



1. A - Maximise The Score

1. Reducción del Problema

- **Descripción en palabras:** Para este problema nos dan cantidad de números par en un tablero, de los cuales por cada turno (es un juego de una persona) tenemos que tomar dos números y el mínimo entre ellos dos sumarlo a nuestro puntaje que inicia en 0. El problema nos pide encontrar la forma más óptima de escoger los números para obtener el puntaje máximo en el juego. Como input nos pasarán primero el número de casos, el número de enteros en el tablero y después los números escritos en el mismo.
- **Descripción en lenguaje matemático:**
 - $A = \{a_1, a_2, \dots, a_n\}; i \in \{1, \dots, n\}$
 - $|A| = 2n$
 - $x, y \in A$
 - $score \leftarrow score + \min(x, y)$
 - Determinar: $\sum_{i=1}^{n/2} \min(x_i, y_i)$; donde este valor debe ser máximo.

2. Reducción de la Solución

- **Descripción en palabras:** Para solucionar el problema debemos organizar el arreglo de números de mayor a menor, y seleccionamos siempre el par de números más grandes posibles, para maximizar la suma de los mínimos de cada par elegido.
- **Descripción en lenguaje matemático:**
 - $A = \{a_1, a_2, \dots, a_n\}$ donde $a_1 \geq a_2 \geq \dots \geq a_n$
 - Los pares óptimos son $(a_1, a_2), (a_3, a_4), \dots, (a_{n-1}, a_n)$
 - El puntaje máximo es $\sum_{i=1}^{n/2} (a_{2i-1}, a_{2i})$

3. Código Realizado y Análisis

- **Enlace(s) Código:**
 - Este es completo con comentarios solo que el judge me dice login failed: [Python con comentarios](#)
 - Este es el original que subí sin comentarios: [Python](#)

4. Casos de Prueba

- **Casos de prueba:**

Input	Output
3 1 2 3 2 1 1 2 1 3 1 1 1 1 1 1	2 2 3

cases: 2
amount of numbers: 3
numbers: 1 2 3 4 5 6
9
amount of numbers: 0
numbers: 1

IndexError

- **Justificación:** El primero es el que nos dieron de **vjudge** y el código lo cumple correctamente, el segundo es un ejemplo propio donde en el primer caso se muestra la correcta funcionalidad del código, y en el segundo se comprueba que cuando pasamos como cantidad de números 0, nos genera error al intentar pasar un número.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- **Estrategia :** la estrategia que seguí fue siempre buscar la forma de tomar los números más grandes, para esto era evidente ordenar los números de mayor a menor, entramos a un bucle que acaba cuando el tablero queda vacío, entonces usando la función pop, siempre se asegura de sacar el primer número del arreglo ya ordenado. Después usando la función min obtenemos el mínimo de estos números y lo sumamos al score que debía estar inicializado antes del bucle en 0, cuando terminemos de sumar el bucle acaba y obtenemos el puntaje máximo.

6. Preguntas de Aprendizaje

- **Temas aplicados:** Ordenamiento de arrays.
- **Dificultad de la implementación:** Al momento de subir esto, me di cuenta de que estaba gastando mucha memoria y tiempo al ordenar el arreglo en cada iteración del bucle, cuando realmente no hacía falta. También recordar cómo se podía tomar un input de varios valores y convertirlos en lista, necesité consultar con chat gpt para realizar este input.
- **Recursos utilizados:** [ordenamiento](#).

7. Feedback LLM

- **Envío código LLM:** [CHAT GPT](#)
- **Comparación de su códigos:**

✗ COSAS QUE PODRÍAS MEJORAR:

Aspecto	Tu versión	Versión optimizada	Comentario
Orden de los números	Ordenas de mayor a menor (<code>reverse=True</code>)	Ordenamos de menor a mayor (<code>sort()</code>)	Al emparejar los menores entre sí, siempre garantizas que <code>min(x,y)</code> no se desperdicia emparejado con un mucho mayor. Esto da mejor resultado.
Eliminación de elementos	Usas <code>pop(0)</code>	No se usa <code>pop</code> , solo indexación	<code>pop(0)</code> es $O(n)$, porque mueve todos los elementos una posición. Esto es el mayor cuello de botella de tu código.
Estructura de control	Bucle <code>while</code> con <code>pop()</code>	Bucle <code>for</code> sobre rangos fijos	El uso de índices fijos es más rápido y más claro aquí, ya que sabes que recorrerás $2n$ elementos.
Complejidad	Aproximadamente $O(n^2)$ por <code>pop(0)</code>	$O(n \log n)$ solo por el ordenamiento	Tu código tiene peor eficiencia debido al uso de <code>pop(0)</code> en bucle.
Salida	Usas <code>print(score)</code> en la función	Acumulamos resultados y los imprimimos al final	Esto hace más limpio el manejo de múltiples casos de prueba.

○

- **Feedback de LLM:** Estoy de acuerdo con la corrección porque hace el código más legible y lo optimiza más, además de no usar tantas variables que se consideran innecesarias.

python

Copiar

Editar

```
# author: jpastor (versión corregida y optimizada)

def maxScore(n, arr):
    arr.sort() # Ordenar de menor a mayor → O(n Log n)
    score = 0
    for i in range(0, 2 * n, 2): # Tomar pares consecutivos
        score += arr[i] # arr[i] es el menor del par
    print(score)

def main():
    x = int(input("cases: ")) # Número de casos de prueba
    for _ in range(x):
        y = int(input("amount of numbers: ")) # n, no 2n
        arr = list(map(int, input("numbers: ").split()))
        maxScore(y, arr) # arr tiene 2n elementos

main()
```

2. B - Recovering a Small String

1. Reducción del Problema

- **Descripción en palabras:** El problema nos pide decodificar de un número entero, una palabra de 3 letras en el abecedario latino. Cada letra del abecedario toma un valor del 1 al 26 donde la letra “a” toma el valor de 1, y la letra “z” toma el valor de 26. Por lo tanto dependiendo del valor que nos pasen deberemos devolver una palabra de 3 letras del alfabeto, sin embargo, se nos dice que para evitar ambigüedades debemos de tener en cuenta que hay strings lexicográficamente más pequeños que otros, esto en base a unas reglas que nos dicen que un string “a” es menor a un string “b”, si ha se considera un prefijo de “b” y si en la primera posición de los strings “a” tiene una letra que es aparece primero en el alfabeto que la primera letra de “b”, teniendo en cuenta que ambos strings son diferentes.
- **Descripción en lenguaje matemático:**
 - Σ^3 conjunto de cadenas de longitud 3 del alfabeto : $\Sigma = \{a, b, \dots, z\}$
 - $u, v \in \Sigma^3 : u = u_1, u_2, u_3 ; v = v_1, v_2, v_3$ donde $u < v$ entonces
 - $\exists i \in \{1, 2, 3\}$ tal que $u_j = v_j$ para todo $j < i$ y $u_i < v_i$
 - dado un $n \in \mathbb{Z}$ cadena de Σ^3 correspondiente a la enésima cadena en orden lexicográfico creciente.

2. Reducción de la Solución

- **Descripción en palabras:** Para resolver este problema debemos pensar en varios casos, ya que a primera vista podemos pensar que siempre podemos usar la palabra “aa” y terminarla con una letra que cumpla con la condición, por ejemplo si nos pasan 26, ya usamos dos veces “a” entonces es 26-2, por lo tanto $24 = x$; así para números pequeños se resuelve rápido con una palabra que empiece con “aa”, de lo contrario intentaremos buscar 3 letras que sumen exactamente ese número, para eso intentamos con diferentes combinaciones, siempre priorizando que la primera letra pueda ser la más pequeña posible, que nos permita que las otras dos letras puedan completar el resto, lo ideal sería una letra que esté entre 1 y 26 que sería esta la primera letra, que permita que la suma de las otras 2 complete el valor faltante, teniendo en cuenta que la suma máxima de estas letras es 52.
- **Descripción en lenguaje matemático:**
 - $a + b + c = x ; x, a, b, c \in \mathbb{Z}$
 - $3 \leq x \leq 78$ ya que con otros números no hay solución
 - si: $3 \leq x \leq 26$ usamos $a, b = 1 ; c = x - 2$ y pasamos estos valores a letras para
 - Σ^3 conjunto de cadenas de longitud 3 del alfabeto : $\Sigma = \{a, b, \dots, z\}$
 - si $x > 26$ buscamos cualquier combinación tal que $a \leq b \leq c$.

