UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Redes de computadoras
2024-1

Práctica 1.2: Reporte.

Profesor:
Verónica Esther Arriola Ríos

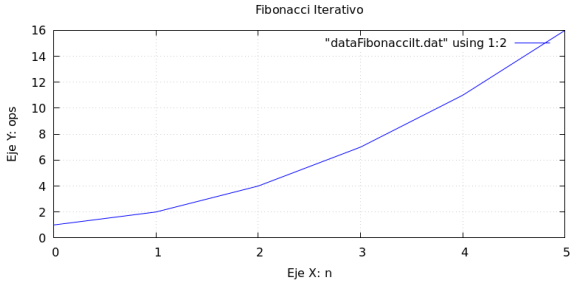
Ayudantes de teoría:
Oscar José Hernández Sánchez
Tania Michelle Rubí Rojas

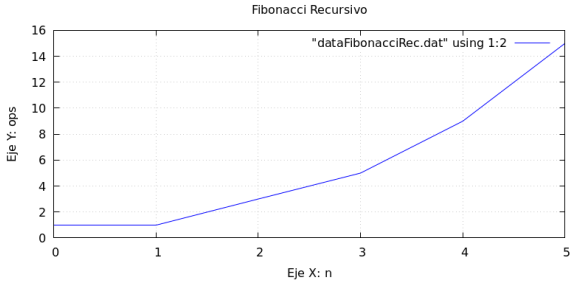
Ayudantes de laboratorio:
Cynthia Lizbeth Sánchez Urbano
Erick Bernal Márquez

Alumno:
Ducloux Hurtado Axel - 316309132

Reporte de la práctica.

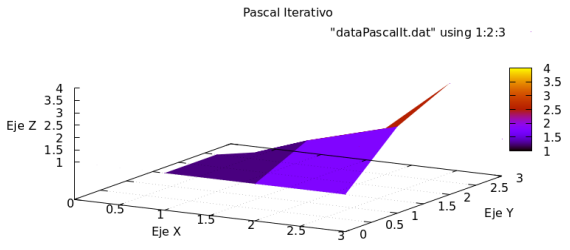
1. Para el método de Fibonacci, genera las gráficas n(entrada) vs número de operaciones y haz un análisis de lo que sucede. ¿Cuál es el orden de complejidad? Justifica.

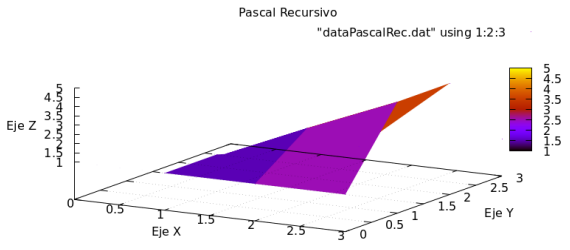




Como podemos apreciar en las imágenes Fibonacci Iterativo se asemeja más a una función lineal ya que los valores no hacen saltos tan grandes entre cada evaluación mientras que en la forma recursiva si. Pero esto lo abordaremos más a fondo en la pregunta 5.

2. Para el método recursivo del cálculo del triángulo de Pascal, genera una gráfica en 3-D en donde el parámetro renglón se encontrará en el eje X, el parámetro columna se encontrará en el eje Y, el número de operaciones en el eje Z y haz un análisis de lo que sucede. ¿Cuál es el orden de complejidad? Justifica.





Podemos notar que en la versión iterativa nuestra función de crecimiento valga la redundancia crece menos rápido que la versión recursiva. Pero esto lo abordaremos más a fondo en la pregunta 5.

3. ¿Cuál es el máximo valor de n que pudiste calcular para Fibonacci sin que se alentara tu computadora? ¿cuánto se tardó? (Puede variar un poco de computadora a computadora (±3), así que no te esfuerces en encontrar un valor específico). **Respuesta:** Para el método iterativo ingresé n = un millón y se calculó al momento; en cambio con el método recursivo ingresé n = 50 y se tardó cerca de un minuto.

4. ¿Cuál es el máximo valor de ren que pudiste calcular para el triángulo de Pascal sin que se alentara tu computadora? ¿cuánto se tardó?
Respuesta: Para el método iterativo probé con ren = 10000 y col = 8000 y se calculó aproximadamente en tres segundos; en cambio con el método recursivo al probar con ren = 100 y col = 80 se calcula mucho más tiempo que el iterativo en el caso que expuse.

