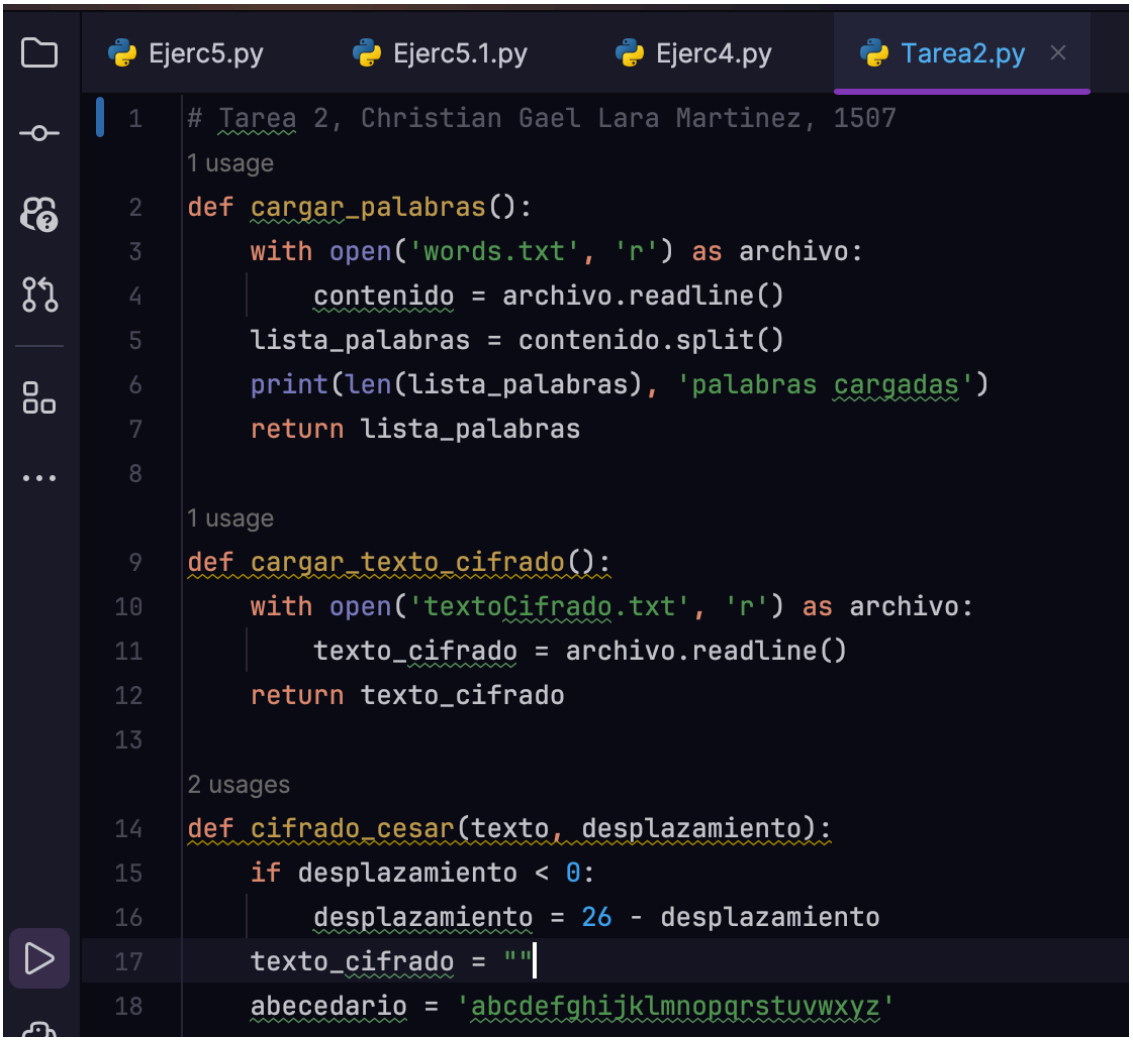


```
Ejerc5.py  Ejerc5.1.py  Ejerc4.py  Tarea2.py
# Tarea 2, Christian Gael Lara Martinez, 1507
1 usage
def cargar_palabras():
    with open('words.txt', 'r') as archivo:
        contenido = archivo.readline()
    lista_palabras = contenido.split()
    print(len(lista_palabras), 'palabras cargadas')
    return lista_palabras

1 usage
def cargar_texto_cifrado():
    with open('textoCifrado.txt', 'r') as archivo:
        texto_cifrado = archivo.readline()
    return texto_cifrado

2 usages
def cifrado_cesar(texto, desplazamiento):
    if desplazamiento < 0:
        desplazamiento = 26 - desplazamiento
    texto_cifrado = ""
    abecedario = 'abcdefghijklmnopqrstuvwxyz'
```

```
    for letra in texto:
        if letra in abecedario:
            indice = abecedario.index(letra)
            nueva_letra = abecedario[(indice + desplazamiento) % 26]
            texto_cifrado += nueva_letra
        else:
            texto_cifrado += letra
    return texto_cifrado

1 usage
def descifrar_cesar(texto, desplazamiento):
    return cifrado_cesar(texto, 26 - desplazamiento)

1 usage
def contar_aciertos(lista_palabras, diccionario):
    aciertos = 0
    for palabra in lista_palabras:
        if palabra in diccionario:
            aciertos += 1
    return aciertos
```

```python
palabras = cargar_palabras()
texto = cargar_texto_cifrado()
max_aciertos = 0
mejor_llave = 0

for llave in range(26):
    texto_descifrado = descifrar_cesar(texto, llave)
    lista_palabras_descifradas = texto_descifrado.split()
    aciertos = contar_aciertos(lista_palabras_descifradas, palabras)

    if aciertos > max_aciertos:
        max_aciertos = aciertos
        mejor_llave = 26 - llave

print("La llave encontrada es:", mejor_llave)
print("Texto descifrado:")
print(cifrado_cesar(texto, mejor_llave))