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:

```
78
zzz
79
impossible
```

Input	copy	Output	copy
5		aav	
24		rzz	
70		aaa	
3		czz	
55		auz	
48			

- **Justificación:** Usamos los casos de prueba del vjudge para comprobar que el código funciona correctamente, después usamos el número 78 para comprobar que la palabra que podemos obtener con el número máximo es “zzz”, si usamos 79 es imposible, lo mismo que con números menores a 3

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- **Casos a tener en cuenta:** vamos a plantear 3 casos, si x es menor que 3 o mayor a 78 el problema no tiene solución, si x es menor o igual que 26, la combinación de letras menor lexicográficamente siempre será una combinación de “aa” y una letra que corresponda al valor de x-2, en el último caso donde $x > 26$, buscaremos combinaciones donde las 3 letras no puedan superar los valores de 26, utilizaremos primero un bucle for que tome valores entre $\max(1, x-26)$ y 27, esto se hace para que el máximo número sea 26 y el menor 1, $x - 26$ nos ayuda a mantener este rango, si x es menor que 26 este caso sabemos que no aplicará y en caso de que por error pase a esta condición, el bucle iniciará desde 1, de lo contrario usaremos la variable remain, que nos muestra por cada iteración la resta entre el valor x y el iterador, este iterador funciona como nuestro valor de la primera letra, con este valor pasamos a un segundo bucle, también de 1 a 27, donde el iterador tomará el valor de la segunda letra, y la tercera letra será la resta de remain y la segunda letra, con esto vamos buscando los valores que nos permitan que la suma de first, second y third, den el valor de x; finalmente tenemos un condicional que obliga a que el valor de third esté entre 1 y 26, entonces siempre existirá la forma de que:

$$first \leq second \leq third$$

- **Estrategia:** (Si la solución fue correcta en el primer intento): La estrategia que seguí fue buscar valores que siempre se mantuvieran entre 1 y 26 principalmente, luego tenía que darle prioridad a la primera letra para que fuera la menor de manera que $abc: b + c = x - a$ donde a fuera la letra más

pequeña posible, por lo tanto la suma máxima que alcanzan a tener $b+c$ es 52, entonces una vez entendemos que la primera letra debe ser la más pequeña, iteramos con esa idea, si el valor es mayor que 28 (porque acá es aaz) pero menor que 52, sabremos que la primera letra siempre será "a" la tercera será "z" y la segunda es la que irá cambiando hasta llegar a z en 52, después de esto las últimas dos letras siempre serán "zz" y la primera letra será la que irá cambiando. La lógica por lo tanto es ir iterando buscando la forma de que la primera letra intenta siempre ser menor o igual que la segunda y la segunda menor o igual que la tercera.

6. Preguntas de Aprendizaje

- **Temas aplicados:** Abecedario ascii.
- **Dificultad de la implementación:** entender las restas necesarias y la implementación de los for para cada una de las letras, además de separar los casos para no reiterar siempre, sino en valores de x donde la última palabra para estos casos base sea "aaz"
- **Recursos utilizados:** [ASCII](#)

7. Feedback LLM

- **Envío código LLM:** [CHAT GPT](#)
- **Comparación de su códigos:** El de LLM tiene los siguientes atributos mientras que el mío no. Es más directo, no tiene tantas validaciones, usa mucho `chr()` y `ord()`, apenas encuentra la primera validación la retorna.
- **Feedback de LLM:** Al pedirle que lo evalúe envía lo siguiente: Estoy de acuerdo con las correcciones porque son principalmente sobre clean code, y ayudan a hacer el código más legible y entendible para todos.

! Problemas / Oportunidades de mejora:

1. **Nombres de variables:** podrían ser más descriptivos.
2. **Complejidad del algoritmo:** Aunque es constante en cuanto al rango ($x \in [3, 78]$), los bucles podrían explicarse mejor para justificar su eficiencia.
3. **Validación del input:** asumes que siempre será un número correcto.
4. **Estilo:** usa convenciones PEP8 para Python (espaciado, nombres, etc.).
5. **Entrada/salida:** elimina los `input()` con mensajes (como `input("")`) si se espera que el programa sea probado automáticamente (por ejemplo, en plataformas de competencia).

3. C - New Year and Hurry

1. Reducción del Problema

- Descripción en palabras:
 - El problema trata sobre el tiempo que tiene Limak para resolver problemas de un concurso y llegar a una fiesta de año nuevo antes de la medianoche, cada problema tiene una dificultad progresiva, donde el tiempo necesario para cada problema se calcula como $i*5$ donde i se refiere al i ésimo-problema $(1,2,3,...,n)$, por lo tanto se nos pide calcular en base a una cierta cantidad de problemas n , y un tiempo necesario para llegar a la fiesta, cuántos problemas puede resolver Limak sin llegar tarde a la fiesta.
- Descripción en lenguaje matemático:
 - $tiempo\ total = 240\ min, tiempo\ por\ problema = i * 5$
 - $tiempo\ para\ llegar\ a\ la\ fiesta = k, tiempo\ para\ problemas = 240 - k$
 - $encontrar\ el\ número\ de\ problemas\ (m)\ tal\ que: 240 - (i * 5) \geq k\ donde\ i\ toma$
 - el índice de los problemas $(1,2,3...n)$

2. Reducción de la Solución

- Descripción en palabras:
 - Para este problema necesitamos buscar cuantos problemas podemos resolver, teniendo en cuenta que necesitamos cierto k tiempo para llegar a la fiesta, para esto vamos iterando, donde comprobamos el tiempo total y el tiempo para llegar a la fiesta, si tenemos suficiente tiempo para resolver un problema sin que nos pasemos del tiempo necesario para llegar a la fiesta a tiempo, podemos entonces resolver el primer problema. Por lo tanto debemos iterar cada vez, donde los valores van de 1 hasta n , estos valores se refieren a los índices de los problemas donde cada problema toma $i*5$ minutos, entonces de esta forma hacemos la operación de restarle al tiempo total el tiempo para resolver un problema, y si aun tenemos tiempo, usamos una variable que vaya sumando 1, por cada problema que podamos resolver, y así sucesivamente teniendo en cuenta que por cada problema el tiempo para resolver aumenta 5 minutos por cada uno.
- Descripción en lenguaje matemático:
 - para $i=1$ hasta n : {si $S + 5i \leq T$: $S \leftarrow S + 5i, c \leftarrow c + 1$
 - si no detener el ciclo

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:

1 237			
0	3 222	4 190	7 1

- Justificación: Usamos los casos de prueba del vjudge para comprobar que el código funciona correctamente, y además usamos el caso donde el tiempo que nos demoramos para llegar a la fiesta es 237, esto significa que no podemos resolver siquiera un problema ya que cada uno toma 5 minutos, por lo tanto el código nos lanza como resultado 0 problemas que podemos resolver.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Estrategia (Si la solución fue correcta en el primer intento): La estrategia que seguí fue principalmente ir comprobando con un condicional si tenía tiempo suficiente para llegar a la fiesta, el uso de una bandera es muy útil ya que la usamos para ir guardando valores necesarios o para acabar con bucles, en este caso me sirvió para ir guardando la cantidad de n problemas que podía resolver, el bucle me permitía irle restando tiempo al tiempo total hasta un límite de problemas, donde el tiempo iba aumentando de 5 en 5, y el condicional me decía si podía agregar un problema para resolver o no.

6. Preguntas de Aprendizaje

- Temas aplicados: banderas,
- Dificultad de la implementación: es un poco confuso ir alterando el tiempo base.
- Recursos utilizados: Conocía “corredores” de semestres pasados.

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de su códigos:
 - Lo más diferencial es el nombre de variables y el hecho de que chat usa variables más específicas y para mejor entendimiento.

vs DIFERENCIAS CLAVE

Elemento	Tu código (jpastor)	Mi código (ChatGPT)
Uso de tiempo	Modifica <code>t_total</code> directamente	Usa variables separadas: <code>total_time</code> , <code>used_time</code>
Legibilidad	Más corto, pero <code>t_total</code> cambia cada vez	Más legible, separa claramente el tiempo disponible
Explicación del <code>if</code>	<code>t_total >= t_travel</code>	<code>used_time + time_to_solve <= total_time</code>
Variables descriptivas	<code>flag</code> (funciona, pero no dice mucho)	<code>solved</code> (más descriptivo del propósito)

- Feedback de LLM: Pida al LLM que evalúe y corrija su código: me parece bien el hecho de hacer más legible el código pero no se si vale la pena crear otras variables para este uso, por lo menos en un problema no muy complejo no lo veo necesario, este seria el codigo corregido:

```
python                                                                    Copiar  Editar

n, t_travel = map(int, input().split())
time_available = 240 - t_travel
time_used = 0
solved = 0

for i in range(1, n + 1):
    time_used += 5 * i
    if time_used <= time_available:
        solved += 1
    else:
        break

print(solved)
```

