3.
Secuencia:

- $s_0 = 0$
- $s_1 = 1$
- $s_2 = 2$
- $s_n = 5s_{n-1} + 4s_{n-2} - 2s_{n-3}$ para $n \geq 3$

a)
Se pretende plantear una solución para la recurrencia utilizando exponenciación y producto de matrices, usando la estrategia del algoritmo *fexpmat*().

Antes de plantear la solución se van a definir algunas funciones. La siguiente función *matMul*, toma dos matrices, $m1$ y $m2$, y devuelve el producto de matrices $m1 \times m2$. Observe que las matrices pueden ser de cualquier tamaño mayor o igual a 1; el único requisito es que el número de columnas de $m1$ sea igual al número de filas de $m2$, si no lo son devuelve una lista vacía.

```
def matMul(m1, m2):
    if len(m1[0]) != len(m2):
        return []
    filas1 = len(m1)
    columnas2 = len(m2[0])
    rangoK = len(m1[0])
    n = len(m1)
    matRes = []
    for i in range(filas1):
        vector = []
        for j in range(columnas2):
            vector.append(0)
        matRes.append(vector)
    for i in range(filas1):
        for j in range(columnas2):
            for k in range(rangoK):
                producto = m1[i][k]*m2[k][j]
                matRes[i][j] += producto
    return matRes
```

La siguiente función *matriz_identidad* toma un número (*lado*) y devuelve la matriz identidad de tamaño *lado* $\times$ *lado*.

```
def matriz_identidad(lado):
    matriz = []
    for i in range(lado):
        vector = []
        for j in range(lado):
            if j == i:
```

```
                vector.append(1)
            else:
                vector.append(0)
        matriz.append(vector)
    return matriz
```

La siguiente función *potMat* toma una matriz (*matriz*), un número (*potencia*) y devuelve el resultado de elevar esa matriz a esa potencia ($matriz^{potencia}$).

```
def potMat(matriz, potencia):
    n = potencia
    if n == 0:
        return matriz_identidad(len(matriz))
    elif n%2 == 1:
        return matMul(matriz, potMat(matriz, n-1))
    else:
        matAux = potMat(matriz, n//2)
        return matMul(matAux, matAux)
```

Para solucionar el problema se necesita una matriz m tal que al multiplicar m^n por la izquierda con el vector columna

$$vc = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

se obtenga lo siguiente

$$m^n \times vc = \begin{bmatrix} s_{n+2} \\ s_{n+1} \\ s_n \end{bmatrix}$$

y así poder obtener el valor s_n.

Por los coeficientes de la sucesión, se sabe que la matriz tiene la forma

$$m = \begin{bmatrix} 5 & 4 & -2 \\ a & b & c \\ d & e & f \end{bmatrix}$$

Calculando el valor de s_3 se obtiene:
$s_3 = 5s_2 + 4s_1 - 2s_0$
$= 5(2) + 4(1) - 2(0)$
$= 14$

Entonces

$$\begin{bmatrix} 5 & 4 & -2 \\ a & b & c \\ d & e & f \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 14 \\ 2 \\ 1 \end{bmatrix}$$

Por la anterior observación, se tienen que satisfacer las siguientes ecuaciones:
$2a + b = 2$
$2d + e = 1$
Se eligirá la siguiente asignación de valores:

- a: 1
- b: 0
- c: 0
- d: 0
- e: 1
- f: 0

Es decir, la matriz m tiene las siguientes entradas:

$$m = \begin{bmatrix} 5 & 4 & -2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Así, el algoritmo para calcular s_n consiste en los siguientes pasos:

1. Dada la matriz m, calcular m^n.
2. Multiplicar m^n (por la izquierda) con el vector columna $vc = [2, 1, 0]$ y obtener el vector columna $vr = [s_{n+2}, s_{n+1}, s_n]$.
3. Devolver la última entrada (la que está más abajo) del vector columna vr.

La siguiente función *secuencia* calcula el n-ésimo elemento de la sucesión descrita. En la línea 8 se calcula la matriz m^n y se guarda en la variable `mat_n`. En la línea 9 se calcula el producto de `mat_n` con `vc` y se guarda en la variable `vr`. En la 10 se devuelve la última entrada del vector columna `vr` (fila 2, columna 0).

```
 1:def secuencia(n):
 2:    m = [[5,4,-2],
 3:         [1,0,0],
 4:         [0,1,0]]
 5:    vc = [[2],
 6:          [1],
 7:          [0]]
 8:    mat_n = potMat(m, n)
 9:    vr = matMul(mat_n, vc)
10:    return vr[2][0]
```

Ahora se procederá a demostrar que el algoritmo es correcto usando inducción sobre n.

Caso base: $n = 0$.

Si $n = 0$, la función *potMat(m, 0)* devuelve la matriz identidad de 3×3 (pues m es de 3×3). Por una propiedad del álgebra lineal se tiene que $I \times M = M$ para cualquier matriz M, donde I es la matriz identidad. Así, el resultado de multiplicar `mat_n` por `vc` es `vc` (es decir, `vr` será igual a `vc`). Entonces, el resultado de devolver la entrada [2][0] de `vr` es 0, así que para este caso, $secuencia(0) = 0 = s_0$, y por lo tanto es correcto.