```

```
/usr/local/bin/python3.12 /Users/christianlara/Desktop/DisenoAnalisisAlgoritmos/Tarea2.py
55900 palabras cargadas
La llave encontrada es: 8
Texto descifrado:
choose a job you love, and you will never have to work a day in your life - a winner is a dreamer who never gives up, so always give the

Process finished with exit code 0
```

EXPLICACIÓN DEL CÓDIGO:

`def cargar_palabras():`

Esta función tiene la tarea de leer un archivo llamado `words.txt` y cargar su contenido. El archivo contiene palabras que serán utilizadas más adelante en el programa. La función abre el archivo en modo de lectura (`'r'`), lee la primera línea y la separa en palabras individuales usando `split()`, que convierte el texto en una lista de palabras. Luego imprime la cantidad de palabras cargadas y devuelve esa lista.

`def cargar_texto_cifrado():`

Esta función abre y lee un archivo llamado `textoCifrado.txt`, que contiene el texto cifrado que el programa va a descifrar. Al igual que la función anterior, abre el archivo en modo de lectura (`'r'`), lee la primera línea del archivo y devuelve ese texto como una cadena de caracteres.

`def cifrado_cesar(texto, desplazamiento):`

Esta función implementa el cifrado César, que es una técnica para cifrar un mensaje desplazando cada letra en el alfabeto por un número dado de posiciones.

1. Si el desplazamiento es negativo, lo ajusta para que no lo sea, utilizando la fórmula `26 - desplazamiento`.
2. Inicializa una cadena vacía (`texto_cifrado`) para almacenar el mensaje cifrado.
3. Define el alfabeto como una cadena con todas las letras minúsculas del inglés.
4. Luego, recorre cada letra del texto y, si esa letra está en el alfabeto, encuentra su posición (`indice`) y la reemplaza por otra letra, desplazada por el valor indicado. Si la letra no está en el alfabeto (por ejemplo, un espacio o símbolo), se agrega tal cual.
5. Al final, devuelve el mensaje cifrado.

