PRACTICA III

Código I (Búsqueda Binaria).

Aplicando el teorema maestro a esta ecuación de recurrencia, podemos ver que a = 1, b = 2 y f(n) = O(1). Como f(n) = O(n^(log\_b(a) - e)) para e > 0, podemos aplicar el primer caso del teorema maestro y concluir que T(n) = Theta(log n).

Esto significa que la complejidad de tiempo de la función busquedaBinariaRecursiva es O(log n), donde n es el tamaño del arreglo.

Código II (Stoge Sort)

Aplicando el teorema maestro a esta ecuación de recurrencia, podemos ver que a = 3, b = 3/2 y f(n) = O(1). Como f(n) = O(n^(log\_b(a) - e)) para e > 0, podemos aplicar el primer caso del teorema maestro y concluir que T(n) = Theta(n^(log\_b(a))) = Theta(n^log\_(3/2)(3)).

Esto significa que la complejidad de tiempo de la función stoogesort es O(n^log\_(3/2)(3)), donde n es el tamaño del subarreglo que se está ordenando.

Código III (Factorial Recursivo)

La función factorial en cada llamada recursiva, la función realiza una cantidad constante de trabajo para multiplicar n por el resultado de la llamada recursiva con n-1. La función tiene una profundidad de recursión de n, por lo que su complejidad de tiempo es O(n).

Esto significa que la complejidad de tiempo de la función factorial es O(n), donde n es el argumento de la función.

Código IV (Fibonacci)

Si hacemos esto, encontramos que T(n) = O(2^n), lo que significa que la complejidad de tiempo de la función fibonacci es exponencial.

Esto significa que la complejidad de tiempo de la función fibonacci es O(2^n), donde n es el argumento de la función.