{O}(n^2)$.



En esta segunda imagen, podemos observar un gráfico de barras que muestra el tiempo promedio de ejecución por día, según la cantidad de días totales. Podemos observar fácilmente que, mientras mayor sea el número de días total a ver, más tiempo tarda nuestro algoritmo.



5. Conclusiones

En conclusión, utilizamos un algoritmo con enfoque bottom-up, construyendo las posibilidades desde el día inicial hasta el final para determinar la mayor ganancia posible. Esto lo logramos

llenando una matriz donde a medida que nos movemos hacia la derecha van pasando los días de entrenamiento y cuando vamos hacia arriba van aumentando los días acumulados de entrenamiento.

Este algoritmo es de programación dinámica ya que para resolver el problema total, lo dividimos en problemas más pequeños y guardamos el óptimo para aquellos subproblemas. Luego a la hora de resolver los problemas mas grandes podemos utilizar los subproblemas resueltos ya computados, ahorrándonos de repetir cálculos que ya realizamos.