`def descifrar_cesar(texto, desplazamiento):`

Esta función utiliza la misma lógica del cifrado César, pero en lugar de cifrar, descifra el mensaje. Para hacerlo, usa la función `cifrado_cesar`, pero invierte el desplazamiento aplicando `26 - desplazamiento`. De esta manera, recorre las letras hacia atrás en el alfabeto, restaurando el texto original.

`def contar_aciertos(lista_palabras, diccionario):`

Esta función toma una lista de palabras descifradas y un diccionario de palabras válidas. La tarea de la función es contar cuántas de esas palabras descifradas están en el diccionario. Recorre cada palabra de la lista y, si esa palabra se encuentra en el diccionario, incrementa un contador (`aciertos`). Al final, devuelve el número total de coincidencias o aciertos.

Variables `palabras`, `texto`, `max_aciertos`, `mejor_llave`

- `palabras`: Almacena la lista de palabras cargadas desde el archivo `words.txt`, obtenida con la función `cargar_palabras()`.

- `texto`: Contiene el texto cifrado cargado desde `textoCifrado.txt`, obtenido con la función `cargar_texto_cifrado()`.
- `max_aciertos`: Guarda el número máximo de coincidencias encontradas entre las palabras descifradas y el diccionario.
- `mejor_llave`: Guarda el valor de la llave que resulta en el mayor número de coincidencias (aciertos).

Bucle `for llave in range(26):`

Este bucle itera a través de todas las posibles llaves de desplazamiento (del 0 al 25, ya que el alfabeto tiene 26 letras).

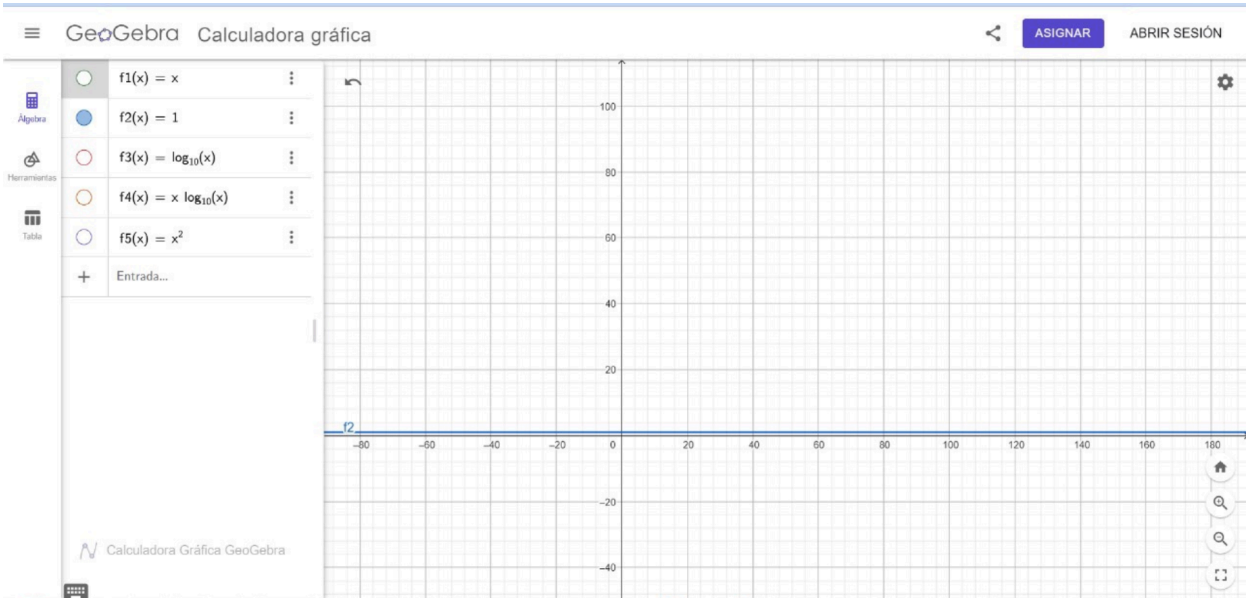
1. En cada iteración, descifra el texto usando la función `descifrar_cesar` y la llave actual.
2. Luego, convierte el texto descifrado en una lista de palabras.
3. Llama a la función `contar_aciertos` para determinar cuántas palabras descifradas coinciden con las del diccionario.
4. Si el número de aciertos con la llave actual es mayor que el número máximo registrado hasta el momento, actualiza el valor de `max_aciertos` y `mejor_llave`.