4. D - Thor

1. Reducción del Problema

- Descripción en palabras: Thor tiene un nuevo celular, pero tiene un problema al momento de leer las notificaciones, hay varias aplicaciones que generan notificaciones, ya que el solo lee las notificaciones en orden que aparecen, no lee primero las notificaciones que no han sido leídas, generadas por las aplicaciones, por lo tanto no hay un orden específico de lectura por lo que puede terminar leyendo de nuevo varias notificaciones. El problema se basa en que Thor no conoce el número de notificaciones sin leer.
- Descripción en lenguaje matemático:
 - No tiene

2. Reducción de la Solución

- Descripción en palabras:
 - Para solucionar el problema debemos de manejar distintos tipos de estructuras de datos, la idea es asignarles un tipo de id a cada notificación, tener una estructura que nos permita guardar notificaciones leídas, y todas las notificaciones con el id y la app que las generó, también un contador que nos permita saber cuántas notificaciones faltan por leer. Entonces podremos usar distintos condicionales para cada uno de los eventos y realizar las operaciones que se necesiten para el problema, cómo agregar notificaciones, leer x notificaciones de una app, o leer x notificaciones que estén de primeras, para finalmente tener un resultado de cuantas notificaciones faltan por leer.
- Descripción en lenguaje matemático:
 - No tiene

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:

		4	6
		1	2
3	4	1	4
1	3	1	2
1	1	3	3
1	2	1	3
2	3	1	3

- Justificación: Estos son los casos de prueba que muestra el vjudge y el código por colab funciona correctamente, pero luego en tests más grandes el código tiene un problema de tiempo, se demora mucho. Ya que estoy usando múltiples

estructuras como sets, colas, arreglos, un arreglo de sets, por lo tanto estoy consumiendo mucha memoria y para problemas grandes con múltiples casos y valores las iteraciones por cada una de las estructuras tomaría demasiado tiempo.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Descripción del error: El código tiene un problema de tiempo de ejecución supongo que debido a la cantidad de arreglos y que para ejemplos muy grandes debe iterar muchas veces, aunque la lógica esté bien, se debe buscar alguna forma de optimizarlo.
- Proceso de depuración: Pase el código por chat gpt ya que no lograba encontrar el error, porque creo que la lógica es la correcta, sin embargo, el chat me dijo que había formas de optimizarlo sin usar tantas estructuras. Probando con los mismos casos de prueba y en otro lenguaje como C + +.
- Solución corregida: [C++](#)

6. Preguntas de Aprendizaje

- Temas aplicados: Estructuras de datos, Diccionarios, Sets,
- Dificultad de la implementación: Implementar un array con un diccionario o conjunto adentro y entender la lógica que esto conlleva al momento de desempaquetar los valores en estas estructuras.
- Recursos utilizados: [dictionaries,sets](#)

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de su códigos: La principal diferencia son los lenguajes en los que están hechos, LLM lo hizo en C + + mientras que yo lo realice en python.

Lenguaje	Python 3	C++ (moderno, con <code>auto</code> , <code>emplace_back</code> , etc.)
Entrada	Usa <code>input().split()</code> por línea	Usa <code>cin</code> y <code>>></code> con <code>sync_with_stdio(false)</code> para velocidad
Cola de notificaciones	<code>deque()</code>	<code>deque<pair<int,int>></code>
Control de lectura hasta cierto punto (evento 3)	Variable <code>runner</code>	Variable <code>read_pointer</code>
Verificación de existencia en set	<code>if x not in set</code>	<code>if (!set.contains(x))</code> (requiere C++20)
Lectura de evento y argumento	<code>parts = input().split()</code> → <code>event_type, x</code>	<code>cin >> event_type >> x</code>
Conversión de tipos	Automática en Python	Tipada explícitamente en C++

1. Validación de límites en evento tipo 3:

- En C++, se agrega una verificación adicional:

```
read_pointer < notif_queue.size()
```

Esto evita acceso fuera de rango.

- En Python, se asume que `runner < t` es suficiente (potencialmente más riesgoso si no se controla bien `t`).

2. Eficiencia y rendimiento:

- C++ es más rápido en tiempo de ejecución por naturaleza del lenguaje.
- Uso de `sync_with_stdio(false);` y `cin.tie(nullptr);` en C++ para mejorar velocidad de entrada/salida.

3. Organización del código:

- En Python, el input y la lógica están todos en la misma función.
- En C++, el `main()` se encarga de entrada principal y delega a la función `thor_notifs()`.

- Feedback de LLM: Pida al LLM que evalúe y corrija su código: El LLM corrigió el código y quedo en este enlace donde el vjduge lo acepto. Demostrando que sí tenía y no era tanto un problema del lenguaje, sino más bien un problema con los arreglos porque estaba iterando muchas veces innecesariamente y no accedía a algunos ítems usando el index de las listas y algunas optimizaciones en los condicionales.

Enlace: <https://vjudge.net/solution/60919623/QQdov4nNpanvWaQm5CJE>

5. E - Búsqueda Binaria

1. Reducción del Problema

- Descripción en palabras:
 - Nos dan un arreglo ordenado de manera ascendente donde el tamaño del arreglo es N y nos dan el valor que buscamos que corresponde a un número dentro del arreglo, el problema se basa en encontrar la primera ocurrencia del entero Q que nos pasan, usando búsqueda binaria.
- Descripción en lenguaje matemático:
 - $A[a_0, a_1, a_2, \dots, a_{N-1}]; a_i \leq a_{i+1} \forall 0 \leq i < N - 1; Q \in A$
 - Hallar: $\min\{i \in [0, N - 1] \mid a_i = Q\}$
 - Con complejidad: $O(\log N)$

2. Reducción de la Solución

- Descripción en palabras:
 - La búsqueda binaria es un algoritmo conocido para reducir el tiempo de búsqueda de un valor en un arreglo de valores ordenados. En este caso lo implementaremos para que nos de la primera ocurrencia de un entero en un arreglo, para esto podemos usar dos punteros uno que apunta la inicio del arreglo y el otro apunta al final del mismo, calculamos el índice medio de estos dos y comparamos, si el valor del arreglo usando el índice medio es el buscado entonces resolvimos el problema, si el valor es menor que el buscado, el valor objetivo debe de estar a la derecha, entonces cambiamos el puntero menor a el índice $medio + 1$, si el valor del arreglo usando el índice medio es mayor que el valor objetivo entonces el valor objetivo debe de estar hacia la izquierda, entonces movemos nuestro puntero más alto a el valor del índice $medio - 1$, y se repite el proceso con rangos más pequeños hasta encontrar el valor objetivo. También en caso de que encontremos el índice medio que cumple con el valor objetivo del arreglo, guardaremos ese valor de índice y moveremos el índice mayor, a un valor de medio - 1, para buscar si hay una ocurrencia más temprana que la que ya encontramos, si no devolvemos el valor encontrado anteriormente.
- Descripción en lenguaje matemático:
 - No aplica

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:

10 4	5 3	10 5
1 3 4 5 5 6 7 8 8 17	1 2 2 3 6	12 12 12 16 17 18 19 20 25
3	1	4
5	0	-1
9	2	2
1	1	-1
	3	12
	3	0
		16
		3
		17
		4

- Justificación: Use el caso de vjudge como primera prueba del código y ahí comencé a experimentar con otros dos casos usando números pequeños primero y repitiendo el 2, para ver si devolvió la primera ocurrencia y si la búsqueda binaria funcionaba bien con los demás números. El último caso use números un poco más grandes para ver si funcionaba con números de dos dígitos y aplicó también, además repetí el 12 tres veces seguidas y devolvió la primera ocurrencia.

5. Iteración en caso de Solución Incorrecta (o explicación Solución Correcta).

- Estrategia (Si la solución fue correcta en el primer intento): La solución es usar el concepto de índices, entonces usamos índices en este caso low,high; donde low es 0 por el inicio de la lista, y high es la longitud de la lista - 1, calculamos el índice medio, usando una fórmula que evita el overflow, y comenzamos a buscar el valor objetivo, cambiando los valores de low y high dependiendo de si el objetivo es más grande o más pequeño que el valor que obtenemos del arreglo buscando por el índice medio. Así iteramos hasta encontrar el valor del índice del arreglo que contiene al valor objetivo.


•

6. Preguntas de Aprendizaje

- Temas aplicados: Búsqueda binaria, Overflow
- Dificultad de la implementación: Entender cómo aplicar el overflow para que el código no sufra con números grandes y esté optimizado.
- Recursos utilizados: [binary search](#), [overflow](#)

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de sus códigos: La diferencia que dice es la cantidad de comentarios que incluyen el código que envíe.

