```
import java.util.Scanner;
import java.util.Stack;

public class AckermannIterativa {

    public static int ackermann(int m, int n) {
        Stack<Integer> stack = new Stack<>();
        stack.push(m);

        while (!stack.isEmpty()) {
            m = stack.pop();

            if (m == 0) {
                n = n + 1;
            } else if (n == 0) {
                stack.push(m - 1);
                n = 1;
            } else {
                stack.push(m - 1);
                stack.push(m);
                n = n - 1;
            }
        }

        return n;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese el valor de m: ");
        int m = scanner.nextInt();

        System.out.print("Ingrese el valor de n: ");
        int n = scanner.nextInt();

        int resultado = ackermann(m, n);
        System.out.println("Ackermann(" + m + ", " + n + ") = " + resultado);
    }
}
```

Función a analizar

```
while (!stack.isEmpty()) {          // O(1)
   m = stack.pop();                 // O(2)

   if (m == 0) {                    // O(1)
      n = n + 1;                    // O(2)
   } else if (n == 0) {             // O(1)
      stack.push(m - 1);            // O(1)
      n = 1;                        // O(1)
   } else {
      stack.push(m - 1);            // O(1)
      stack.push(m);                // O(1)
      n = n - 1;                    // O(2)
   }
}
```

(MAX(1,MAX(2,MAX(1,MAX(1,2)))) + 1)*T#ciclos

3*T#ciclos

Número de iteraciones (T#`ciclos`)

El número total de iteraciones del ciclo `while` está determinado por el comportamiento de la función de Ackermann, ya que este algoritmo simula dicha función de forma iterativa. La función de Ackermann se define como:

```
A(m, n) =
    n + 1                   si m = 0
    A(m - 1, 1)             si m > 0 y n = 0
    A(m - 1, A(m, n - 1)) si m > 0 y n > 0
```

Dado que esta función crece más rápido que cualquier función primitiva-recursiva, el número de iteraciones está relacionado directamente con su resultado:

```
T#ciclos ≈ A(m, n)
```

Tiempo total

Si cada iteración tiene un costo constante de 3 operaciones, el tiempo total del algoritmo será:

`T(m, n) = 3 × A(m, n)`