`print("La llave encontrada es:", mejor_llave)`

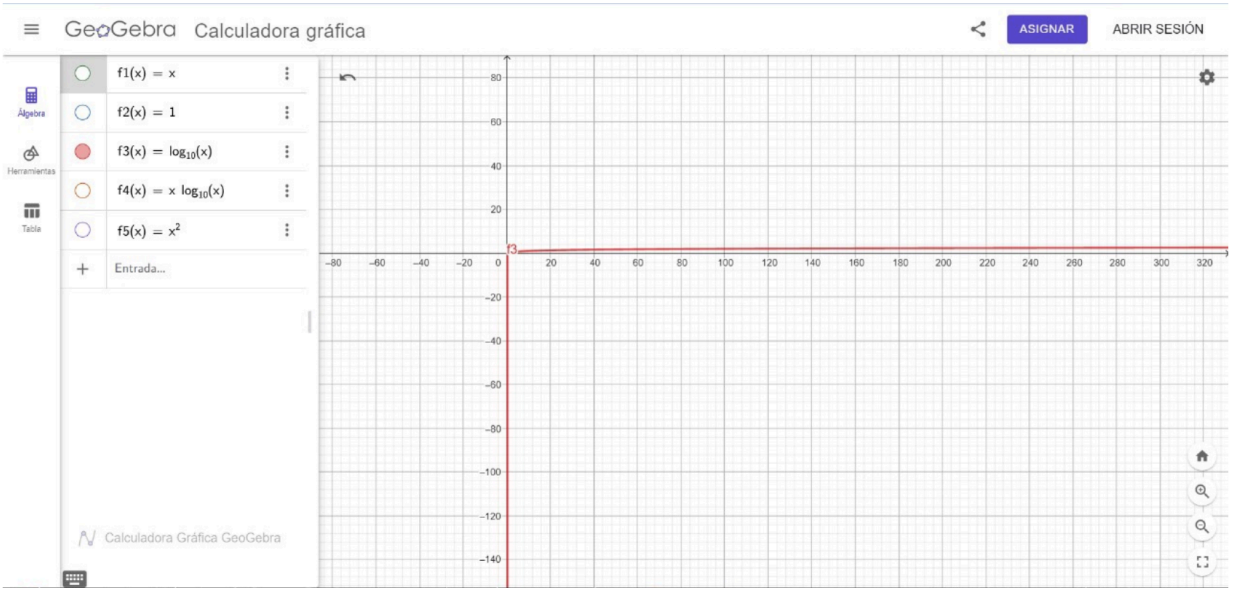
Después de que el bucle termina, imprime la mejor llave que logró descifrar la mayor cantidad de palabras.