Aspecto	Tu código (jpastor)	Mi versión 
Complejidad	Correctamente comentada	No la incluí explícitamente, pero es la misma: <code>O(q log n)</code>
Funcionalidad	Correcta y eficiente	Correcta y eficiente
Uso de variables	Manejo manual de <code>mid</code> , <code>low</code> , <code>high</code>	Igual
Estilo de código	Comentarios embebidos línea a línea. Buen detalle, pero puede ensuciar la lectura.	Más limpio, sin comentarios de complejidad repetitivos
Nombres de funciones	✅ Claro (<code>binary_search</code>)	✅ Claro (<code>binary_search_first_occurrence</code>)
Legibilidad	Puede mejorar quitando comentarios redundantes o usando docstrings y bloques	Alta legibilidad, bloques separados y nombres autoexplicativos
Escalabilidad	Escala bien	Escala bien
Entrada y salida	Correcta, directa con <code>input()</code>	Igual

- Feedback de LLM: Pida al LLM que evalúe y corrija su código: Nuevamente solo me dice sobre los comentarios realizados y el hecho de haber declarado `mid = 0` ya que no es necesario y estoy de acuerdo aunque es un detalle muy pequeño.

6. F - Búsqueda Binaria 2

1. Reducción del Problema

- Descripción en palabras:
 - El problema nos pide aplicar la búsqueda binaria para encontrar en un arreglo la última iteración de un entero dado, nos dan el tamaño del arreglo N y el valor objetivo a a buscar dentro del mismo.
- Descripción en lenguaje matemático:
 - $A[a_0, a_1, a_2, \dots, a_{N-1}]; a_i \leq a_{i+1} \forall 0 \leq i < N - 1; Q \in A$
 - Hallar: $\max\{i \in [0, N - 1] \mid a_i = Q\}$
 - Con complejidad: $O(\log N)$

2. Reducción de la Solución

- Descripción en palabras: En este caso aplicaremos la búsqueda binaria del ejercicio anterior, usaremos los 3 índices, low, mid, high sin embargo, cambiaremos una condición, y es que cuando el arreglo con índice mid coincida con el valor objetivo, moveremos el rango a el valor del índice + 1, para seguir buscando si hay otra ocurrencia, guardando el valor del índice que ya nos sirvió y retornando en caso de que no encontremos otra ocurrencia y así hasta hallar la última ocurrencia del valor objetivo.
- Descripción en lenguaje matemático:
 - no aplica

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:

10 4	5 3	10 5
1 3 4 5 5 6 7 8 8 17	1 2 2 3 6	12 12 12 16 17 18 19 20 25
3	1	4
5	0	-1
9	2	2
1	2	-1
	3	12
	3	2
		16
		3
		17
		4

- Justificación: Use los mismos casos del ejercicio anterior y en este caso devuelve la última iteración de los valores objetivo, como en el tercer caso que nos devuelve el índice 2 cuando buscamos el número 12, este en el arreglo tiene índices 0,1,2 siendo 2 la última ocurrencia este es el valor que nos retorna el código.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)


- Estrategia (Si la solución fue correcta en el primer intento): Para solucionar este problema seguí la misma estrategia que se aplica para la búsqueda binaria normal, el manejo de 3 índices e ir recortando la lista, sin embargo, en esta era importante seguir buscando ocurrencias, para no quedarnos con la primera sino con la última. Para esto no terminamos al código apenas encuentre la primera ocurrencia sino que guardamos ese índice en una variable por si no encontramos otra ocurrencia, y seguimos buscando, moviendo los índices al valor del índice + 1, así si encontrábamos otra ocurrencia seguimos el proceso de guardarla en una variable el índice correspondiente, si no lo devolvemos y terminamos el problema.

6. Preguntas de Aprendizaje

- Temas aplicados: Búsqueda binaria, Overflow
- Dificultad de la implementación: Entender cómo aplicar el overflow para que el código no sufra con números grandes y esté optimizado.
- Recursos utilizados: [binary search](#), [overflow](#)

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de su códigos:

Aspecto	Tu código	Código sugerido anteriormente	
Estilo de código	Menos separación visual, algunas líneas un poco más compactas	Estilo más limpio, más legible, funciones bien separadas	
Anotaciones	Incluyes comentarios con complejidad de cada línea	No se incluyen, pero el código es autoexplicativo	
Formato de entrada	<code>input("")</code> innecesario (puede ser solo <code>input()</code>)	Uso estándar de <code>input()</code>	
Buena práctica	<code>mid = 0</code> inicial innecesario; ya se redefine antes de su uso	Se define <code>mid</code> directamente dentro del <code>while</code>	
Reutilización y claridad	Función separada <code>main()</code> para claridad y modularidad	Todo en línea principal	

- Feedback de LLM: Lo evaluó y solo quito los comentarios y quito la inicialización de la variable mid en el código con lo cual estoy de acuerdo aunque el inicializar una variable tan simple no afecta el rendimiento.

7. G - Factorial vs Power

1. Reducción del Problema

- Descripción en palabras:
 - Nos dan dos secuencias de números dadas por las funciones $f(n) = n!$; $g(n) = a^n$ siendo n un entero positivo y $a > 1$ la secuencia formada por $g(n)$ es más grande que la otra para un número finito de números, pero para un k ; $f(n)$ es mayor que $g(n)$ siempre que $n \geq k$. El problema reside en encontrar el menor valor positivo de n para que $f(n) > g(n)$.
- Descripción en lenguaje matemático:
 - $n > 0$; $n \in \mathbb{Z}$ encontrar $n : f(n) > g(n)$ donde
 - $a > 1$: $g(n) = a^n$; $f(n) = n!$

2. Reducción de la Solución

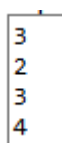
- Descripción en palabras: Para solucionar el problemas debemos comparar la función $f(n)$ con $g(n)$ de forma que dado un número a hallemos el n que cumple con esa condición. Para ello podemos comparar las funciones con valores de n y a iterando hasta encontrar el valor que cumpla con la condición.
- Descripción en lenguaje matemático:
 - no aplica

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:



```
3
2
3
4
```

- Justificación: use este como único caso de prueba, que viene de vjudge para probar si el código funcionaba bien con este primer caso y si existían errores comenzar a depurar, ya que considero que no hacía falta usar otros porque tenía dudas sobre si mi código funciona correctamente.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Descripción del error: El error nace principalmente en cómo está diseñado el código, el tiempo de ejecución es muy lento debido a que se itera N veces, y para números muy grandes ejecutar la función factorial con n muy grande, por lo tanto el código se demora demasiado tiempo. Además ya no solo la iteración está mal, sino que la lógica para optimizarlo más y encontrar una respuesta

mejor, es usar \log_{10} para reducir los valores y que sean más fáciles de procesar. No considere en qué números muy grandes harían un desbordamiento de memoria y tiempo, ya que la función factorial y de potencias con números muy grandes ni siquiera un computador con mucha potencia podría ejecutarlo con el código que realice

- Proceso de depuración: Usando un **código de codeforces** para guiarme encontré la solución. El código generaba primer los valores factoriales pero usando \log_{10} para obtener valores más pequeños, los almacenaba en una lista de tamaño 10^6 , de esta manera usando la búsqueda binaria ya podía comparar los valores de manera mas rapida y devolver el valor de n que cumpliera con la condición $f(n) > g(n)$, lo que hacía era generar la lista de factoriales usando \log_{10} , usaba la búsqueda binaria para devolver el índice mid en una variable n si en algún momento del bucle se encontraba el valor que cumplía con la condición del problema, guardaba el valor del índice mid en la variable n , pero seguía “recortando” la lista, dejando el “final” de la lista recortada en el valor encontrado - 1, es decir en la posición anterior al valor encontrado, de esta manera el bucle seguía buscando a ver si existía un valor n más pequeño que siguiera cumpliendo con la condición, y si no lo encontraba el bucle acababa porque el índice low ya llegaría a ser mayor que high y acabaría rápidamente.
- Solución corregida: Esta es la solución del código que encontré que subí a vjudge, el código es de una prueba que se hizo en codeforces que encontré como la mejor solución al problema. [C + + de otro autor](#); realice la misma lógica en python sin embargo sigue lanzando error en time exceeded [Python](#).

