1. Tenemos una función que define dos variables enteras. las inicializa (“j” con 1, e “i” con n*n). Después inicia un ciclo “for” que se repite mientras “i” sea mayor que cero y en cada iteración “i” se divide entre 2. Dentro del ciclo se define una variable llamada “suma” (entera), y se inicia con la suma de i+j.
   se imprime en pantalla “Suma” seguido del valor. y finalmente se incrementa el valor de “j” una unidad.
   el ciclo se ejecuta log2(n * n).

   Entonces:

   La primera línea declara variables enteras, se ejecuta una vez
   la segunda línea inicializa “j” con 1, se ejecuta una vez
   la tercera línea inicializa “i” n*n, se ejecuta una vez
   el ciclo for (se ejecuta mientras “i” sea mayor que cero, se ejecuta log2(n*n)+1 veces
   la última línea dentro del ciclo incrementa “j”, se ejecuta log2(n*n) veces.

   ¿Qué se obtiene al ejecutar algoritmo 1(8)?:

   En la primera línea se declara la variable “i” y se inicializa “j” en 1.
   En la segunda línea, “i” se inicializa como n*n osea 8*8 (qué es 64)
   El ciclo “for” se ejecuta mientras “i” sea mayor a 0. como “i” se inicializó en 64, se ejecutará log2(64)+1 veces (osea 7 veces)
   En la primera iteración del ciclo “suma” se inicializa como i+j (osea 64+1, qué es 65), en pantalla se imprime “Suma 65”.
   En la segunda iteración, “i” se actualiza a “i/2”, osea “64/2” que es 32. “j” se incrementa a 2. y “Suma” se actualiza a “i+j”, osea “32+2” que es 34. y se imprime “Suma 34”
   En la tercera iteración del ciclo “i”, se actualiza a “i/2”, que sería “32/2” que es 16. “j” se incrementa a 3, y “suma” se actualiza a “i+j”, ósea “16+3” que da 19. y se imprime “Suma 19”.
   En la cuarta iteración del ciclo, “i” se actualiza a “i/2”, es decir “16/2” que da 8. “j” se incrementa a 4, y “suma” se actualiza a “i+j”, es decir 8+4” que da 12. y se imprime “Suma 12”.
   En la quinta iteración del ciclo, “i” se actualiza a “i/2”, es decir “8/2” que da 4. “j” se incrementa a 5, y “suma” se actualiza a “i+j”, entonces queda “4+5” que da 9. y se imprime “Suma 9”.
   En la sexta iteración del ciclo, “i” se actualiza a “i/2”, queda “4/2” que da 2. “j” se incrementa a 6, y “suma” se actualiza a “i+j”, es decir “2+6” que da 8. y se imprime “Suma 8".
   En la séptima y última iteración del ciclo, “i” se actualiza “i/2”, es decir “2/2” que da 1. “j” se incrementa a 7, y “suma” se actualiza a “i+j”, que es “1+7” que da 8. y se imprime “Uma 8”.
   Como el valor de “i” ahora es 0, el ciclo “for” termina y la función también.

   (la consola deberia verse asi:
   Suma 65
   Suma 34
   Suma 19

Suma 12
Suma 9
Suma 8
Suma 8

Finalmente, la complejidad del algoritmo es O(log(n^2)). una complejidad logarítmica con respecto al cuadrado de “n”. Esta complejidad se debe a la iteración del ciclo “for” que se realiza un número logarítmico de veces con respecto a n^2, y la división de “i” dentro del ciclo, que también tiene un tiempo logarítmico.

2.

La primera línea se ejecuta una sola vez al inicio de la función.
La segunda línea se ejecuta una sola vez al inicio de la función.
La tercera línea se ejecuta (2 * n) / 4 veces, que es lo mismo que n / 2 veces.
La cuarta línea se ejecuta raíz cuadrada de (n) veces en el peor de los casos, ya que el bucle se detiene cuando j * j supera a n.
La línea quinta se ejecuta raíz cuadrada de (n) * (n / 2) veces, ya que el segundo bucle se ejecuta raíz cuadrada de (n) veces y el primer bucle se ejecuta n / 2 veces. Por lo tanto, el cuerpo del segundo bucle for se ejecuta raíz cuadrada de (n) veces por cada iteración del primer bucle.

La complejidad del algoritmo es O(n*raíz cuadrada de (n)). porque el segundo bucle “for” se ejecuta raíz cuadrada de (n) veces en el peor de los casos, y el primer bucle “for” se ejecuta n/2 veces.

3.

La primera línea se ejecuta una vez, porque es la definición de la función.
La segunda línea una vez, se declaran las variables “i,j,k”
La tercera línea se ejecuta n-1 veces, ya que comienza en “n” y disminuye en uno hasta llegar a 2.
La cuarta línea se ejecuta a veces en cada iteración del bucle “for” anterior. se ejecuta n*(n-1) veces.
La quinta línea se ejecuta “i” veces en cada iteración del bucle anterior. se ejecuta (n^2 - n)/2 veces.
La sexta línea se ejecuta tantas veces como se indica en el resultado del tercer bucle “for” que es (n^3 - n)/6.

La complejidad es de O(n^3), porque son 3 bucles anidados y cada uno itera n, n-1, y n/2 veces. y se realizan una cantidad fija de impresiones en pantalla en cada iteración.

4.

La primera línea se ejecuta una vez
La segunda línea se ejecuta una vez
La tercera línea se ejecuta una vez
La cuarta línea se ejecuta n+1 veces

La quinta y sexta línea se ejecuta n veces
La septima linea se ejecuta n*(n+1)/2 veces
La octava, novena y décima línea se ejecutan n*(n-1)/2 veces
En la línea 13, 14, y 15 del “else” se ejecuta n*(n-1)/2 veces
La línea 17 del “++j;” se ejecuta n*(n+1)/2 veces
Y la línea 20 del “return” se ejecuta una vez

La complejidad de tiempo del algoritmo es O(n^2) debido a que hay dos bucles anidados que iteran a través de la matriz "valores".

Este es un algoritmo que toma una matriz de enteros "valores" de longitud "n" y devuelve un contador de cuántas veces se cumple una determinada condición en la matriz.

5.

Las líneas 1 y 2 se ejecutan una vez.
La línea 3 se ejecuta 6 veces.
La línea 4 se ejecuta 6 veces
Y la línea 5 se ejecuta 6 veces
(Esto es en el caso de que “n” sea un número divisible exactamente entre 5)

La complejidad es de O(n) porque depende del valor de “n” para definir qué tanto es el recorrido que se debe hacer en el algoritmo.

6.

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Tamaño entrada	Tiempo	Tamaño entrada	Tiempo
5	0.061s	35	13.193s
10	0.076s	40	2m.27.752s
15	0.101s	45	
20	0.057s	50	
25	0.132s	60	
30	1.709s	100	

La complejidad es O(2^n). donde “n” es el número de términos de la secuencia fibonacci que se desee calcular.

8.

Tamaño entrada	Tiempo solución propio	Tiempo solucion profesores
100	0.035s	
1000	0.098s	
5000	2.805s	
10000	12.696s	
50000	5 min 10.613s	
100000		
200000		