`print("Texto descifrado:")`

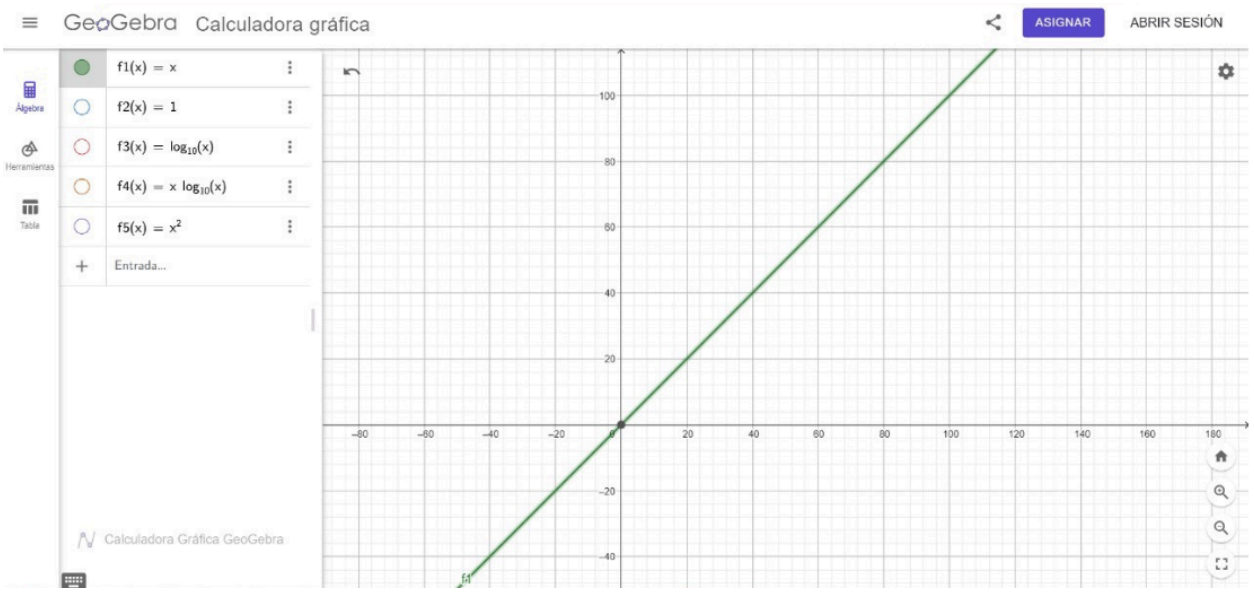
Imprime el texto descifrado usando la mejor llave encontrada y la función `cifrado_cesar`.



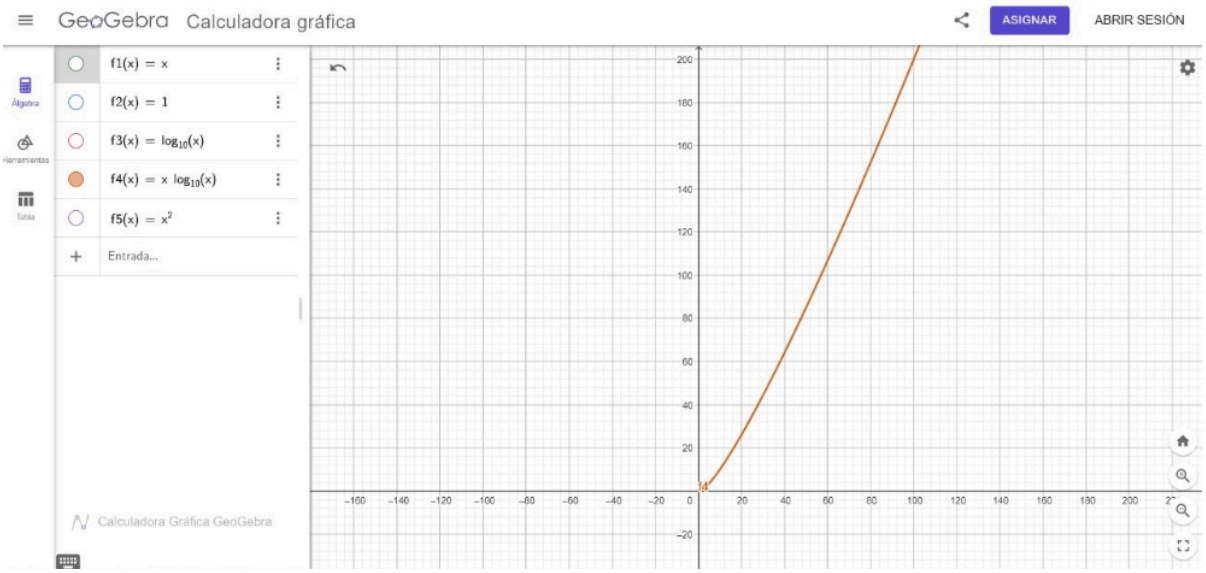
Sin lugar a dudas, esta es la mejor opción que podrías considerar. No importa en lo En resumen, una función constante O(1)O(1)O(1) es mejor en notación Big O porque es **la más eficiente**: su tiempo de ejecución es fijo y no depende del tamaño de los datos. En problemas donde la entrada es grande, esta eficiencia resulta crucial.