6. Preguntas de Aprendizaje

- Temas aplicados: Búsqueda Binaria, Iteraciones, Uso del logaritmo.
- Dificultad de la implementación: Lo más complicado fue como aplicar el uso del logaritmo, había tenido experiencias antes de optimizar la comparación de valores usando logaritmos pero en este ejercicio no pensé de primera mano que fuera necesario. También crear una lista acumulativa usando factorial con logaritmos, ya que pensaba que sería mucho menos óptimo debido a que la lista de valores a crear es demasiado grande. Sin embargo, el código de “*Rajon Bardhan*” aplica esto y en un tiempo bastante bueno.
- Recursos utilizados: [binary search](#), [overflow](#), [logaritmos](#), [codigo de codeforces](#)

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de su códigos:

Aspecto	Python	C++
Lenguaje	Python 3	C++17
Precomputación de $\log(n!)$	<code>array('d')</code> con 500,001 posiciones, desde $i = 2$ a $N = 500000$	<code>vector<double></code> con tamaño $N = \lceil e \cdot \max a \rceil + 5$
Tipo de log	<code>math.log10(i)</code>	<code>log(i)</code> (logaritmo natural base e)
Comparación dentro del binario	<code>log10(n!) > n * log10(a)</code>	<code>log(n!) > n * log(a)</code>
Tamaño del arreglo	Fijo: 500,000	Dinámico según máximo a en input
Tiempo de precomputación	Rápido (~30–70 ms)	Rápido, pero depende de $\max a$ (~50–120 ms)
E/S (entrada/salida)	<code>input()</code> y <code>print()</code> (más lento en Python puro)	<code>cin.tie(0); ios::sync_with_stdio(false);</code> → muy rápido
Velocidad de ejecución total	300–400 ms (dependiendo del entorno)	60–120 ms en general (C++ más rápido)
Complejidad tiempo total	$O(N + t \cdot \log N)$	$O(N + t \cdot \log N)$
Complejidad memoria	$O(N)$, donde $N = 500,000$	$O(N)$, donde $N = e \cdot \max a$

- Feedback de LLM:** Pida al LLM que evalúe y corrija su código: El código que mande después con la nueva lógica implementada, el LLM hace el código de la imagen. Con lo cual no sigo convencido que generar una lista tan grande con valores pueda ser demasiado óptimo, aunque al final el vjudge lo acepte no me termina de convencer este tipo de solución, aunque usando búsqueda binaria pueda resultar óptimo en tiempo y búsqueda en memoria, generar una lista tan grande de primera mano no me parece bueno. Por lo demás estoy de acuerdo con las otras correcciones

```

import math
import sys
from array import array

input = sys.stdin.readline # Entrada rápida

def binary_search(arr, a):
    loga = math.log10(a)
    low, high = 1, len(arr) - 1
    result = -1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] > mid * loga:
            result = mid
            high = mid - 1
        else:
            low = mid + 1
    return result

def main():
    N = 500_000
    facts = array('d', (0.0 for _ in range(N + 1))) # Inicializa correctamente con ceros
    for i in range(2, N + 1):
        facts[i] = facts[i - 1] + math.log10(i)

    t = int(input())
    results = []
    for _ in range(t):
        a = int(input())
        results.append(str(binary_search(facts, a)))

    sys.stdout.write('\n'.join(results) + '\n') # Salida rápida

```

8. H - Frequent Prime Ranges

1. Reducción del Problema

- **Descripción en palabras:**
 - Dado un número N y un número K , se nos pide contar cuantos subrangos consecutivos existen dentro de un intervalo $[2..N]$, que contiene al menos K números primos. Por lo tanto debemos verificar cuantos K primos hay dentro de un intervalo y contar todos los subrangos posibles.
- **Descripción en lenguaje matemático:**
 - Sea $S = \{2, 3, \dots, N\}$ tenemos un subrango $[L, R] \subseteq S$ como K frecuente primo si:
 - $\{p \in P | L \leq p \leq R\} \geq K$
 - Calcular # de intervalos tal que $2 \leq L \leq R \leq N$

2. Reducción de la Solución

- **Descripción en palabras:**
 - Para este problema usaremos una técnica de punteros, ya que para no tener que iterar siempre que queramos ver si un subarreglo contiene k primos o no, podemos usar punteros que siempre mantengan una distancia que permita que exista k números primos entre ellos. Para ello podemos usar la criba de eratóstenes para generar una lista que nos diga qué números son primos y cuáles no. Con esto crearemos dos punteros para ir moviéndose a través de la lista, el puntero 2 será nuestro puntero "inicial" que se mantendrá al inicio de la lista, el puntero 1 irá avanzando en un bucle que buscará números primos mientras la recorre, cuando encontremos k primos, guardaremos el valor del puntero, calcularemos el total de subarreglos usando los punteros y moveremos el puntero 2, así hasta encontrar todos los subarreglos posibles.
- **Descripción en lenguaje matemático:**
 - no aplica

3. Código Realizado y Análisis

- Enlace(s) Código: [python](#), [java](#)

4. Casos de Prueba

- Casos de prueba:

				4
	3			2 1
	5 2			5 2
1	10 3			5 1
2 1	100 5			9 3

- Justificación: Estos casos de prueba me sirven para ver el funcionamiento del código en varios lenguajes principalmente, el tercer caso de prueba me dice si funciona bien ya que es el del vjudge, el primero me indica lo mismo, si las cuentas se están haciendo correctamente, el segundo es más para explorar el tiempo de demora del código.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Descripción del error: En el caso de python el error era el tiempo de ejecución, tengo el error de tiempo excedente entonces intente con java, a los primeros intentos no comprendía el error ya que seguía la misma lógica y consideraba que el código estaba bien. Entonces tuve que preguntarle a la IA cual podría ser la causa del error y era el tipo de dato que usaba en algunas variables, ya que el vjudge en algunos casos de prueba si usa números muy grandes, en java el dato tipo *int* podría fallar por desbordamiento de números, entonces es mejor usar *long* con este cambio el código se ejecuta correctamente y el vjudge lo acepto, tambien hice el intento traduciendo el código C++ y ejecuto mucho mas rapido y tambien correctamente.
- Proceso de depuración: El proceso de depuración use varios casos entregados por la IA para probar el código y haciendo los cálculos y pruebas manuales los subarreglos veía que la lógica código estaba bien, Sin embargo, desconocía que el desbordamiento me cambiaría los valores de los números y por eso en primeros intentos mandaba Wrong Answer.
- Solución corregida: [C++](#). mando esta solución ya que es la mas optima pero sigue la misma lógica que las que implemente en java y python.
- Estrategia (SOLUCIÓN INCORRECTA POR TIEMPO NO LÓGICA): Aunque en los primeros intentos el vjudge no aceptaba la solución, la lógica estaba bien implementada, usar dos punteros que recorran la lista hasta que uno de ellos llegue al punto donde la lista que exista entre los dos punteros tenga al menos k primos, funciona correctamente, cuando encontramos todos los subrangos de esta forma movemos el puntero que se mantenía como inicio al siguiente número y volvemos a realizar el proceso solo que como el otro puntero ya está avanzado, solo tenemos que buscar el siguiente valor que nos permita que existan de nuevo k primos, de esta forma si llega un momento en que el puntero que señala los límites de las listas es igual al número que nos pasan del rango de la lista, el bucle solamente avanza hasta llegar al final.

6. Preguntas de Aprendizaje

- Temas aplicados: punteros, iteraciones, tipos de datos, criba de eratóstenes.
- Dificultad de la implementación: Comprender como hacer el bucle para no restar ni sumar más subrangos de los que había, puesto que en el código al calcular el total si no le sumamos 1, sería como si no estuviéramos contando el subrango en el que terminamos, también usamos un tipo de suma gaussiana en caso de que k sea 0 o sea no hay restricciones y podemos calcular el total de sub arreglos usando n ya que no hay necesidad de tener números primos.

- Recursos utilizados: [punteros](#), [tipos de datos de java](#), [criba de eratóstenes](#).

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#).
- Comparación de su códigos: El código que hizo chat en C++ encuentra todos los primos usando la criba de eratóstenes, luego calcula cuántos primos hay desde 2 hasta cada número, un conteo acumulado. Para cada número R desde 2 hasta N, busca el menor inicio L del subrango tal que haya al menos K primos entre L y R y entonces usa un tipo de búsqueda binaria para encontrar cuántos rangos que comienzan con L son válidos y suma todos los casos que cumplan.

Aspecto	⚙️ C++ (Prefijo + Binary Search)	🚀 Java (Sliding Window)	📄
Complejidad por caso	$O(N \log N)$	$O(N)$	
Preprocesamiento	$O(N \log \log N)$	$O(N \log \log N)$	
Memoria	$O(N)$ (2 arrays)	$O(N)$ (1 array)	
Velocidad práctica	Buena	Más rápida	
Claridad del código	Modular y limpio	Requiere cuidado en ventana	
Manejo $K = 0$	Incluido en lógica general	Requiere caso especial	

- Feedback de LLM: Pida al LLM que evalúe y corrija su código: Estas mejoras son las que propone en LLM, Realiza un cambio sustancial en la función de contar rangos para evitar errores si se salen los índices del valor máximo del array.