H. I. Existe una $k \geq 0$ tal que al ejecutar el algoritmo $secuencia(k)$, al final se cumple que $mat\_n = m^k$ (por definición de `potMat`) y que $m^k \times vc$ (resultado de la función `matMul(mat_n, vc)`) es igual al vector columna

$$\begin{bmatrix} s_{k+2} \\ s_{k+1} \\ s_k \end{bmatrix}$$

el cual se guarda en `vr`, y por lo tanto, $secuencia(k) = s_k$.

Se demostrará que para $secuencia(k + 1)$, `vr` guarda el vector columna

$$\begin{bmatrix} s_{k+3} \\ s_{k+2} \\ s_{k+1} \end{bmatrix}$$

y entonces $secuencia(k + 1) = s_{k+1}$.

Demostración.

Se sabe que para $secuencia(k+1)$, el algoritmo primero calcula m^{k+1} y luego multiplica la matriz resultante por el vector vc; es decir, se realiza la operación $m^{k+1} \times vc$ y se guarda el resultado en vr. Pero se sabe que la multiplicación de matrices es asociativa, así que $m^{k+1} \times vc = m \times (m^k \times vc)$.

Por hipótesis se sabe que

$$m^k \times vc = \begin{bmatrix} s_{k+2} \\ s_{k+1} \\ s_k \end{bmatrix}$$

Así pues

$$m^{k+1} \times vc = m \times (m^k \times vc) = \begin{bmatrix} 5 & 4 & -2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} s_{k+2} \\ s_{k+1} \\ s_k \end{bmatrix} = \begin{bmatrix} 5s_{k+2} + 4s_{k+1} - s_k \\ s_{k+2} \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} s_{k+3} \\ s_{k+2} \\ s_{k+1} \end{bmatrix}$$

Este vector resultante se guarda en `vr`. Observe que s_{k+3} se obtiene de la definición de la secuencia.

Al final, el algoritmo devuelve la última entrada del vector columna `vr`, la cual es s_{k+1}, y por lo tanto $secuencia(k + 1) = s_{k+1}$. ■

b)

La operación de multiplicación de matrices tiene complejidad $O(\ell^3)$, donde ℓ es el máximo entre los siguientes valores:

- Número de columnas de `m1`.
- Número de columnas de `m2`.
- Número de filas de `m1`.
- Número de filas de `m2`.

Pero para el caso del algoritmo de *secuencia* se sabe que `m1` sólo puede ser una matriz de 3×3 y `m2` puede ser una matriz de 3×3 o de 3×1. Así que el peor caso se da cuando tanto `m1` y `m2` son de 3×3.

Se contará el número de operaciones que realiza `matMul(m1, m2)` donde ambas matrices son de 3×3. Para simplificar, sólo se van a contar las operaciones lógico-aritméticas (la operación `+=` contará como una operación aritmética). En la línea 2 se realiza una comparación. En las línea 17 se realiza una multiplicación y en la 18 una suma con asignación. Se sabe que para este caso $filas1 = columnas2 = rangoK = 3$, así que se realizan $3^3 = 27$ multiplicaciones y 27 sumas con asignación. Así, el número total de operaciones será 1+27+27=55.

La función `matriz_identidad` construye una matriz de 3×3 (para nuestro caso particular) y realiza $3 \times 3 = 9$ operaciones.

Ahora se calculará el número de operaciones de la función $potMat(matriz, n)$ para el peor caso.

1. En el caso base, cuando n es 0, se realiza una comparación más las 9 operaciones de `matriz_identidad`, así que son 10 en total.
2. Si n es impar entonces se realizan 2 comparaciones, 55 operaciones de la función $matMul$, más el número de operaciones de $potMat(matriz, n-1)$. Por lo tanto son 57 + **numOps**$(potMat(matriz, n-1))$, donde **numOps** es el número de operaciones de la función entre paréntesis. Observe que si n es impar entonces $n-1$ es par, así que la siguiente llamada recursiva cae en el caso 3.
3. Si n es par entonces se realizan 2 comparaciones más el número de operaciones de $potMat(matriz, n/2)$, más las 55 operaciones de $matMul$. Así que son 57 + **numOps**$(potMat(matriz, n/2))$.

En el peor caso, cada dos llamadas recursivas el argumento n de $potMat$ será un número impar. Pero cuando no es impar se divide n a la mitad. En el peor caso se realizan 57+57=114 operaciones por el número de llamadas recursivas para el caso par. Como n se divide a la mitad para cada llamada donde el argumento es par, entonces se realizan $114 \log(n)$ operaciones en el peor caso.

Finalmente, se calcularán las operaciones de la función *secuencia*.

- La función $potMat(m, n)$ (línea 8) realiza a lo sumo $114 \log(n)$ operaciones.
- La función $matMul(mat\_n, vc)$ realiza menos de 55 operaciones.

Así, $114 \log(n) + 55$ es una cota superior para el número de operaciones de *secuencia*(n), lo cual es $O(\log n)$. ■