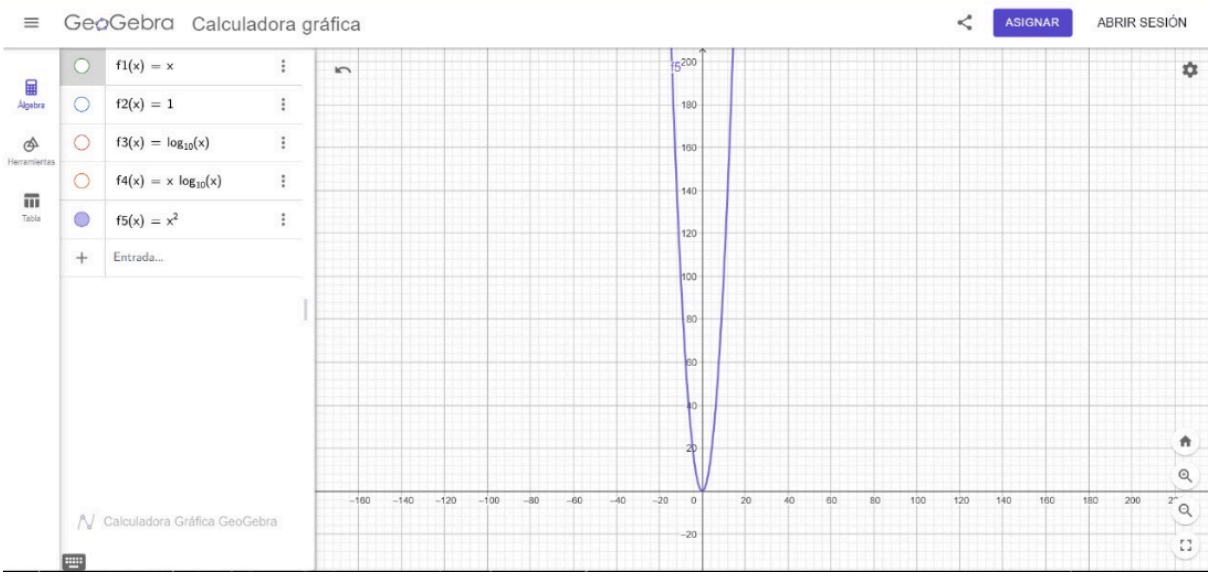
Las funciones logarítmicas crecen mucho más lentamente que las funciones lineales o cuadráticas. Esto significa que, aunque el tamaño de los datos aumente significativamente, el tiempo de ejecución de una función logarítmica no se incrementa de forma dramática.



Una función con complejidad lineal O(n)O(n)O(n) es mejor porque proporciona un equilibrio ideal entre tiempo de ejecución y escalabilidad, lo que la hace más eficiente para manejar grandes conjuntos de datos en la práctica.



Una función con complejidad **O(n log n)** realiza muchas menos operaciones que una **O(n²)** conforme el tamaño del problema (n) crece.
Este tipo de complejidad es común en algoritmos eficientes como los de ordenamiento o búsqueda, lo que los hace ideales para grandes volúmenes de datos.
O(n log n) es una excelente combinación de velocidad y eficiencia,



En la notación Big O, **una función cuadrática no es mejor** que otras con menor complejidad como O(n) o O(log n). De hecho, en términos de eficiencia, las funciones cuadráticas (O(n²)) suelen ser **peores** en comparación con otras de menor orden porque crecen más rápido conforme aumenta el tamaño del input.