Mejora	Justificación	📄
<code>pt1 → left</code> , <code>pt2 → right</code>	Mejora la legibilidad	
Validación <code>left <= right</code>	Evita errores de límite o <code>ArrayIndexOutOfBoundsException</code>	
Condición clara en <code>while</code>	Asegura que se reduzca <code>primeCount</code> cuando se mueve	
Comentarios útiles	Explican lógica sin ruido	

```
// Cuenta subrangos [L..R] en [2..n] con al menos k primos
public static long countKFrequentRanges(int n, int k, boolean[] primes) {
    long total = 0;
    int left = 2, primeCount = 0;

    for (int right = 2; right <= n; right++) {
        if (primes[right]) primeCount++;

        while (left <= right && primeCount >= k) {
            total += (n - right + 1); // Todos los subrangos [left..right..n] son válidos
            if (primes[left]) primeCount--;
            left++;
        }
    }

    return total;
}
```

9. I - Sudoku Solver

1. Reducción del Problema

- Descripción en palabras:
 - El problema nos da un sudoku con algunos números dentro, la tarea es resolver el sudoku el cual tiene una única solución.
- Descripción en lenguaje matemático: no aplica

2. Reducción de la Solución

- Descripción en palabras:
 - Para resolver el sudoku haremos uso de la “fuerza bruta” y de backtracking de la recursión, que nos permite ir probando diferentes soluciones y si la solución no funciona regresa a un último estado donde el sudoku iba funcionando correctamente y cambia el valor del número en la celda que se encontraba en ese momento. Es decir vamos iterando celda por celda teniendo en cuenta los números vecinos en las filas, columnas y cuadros 3x3, y vamos probando numero por numero hasta que resolvamos el sudoku por fuerza bruta.
- Descripción en lenguaje matemático: no aplica.

3. Código Realizado y Análisis

- Enlace(s) Código: [python](#)

4. Casos de Prueba

- Casos de prueba:

1	1	1 3 5 6 7 2 8 9 4
.46...9..	1.5..2.9.	7 2 8 5 4 9 1 6 3
.3.1.....	.2.549.63	9 4 6 8 3 1 5 2 7
.2..6...85	.4..3..2.	2 6 9 4 1 5 7 3 8
...87....	.6..1..3.	4 7 3 9 2 8 6 5 1
6...3...4	4..928..1	5 8 1 7 6 3 2 4 9
....14...	5.1.6...9	6 9 4 1 5 7 3 8 2
79..5..3.	.9.....	8 1 2 3 9 6 4 7 5
....2.4.	8..39.475	3 5 7 2 8 4 9 1 6
..2...61.	35728.9..	

- Justificación: Use la primera imagen que es el caso de prueba del vjudge para probar primero si el código funciona correctamente, y ya después usando una página de sudokus, pase el sudoku a los strings de la segunda imagen representando los vacíos como puntos y la tercera imagen es el sudoku solucionado para comprobar el funcionamiento del segundo caso.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Descripción del error: El error se encuentre principalmente en el tiempo de ejecución del código, Al momento de diseñar el código ya notaba la complejidad que tendría debido a que uso 3 bucles for en el código para hallar la solución con fuerza bruta, además de que hacía uso del backtracking, por lo tanto si por fuerza bruta una solución fallaba, toca volver al estado anterior que guardó la recursión, y intentar de nuevo con otro numero asi hasta hallar la solución, por lo tanto en el peor de los casos la complejidad en tiempo llegaba a ser de hasta $O(9^k)$ siendo k la cantidad de celdas vacías en el tablero.
- Proceso de depuración: El proceso de depuración lo hice con los casos de prueba y el código ejecutaba en un tiempo promedio de 40 a 60 ms sin problemas, sin embargo no use casos donde se nos pasará más de un tablero con mayor complejidad, es decir con menos pistas iniciales. Por lo tanto ya solo viendo que el tiempo límite era excedido y viendo la complejidad de mi código era fácil ver el problema que tenía era de tiempo y no de respuesta. Por lo tanto, optimizar el código era la principal prioridad. Investigando existe algo llamado Heurística MRV siendo una técnica que prioriza la selección de variables con el menor número de valores posibles que aún pueden ser asignados. Esto significa que la nueva función que añade esto(find best cell), busca entre todas las celdas vacías, elige la que tiene menos opciones válidas posibles, esto permite que se reduzca la rama del árbol de búsqueda, porque las celdas más restringidas tienden a fallar antes si hay errores y entonces el proceso del backtracking sera mas rapido porque estaremos iniciando desde un punto donde habrá menos posibilidad de errores gracias a la heurística MRV, esto se aplica en el nuevo código realizado en C++ para mayor optimización, porque en python seguía habiendo problemas con el tiempo.

- Solución corregida: [python](#), [C++](#)

6. Preguntas de Aprendizaje

- Temas aplicados: backtracking, Heurística MRV, recursividad, cálculo de vecinos.
- Dificultad de la implementación: Enfrenta dos dificultades, el cálculo de los vecinos no pensaba bien como calcular los 9 sectores diferentes de 3x3, de manera que el código validará los números que ya estaban en esas secciones sin confundirlas, la solvente viendo ejemplos en foros e internet usando el módulo para no salirse del rango y validar siempre los números en estos espacios. La otra dificultad fue la implementación de la Heurística MRV ya que era un tema nuevo y no comprendía cómo funcionaba correctamente, para ello me tomé la tarea de realizar unos sudokus a mano y veía como inconscientemente, siempre buscaba resolver los cuadros que tenían mayor cantidad de numeros y así, y es lo mismo que realizaba la Heurística, buscar las celdas cuyas posibilidades fueran mínimas y empezar a resolver desde ahí para tener menos errores.
- Recursos utilizados: [sudoku](#), [backtracking](#), [Heurística MRV](#)

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de su códigos:

Aspecto	Tu Versión (Python)	Versión Optimizada (C++)	
Lenguaje	Python	C++	
Representación de tablero	Lista de listas de enteros (<code>board[row][col]</code>)	Arreglo <code>char[9][9]</code>	
Celda vacía	<code>0</code>	<code>'.'</code>	
Selección de celda	Busca la celda vacía con menos opciones (<code>find_best_cell</code>)	Igual, pero con cálculo más rápido usando <code>bitmask</code>	
Validación de número	Verifica fila, columna y subcuadro con bucles	Verifica con operaciones de bits (<code>0(1)</code>) en vez de <code>0(9)</code>)	
Estructura de datos para dominio	Ninguna (solo recalcula al vuelo cada vez)	<code>bitmask</code> : guarda en <code>uint16_t</code> qué números están disponibles	
Velocidad estimada	~300-1800 ms con 30 casos	< 200 ms incluso con 30 tableros difíciles (en máquina moderna)	
Complejidad en peor caso	<code>0(9^k)</code> donde <code>k</code> = celdas vacías (explícito)	Igual, pero constante multiplicativa 10x más rápida por uso de bits	
Estrategia de backtracking	Recursiva con heurística de mínimo dominio	Igual, pero implementada con swap in-place y sin recalcular dominios completos	

E/S (entrada/salida)	<code>input()</code> , <code>print()</code>	<code>cin/cout</code> con <code>ios::sync_with_stdio(false);</code> y <code>cin.tie(nullptr)</code>
Memoria extra usada	Muy poca, solo el stack de recursión	Igual, pero más compacta por uso de <code>uint16_t</code>
Compilación necesaria	No aplica (interpretable)	Requiere <code>g++ -O3 -march=native</code> para máximo rendimiento
Facilidad para debugging	Alta (por ser Python)	Media, requiere recompilar y revisar trazas
Aptitud para competencia	Buena para problemas pequeños	Ideal para problemas con límite de tiempo muy ajustado

- Feedback de LLM: Pida al LLM que evalúe y corrija su código: Le pido a chat que evalúe el código, directamente me recomienda pasar mi código a C++ para hacer uso de los bitmarks tipo C++ y me devolvió el mismo código del envío que realizo [CHAT GPT](#). Aunque no comprendo el uso de los bitmask y uso de bits en C++, es mucho mejor que la solución que implemente que trabaje por fuerza bruta sin tomar en cuenta casos que podrían no tener solución.

10. J - Chessboard and Queens

1. Reducción del Problema

- Descripción en palabras:
 - El problema nos entrega un tablero de ajedrez con ciertas posiciones reservadas, denotadas con un '*' y los demás espacios libres se representan con un '.'; Se nos pide hallar el número total de formas en que podemos posicionar 8 reinas en el tablero de forma que las reinas no se ataquen entre sí, teniendo en cuenta que solo podemos posicionar las reinas en los espacios libres y no los reservados.
- Descripción en lenguaje matemático:
 - No aplica

2. Reducción de la Solución

- Descripción en palabras:
 - Para solucionar este problema debemos comprobar que en cada fila que se ponga una reina, la columna en la que se encuentre no se interponga entre otras reinas, es decir no quede atacando a otra reina. Para esto podemos iterar múltiples veces posicionando las reinas en

diferentes posiciones de manera que vayamos encontrando diferentes soluciones y sumando 1 por cada solución encontrada. Esta solución encontrada debe cumplir con que ninguna reina ataque a las otras de manera que debemos comprobar que las diagonales y columnas de una reina no se vean interceptadas por otra.

