Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

Campus Sonora Norte

Act 1.1 - Funciones Iterativas, Recursivas y su análisis de Complejidad

Curso:

Programación de estructuras de datos y algoritmos fundamentales

(TC1031)

Estudiante:

Daniel Alfredo Barreras Meraz A01254805

Docente:

Baldomero Olvera Villanueva

Fecha de entrega:

27 de agosto de 2023

Código fuente

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

/* Este programa calcula la sumatoria de los primeros n números naturales utilizando tres métodos diferentes:
  iterativo, recursivo y directo.
  Autor: Daniel Alfredo Barreras Meraz
  Matrícula: A01254805
  Fecha de creación/modificación: 28/08/2023
*/


/* Este algoritmo tiene una complejidad de O(n) ya que el for se ejecuta n veces y cada vez que se ejecuta
se hace una operación constante (suma += i) por lo que la complejidad es O(n) * O(1) = O(n) */
int sumatoriaIterativa(int n){
  if (n <= 0)
    return 0;

  int suma = 0;
  for (int i = 0; i <= n; i++)
    suma += i;

  return suma;
}

/* Este algoritmo tiene una complejidad de O(n) ya que la función se llama a sí misma n veces y cada vez que se llama
  se hace una operación constante (n + sumatoriaRecursiva(n - 1)) por lo que la complejidad es O(n) * O(1) = O(n) */
int sumatoriaRecursiva(int n){
  if (n <= 0)
    return 0;

  if (n == 1)
    return 1;
  else
    return n + sumatoriaRecursiva(n - 1);

}

/* Este algoritmo tiene una complejidad de O(1) ya que solo se hace una operación constante (n * (n + 1)) / 2
  por lo que la complejidad es O(1) */
int sumatoriaDirecta(int n){
  if (n <= 0)
    return 0;

  return (n * (n + 1)) / 2;
}
```

```cpp
int main(){
  std::ifstream file("input.txt");
  std::string line;

  std::vector<int> iterativeResults, recursiveResults, directResults;

  // Lecutra de casos de prueba del archivo input.txt
  while (std::getline(file, line)){
    int n = std::stoi(line);
    iterativeResults.push_back(sumatoriaIterativa(n));
    recursiveResults.push_back(sumatoriaRecursiva(n));
    directResults.push_back(sumatoriaDirecta(n));
  }

  file.close();

  std::cout << "Casos de prueba, algoritmo iterativo: " << std::endl;
  for (int result : iterativeResults)
    std::cout << result << std::endl;

  std::cout << "\n" << "Casos de prueba, algoritmo recursivo: " << std::endl;
  for (int result : recursiveResults)
    std::cout << result << std::endl;

  std::cout << "\n" << "Casos de prueba, algoritmo directo: " << std::endl;
  for (int result : directResults)
    std::cout << result << std::endl;

return 0;
}
```

Casos de prueba

Caso de prueba	**Entrada**	**Salida esperada (Iterativo)**	**Salida esperada (Recursivo)**	**Salida esperada (Directo)**
1	-5	0	0	0
2	-10	0	0	0
3	0	0	0	0
4	15	120	120	120
5	20	210	210	210
6	25	325	325	325

Comprobación de casos de prueba

```
Casos de prueba, algoritmo iterativo:
0
0
0
120
210
325

Casos de prueba, algoritmo recursivo:
0
0
0
120
210
325

Casos de prueba, algoritmo directo:
0
0
0
120
210
325
```

Explicación de complejidad algorítmica utilizando ecuaciones recurrentes

Algoritmo iterativo: Este algoritmo declara una variable inicializada en 0 y utiliza un bucle for para calcular la sumatoria. La variable se inicializa una vez y el bucle se ejecuta n veces y en cada iteración se realiza una operación constante (suma += i). Por lo tanto, la complejidad de tiempo del algoritmo iterativo es O(n) * O(1) = O(n).

Podemos escribir la ecuación recurrente para este algoritmo como:

$$T(n) = T(n - 1) + 1$$

Donde T(n) es el tiempo de ejecución para calcular la sumatoria de los primeros n números naturales.

Si resolvemos esta ecuación, obtenemos

$$T(n) = T(n - 1) + 1$$

$$= T(n - 2) + 1 + 1$$

$$= T(n - 2) + 1 + 1 = T(n - 3) + 1 + 1 + 1 \ldots$$

$$= T(1) + 1 + \ldots + 1.$$

$$= 1 + (n - 1)$$

$$= n$$

Por lo tanto, la complejidad de tiempo del algoritmo iterativo es O(n).

Algoritmo recursivo: Este algoritmo utiliza recursión para calcular la sumatoria. La función se llama a sí misma n veces y en cada llamada se realiza una operación constante (n + sumatoriaRecursiva(n - 1)). Como resultado, la complejidad de tiempo del algoritmo recursivo es O(n) * O(1) = O(n).

Podemos escribir la ecuación recurrente para este algoritmo como:

$$T(n) = T(n - 1) + 1$$

Donde T(n) es el tiempo de ejecución para calcular la sumatoria de los primeros n números naturales.

Si resolvemos esta ecuación, obtenemos

$$T(n) = T(n - 1) + 1$$

$$= T(n - 2) + 1 + 1$$

$$= T(n - 2) + 1 + 1 = T(n - 3) + 1 + 1 + 1 \ldots$$

$$= T(1) + 1 + \ldots + 1.$$

$$= 1 + (n - 1)$$

$$= n$$

Por lo tanto, la complejidad de tiempo del algoritmo recursivo es O(n).

Algoritmo directo: Este algoritmo utiliza una fórmula matemática, la cual es comprobable que es equivalente a la sumatoria de números para calcular la sumatoria directamente sin utilizar un bucle o recursión. Solo se realiza una operación constante $\frac{n*(n+1)}{2}$. Esto nos dice que la complejidad de tiempo del algoritmo directo es O(1).

Podemos escribir la ecuación recurrente para este algoritmo como:

$$T(n) = 1$$

Esta ecuación ya está resuelta, por lo tanto:

$$T(n) = 1$$

La complejidad de tiempo del algoritmo iterativo es O(1).