- Descripción en lenguaje matemático:
 - No aplica.

3. Código Realizado y Análisis

- Enlace(s) Código: [Python](#)

4. Casos de Prueba

- Casos de prueba:

```

.....
*.....
.....
.....
.....
**.....
*.....
.....
.....

```

- Justificación: Use dos casos de prueba, el caso base de vjudge para probar si el código entrega la respuesta correcta con este caso base, y el segundo caso es para comprobar que el código si está leyendo bien el problema, ya que si no hay restricciones de casillas reservadas, entonces por la teoría de este problema, sabemos que el máximo de soluciones es 92.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Estrategia (Si la solución fue correcta en el primer intento): La estrategia que seguí fue parecida al del sudoku, diseñe dos funciones principales. La primera es una función que me valida si una casilla ya se ve afectada por una reina, es decir si al poner una reina x en esa casilla, otra reina la atacara sea por columna o alguna de las diagonales, El ejercicio nos especifica poner 8 reinas por lo tanto no hay que hacer comprobaciones en las filas ya que solo puede haber 1 por fila. Para resolver el tablero debe iterar por las casillas, siguiendo un método parecido al de la fuerza bruta, si una casilla no está ocupada, posicionar una reina, vuelvo a llamar la función pero cambiando de fila, para entonces buscar una nueva casilla en las siguientes filas, si llegamos a una casilla que se encuentra bloqueada o reservada, simplemente volvemos al bucle e incrementamos el valor de las columnas. Si al final de iterar 8 veces las columnas, no llegamos al valor del contador que sea igual a 8, significa que tuvimos algún error y no completamos el ejercicio por lo que no pudimos posicionar las 8 reinas en el tablero, entonces hacemos backtracking en donde posicionamos la primera reina y volvemos a iniciar pero cambiando la posición a la siguiente y así sucesivamente, hasta que una solución nos funcione, la agregamos a un contador y así sucesivamente hasta probar todas las

soluciones posibles y encontrar el total.

6. Preguntas de Aprendizaje

- Temas aplicados: Backtracking, cálculo de diagonales,
- Dificultad de la implementación: Se me dificultó la parte de validar las casillas, ya que no entendía completamente como hacer que el código me verificará ambas diagonales, puesto que no comprendía como se hacen las sumas y restas iterando para buscar cada diagonal dependiendo de la casilla que le pasara a la función.
- Recursos utilizados: [backtracking](#), [problema de las 8 reinas](#)


7. Feedback LLM

- Envío código LLM: [CHAT GPT](#).
- **Comparación de su códigos:** El código de LLDM es más eficiente ya que no hace tan mal uso del backtracking, puesto que no accede a soluciones falsas, debido a que usa Sets, además de que no usa números como yo para representar diferentes estados de las casillas.

Aspecto	Primer Código (con sets)	Segundo Código (con is_valid)
Algoritmo base	Backtracking optimizado con <code>set()</code>	Backtracking clásico con verificación directa
Chequeo de conflictos	$O(1)$ con <code>set()</code> para columnas y diagonales	$O(\text{row})$ por fila en <code>is_valid</code>
Representación del tablero	Lista de strings (<code>board[row][col]</code>)	Matriz 8x8 con valores <code>0</code> (libre), <code>-1</code> (bloqueado), <code>1</code> (reina)
Estructuras auxiliares	<code>used_columns</code> , <code>used_diagonals</code> , <code>used_antidiagonals</code>	Solo el tablero (<code>board</code>)
Manejo de casillas bloqueadas	Se salta si <code>board[row][col] == '*'</code>	Se representa como <code>-1</code> en la matriz
Backtracking	Muy eficiente: no entra a ramas inválidas gracias a los sets	Eficiencia menor: prueba columnas y verifica luego
Legibilidad / modularidad	Alta: separación clara de lógica	Alta también, usa funciones auxiliares como <code>is_valid</code>
Complejidad temporal	$O(8!) \approx 40320$ (en el peor caso sin restricciones)	$O(8! \times 8)$ en peor caso por verificación $O(\text{row})$

- **Feedback de LLM:** Pida al LLM que evalúe y corrija su código: Estoy de acuerdo con no ensuciar tanto el código con complejidades, aunque se entiende bien qué complejidad se aplica en las líneas y bloques de código, pero considero que las condiciones si están claras.

✖ Aspectos a corregir o mejorar:

Problema	Detalle	Mejora sugerida	
Ambigüedad de valores en <code>board</code>	Usas <code>0</code> , <code>-1</code> , y <code>1</code> , pero no está documentado ni es explícito	Usa constantes o comentarios claros	
Nombres de variables	<code>board</code> se usa como entrada y estado, pero sería más claro <code>state</code> para el estado	Mejora semántica	
<code>is_valid()</code> puede ser más legible	Verificación de diagonales es correcta pero algo densa	Clarificar condiciones	
Complejidad	Sigue siendo $O(8! \times 8)$ por verificación $O(\text{row})$	Se puede mejorar con sets (opcional si mantienes estilo clásico)	
Comentarios de complejidad están desalineados	Algunos no tienen sentido o están mal ubicados	Elimínalos o corrégelos	

11. K - Missing LDAP

1. Reducción del Problema

- **Descripción en palabras:** El problema busca que dado el nombre completo de un estudiante, se le otorgue un usuario único que no haya sido usado por otros estudiantes, estos usuarios o LDAP se forma usando la primera letra del primer nombre, la primera letra del segundo nombre (si hay), el primer apellido completo, y la primera letra del segundo apellido, si este LDAP ya ha sido usado, entonces se prueban otros más largos usando más letras del primer nombre, seguida de la inicial del segundo nombre, el primer apellido completo y se van agregando letras del segundo apellido. El problema es hacer un sistema interactivo donde usando máximo 20 preguntas, encontremos el primer LDAP que esté disponible.
- **Descripción en lenguaje matemático:** No aplica

2. Reducción de la Solución

- **Descripción en palabras:** Para resolver el problema primero hay que generar todo los posibles LDAP dado un nombre completo de un estudiante, teniendo en cuenta las reglas para generar cada uno de ellos, e ir iterando por cada uno de los generados hasta encontrar uno que no haya sido usado aún. Para ello dejaremos al sistema ser interactivo e ir mostrando diferentes LDAP cuando responda 0, significa que está libre y 1 que no está libre.
- **Descripción en lenguaje matemático:** No aplica

3. Código Realizado y Análisis

- **Enlace(s) Código:** [Python](#)

4. Casos de Prueba

- **Casos de prueba:**

```
james rodriguez rubio
0
0
1
```

- **Justificación:** Este es el único caso de prueba que use para ver cómo funcionaba medianamente el código ya que no entendía correctamente el problema y creo que el output de ese ejemplo está mal.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- **Descripción del error:** Al obtener el resultado incorrecto de mi envío depure el código imprimiendo los LDAP generados, y estaban mal implementados, no estaba generando correctamente los LDAP ya que no tenía en cuenta que

primero debía usar las letras del primer nombre y a partir de ahí agregar letras del segundo apellido cumpliendo que hasta que no estuvieran todas las letras del primer nombre, las cantidad de letras del segundo apellido tenían que ser menores a la cantidad del primero usando, así generando todas las combinaciones siguiendo las reglas, en el caso con el nombre “*james rodriguez rubio*” no hay problema porque el primer nombre tiene la misma longitud que el segundo apellido, pero con otros casos como usando mi nombre, la lista se queda corta sin todos los posibles LDAP.

- **Proceso de depuración:** La imagen siguiente es toda la cantidad de LDAPs que debería generar el programa correctamente, un total de 18, sin embargo, mi código se queda corto y no completa los 18 sino 14 porque no tiene en cuenta el caso en el que el segundo apellido es más largo que el primer nombre. Además también revise que si hay un nombre demasiado grande la iteración se podría hacer muy larga buscando en orden los usuarios creados y usados, entonces haría falta implementar ese proceso que busque más rápido los usuarios usados o no usados.

```
japastor
japastors
joapastor
joapastors
joapastorsa
johapastor
johapastors
johapastorsa
johapastorsan
johnapastor
johnapastors
johnapastorsa
johnapastorsan
johnapastorsand
johnapastorsando
johnapastorsandov
johnapastorsandova
johnapastorsandoval
```

- Solución corregida: [Python](#)

6. Preguntas de Aprendizaje

- **Temas aplicados:** Búsqueda binaria, bucles for anidados
- **Dificultad de la implementación:** Lo complicado fue entender al final porque fallaba el código, ya que no había comprendido desde un principio la forma correcta de generar los LDAP y con nombres grandes o diferentes a el ejemplo los LDAP no se generaron correctamente, entonces había fallado desde lo más básico. La búsqueda binaria tampoco la había implementado y era necesario para disminuir el tiempo de búsqueda, para no reiterar linealmente, ya que nuestra lista de LDAP estaba ordenada y teníamos que buscar el primer usuario que no estuviera usado.
- **Recursos utilizados:** [binary search](#), [bucles anidados](#).

7. Feedback LLM

- **Envío código LLM:** [CHAT GPT](#)
- **Comparación de su códigos:** Lo más importante es que genere una variable ans que al final no use para nada, tenía el fin de hacer más legible el código pero termina usando los índices mejor.

<code>flush()</code>	Incrustado con <code>import</code>	Función propia, <code>import</code> <code>global</code>
Comparación en búsqueda binaria	Compara string <code>"0"</code> / <code>"1"</code>	Usa <code>int(input())</code> , compara enteros
Variables	Cortas	Más descriptivas
Comentarios de complejidad	No	Sí, detallados
Control caso base búsqueda	Correcto	Incorrecto (<code>query == low</code>)

- **Feedback de LLM:** Pida al LLM que evalúe y corrija su código: Estoy de acuerdo con chat gpt ya que son cambios principalmente de legibilidad y evitar bugs en caso de un input diferente a los esperados, entonces ayuda a manejar los errores y procura el clean code en el código.

1. ❌ `binary_search` usa `input()` como `int(input())`, lo cual rompe el flujo
 - En la versión original se esperaba que el juez devolviera `"1"` o `"0"` como `strings`, no enteros.
2. ❌ Lógica incorrecta al imprimir `! variants[low]` al final
 - Si `low` se sale del rango (porque ningún `0` fue encontrado), puedes obtener un `IndexError`.
 - Además, cuando encuentras un `"0"`, debes recordar el índice, no solo seguir bajando `high`.
3. ❌ Comentarios de complejidad mal ubicados y confusos
 - Algunas complejidades están mal etiquetadas (`O(N^2)` para generación no es correcta si no generas todas las combinaciones posibles).
4. ❌ Código no estructurado con `__name__ == "__main__"` en la segunda versión (aunque lo tenías bien antes).

12. L - Smash the Trash

1. Reducción del Problema

- **Descripción en palabras:** El problema trata sobre la limpieza de la basura de una calle de la ciudad, para esto nos dicen que el alcalde quiere limpiar la basura de n lugares, limpiando la basura del lugar 1 en el día 1 y así sucesivamente. Para esto nos dicen que hay ciertas condiciones, 1 trabajador puede limpiar 1 kilo de basura por día, o mover dos kilos de basura al siguiente lugar, por obvias razones el último lugar sólo puede ser limpiado y no se puede mover la basura. Por lo tanto nos piden que dado n lugares y la cantidad de basura en cada uno de ellos, mostrar la cantidad mínima de trabajadores necesarios para limpiar toda la basura de todos los lugares en n días, teniendo en cuenta las condiciones de los trabajadores.
- **Descripción en lenguaje matemático:**
 - no aplica

2. Reducción de la Solución

- **Descripción en palabras:** Para solucionar el problema, debemos de buscar la cantidad de trabajadores mínimos que necesitamos para limpiar la basura y moverla de forma que el ultimo día quedemos con la misma o mayor cantidad de trabajadores de la basura que hay en el lugar incluyendo si se movió basura de lugares pasados a ese último lugar. Para ello podemos primero validar con que cantidad de trabajadores limpiamos toda la basura, pasandole cantidades arbitrarias de trabajadores, con esto podemos entonces buscar la forma de iterar por una cantidad de trabajadores y probarlos, hasta encontrar el mínimo número de trabajadores necesarios para cumplir con la limpieza del lugar .
- **Descripción en lenguaje matemático:**
 - no aplica

3. Código Realizado y Análisis

- **Enlace(s) Código:** [Python](#)

4. Casos de Prueba

- **Casos de prueba:**

2	3	4	3
30 40	5 1 2	8 1 2 7	0 0 0

- **Justificación:** Use estos casos de prueba para comprobar el funcionamiento del código, el primero si o si el número mínimo debe ser 40 porque si no no hay forma de recoger toda la basura en el último lugar, los dos siguientes son los casos de prueba del vjudge, y el ultimo para verificar que el código funciona cuando le paso una cantidad de lugares pero no hay basura en ellos por lo tanto no se necesitan trabajadores.

5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)

- Estrategia: La estrategia que seguí fue similar a la que llevó implementando desde el problema del sudoku, usar una función validadora y después buscar el valor de los trabajadores iterando. La primera función usa las condiciones que nos dieron en el problema y pasándole una cantidad de trabajadores, los lugares y la cantidad de basura en cada una, determina si es posible o no con una cantidad arbitraria de trabajadores limpiar toda la basura en todos los lugares, si es posible y al ultimo dia hay la misma cantidad de trabajadores o mayor que la cantidad de basura, entonces nos devuelve una confirmación de que la cantidad de trabajadores que le pasamos si funciona. Con esta función realizada, entonces procedo a diseñar una función usando búsqueda binaria, ya que es más útil y eficiente que iterar numero por numero hasta encontrar una cantidad de trabajadores que funcione. La búsqueda binaria la hacemos desde 0 hasta la suma de toda la basura que hay en todos los lugares, así tenemos un límite seguro para la cantidad de trabajadores que recogerán toda la basura. Entonces usamos la búsqueda binaria entre estos valores, donde el valor del “Índice medio” lo vamos probando en el validador y si funciona lo guardamos en una variable, y reducimos el rango para buscar un valor más pequeño en caso de que exista, si no se encuentra otro valor entonces retornamos el valor encontrado y así resolvemos el problema.

6. Preguntas de Aprendizaje


- Temas aplicados: Búsqueda binaria
- Dificultad de la implementación: Solo fue difícil entender un poco como funcionan los movimientos, ya que había que tener en cuenta que si la cantidad de movimientos a realizar era mayor que la cantidad de trabajadores que podían mover basura, entonces esa cantidad de trabajadores no sería suficiente para limpiar la basura en todos los lugares ya que llegaría al punto en que no se podría limpiar toda la basura.
- Recursos utilizados: [búsqueda binaria](#)

7. Feedback LLM

- Envío código LLM: [CHAT GPT](#)
- Comparación de su códigos: El código de chat gpt es más formal y mucho más riguroso dándole menos lugar al error usando fórmulas más seguras en vez de iteraciones tan simples como mi código.

Función principal de decisión	<code>is_valid_workers(n, trash, workers)</code>	<code>feasible(W)</code>
Lógica de transporte	Simula día a día con "carry" acumulado de basura	Usa intervalos <code>[low, high]</code> para el <code>carry_i</code> viable
Condición de rechazo	Si <code>movement > total_trash // 2</code> o <code>final > workers</code>	Si <code>carry</code> excede límites o intervalo inválido (<code>low > high</code>)
Cómo calcula el nuevo carry	<code>carry = 2 * movement</code>	Usa fórmulas: <code>new_low = 2*(xi_low - W)</code> , <code>new_high = xi_high - (xi_high % 2)</code>
Último día (restricción final)	Verifica si <code>trash[-1] + carry <= workers</code>	Requiere <code>t[n-1] + carry_n <= W</code>
Búsqueda binaria	<code>binary_search_workers(n, trash)</code>	Dentro del <code>main()</code> con variables <code>left</code> , <code>right</code> , <code>mid</code>
Complejidad temporal	<code>O(n log sum(trash))</code>	<code>O(n log sum(t))</code> (idéntica)
Complejidad espacial	<code>O(n)</code> por lista <code>trash</code>	<code>O(n)</code> por vector <code>t</code>
Estilo de implementación	Directo, más imperativo, menos generalizable	Más formal, usa rango de estados posibles (<code>low</code> , <code>high</code>)

- Feedback de LLM: Pida al LLM que evalúe y corrija su código: El código me muestra las siguientes recomendaciones, debido a que son solo mejoras de legibilidad y buenas prácticas del código no hay razones por las cuales no aplicarlas.

Problema / Mejora	Detalle	
Comentarios redundantes	Algunos comentarios simplemente repetían el código (<code># O(n)</code> , <code>M(1)</code>), podrías dejarlos solo donde aporten.	
Verificación final duplicada	La condición final <code>if (trash[-1] + carry) > workers:</code> era correcta, pero puede integrarse directamente como <code>return (trash[-1] + carry) <= workers</code> para mayor claridad.	
Nombres más claros	Renombré <code>movement</code> → <code>required_movement</code> para mayor claridad.	
Protección estándar (<code>__main__</code>)	Añadí <code>if __name__ == "__main__":</code> como buena práctica en scripts.	
Legibilidad general	Reordené comentarios y espaciado para mayor limpieza.	