5. Justifica a partir del código ¿cuál es el orden de complejidad para cada uno de los métodos que programaste?

   Fibonacci Recursivo: Para obtener la complejidad de fibonacci recursivo sabemos que fibonacci (n) es igual a fibonacci (n-1) + fibonacci (n-2) y calcular la complejidad de fibonacci (n) sería n más la complejidad de fibonacci (n-1) +fibonacci (n-2) siguiendo esta fórmula de complejidad podemos ver que generamos un árbol en el que cada nodo representa una operación para calcular la suma de los siguientes dos números de fibonacci y podemos ver que la altura de este árbol es de tamaño n, por lo tanto el número de nodos del árbol es 2^n, por lo tanto la complejidad de fibonacci recursivo es 2^n.

   Fibonacci Iterativo:

```java
public int fibonacciIt(int n){
    contador++;
    if (n < 0) throw new IndexOutOfBoundsException();
    if (n == 0) return 0;
    if (n == 1) return 1;

    int fibonacci0 = 0;
    int fibonacci1 = 1;
    for (int i = 0; i < n-1 ; i++){
        int aux = fibonacci0;
        fibonacci0 = fibonacci1;
        fibonacci1 += aux;
        contador ++;
    }
    return fibonacci1;
}
```

   A partir del código calculemos utilizando la notación O grande para obtener la complejidad, obteniendo la complejidad de cada línea de código:

   Linea 1: contador++; → se hacen tres operaciones, una de asignación, otra de acceso y una suma; i ¡ n-1 esto es igual a una n. Pero al final esto es de orden constante, es decir, O(1).

   Línea 2 y 3: En ambas líneas se hacen tres operaciones, una de acceso, de comparación y de regreso. Pero al final esto es de orden constante, es decir, O(1).

   Línea 4 y 5: En ambas líneas se hacen dos operaciones de asignación y declaración. Pero al final esto es de orden constante, es decir, O(1).

Línea 8 a 10: Dentro de estas líneas se hace una iteración de 0 hasta n siempre, donde se realizan en 4 líneas operaciones de asignación e incremento, pero podemos deducir que al final si sumamos las complejidades de estas 4 líneas sería de orden constante, por lo que la complejidad del for al final es de orden líneal, es decir, O(n).

Línea 11: return fibonacci1; → se hace una operación de regreso. Pero al final esto es de orden constante, es decir, O(1).

Finalmente si sumamos las complejidades, la complejidad del for es la que predomina, por definición de la notación O grande. Por lo tanto, la complejidad de nuestro algoritmo es de orden lineal.

Pascal Recursivo:
Para obtener la complejidad de fibonacci recursivo sabemos que Pascal (x,y) es igual a Pascal (x-1)(y-1) + Pascal (x-1 (y) y calcular la complejidad de Pascal (x,y) sería (x,y) más la complejidad de Pascal (x-1,y-1) + Pascal (x-1)(y) siguiendo esta fórmula de complejidad podemos ver que generamos un árbol en el que cada nodo representa una operación para calcular la suma de los siguientes dos números de Pascal y podemos ver que la altura de este árbol es de tamaño (x,y), por lo tanto el número de nodos del árbol es 2^n, por lo tanto la complejidad de fibonacci recursivo es 2^n.

Pascal Iterativo:

```
public int tPascalIt(int ren, int col){
    contador = 1;
    if(col == 0 || ren == col) return 1;
    if (col < 0 || ren < 0 || col > ren) throw new IndexOutOfBoundsException();

    int [][] tPascal = new int[ren+1][col+1];
    tPascal [0][0] = 1;
    tPascal [1][0] = 1;
    tPascal [1][1] = 1;

    for (int i=0;i<ren+1;i++ ) {
        for (int j=0; j<col+1 && j <= i ;j++ ) {

            if(j == 0){
                tPascal[i][j] = 1;
            }else{
                if(i==j){
                    tPascal[i][j] = 1;

                }else{
                    contador ++;
                    tPascal[i][j] = tPascal[i-1][j-1] + tPascal[i-1][j];
                }
            }

        }
    }
    return tPascal[ren][col];
}
```

Siguiendo la misma lógica de la parte iterativa de fibonacci todas las llamadas antes del for tienen una complejidad de orden constante, es decir, O(1). A excepción de la línea 5 donde inicializamos una matriz (ren+1)(col+1), por lo que crear dicha matriz toma un orden O(n*m) donde n = ren+1 y m = col+1.
Ahora para la parte iterativo, como hay dos for anidados donde la j si se relaciona estrictamente con la i, donde la n de la j se multiplica con la n de la i, queda de la siguiente forma n*n y obteniendo la cuenta se tiene n^2 donde las constantes no se toman en cuenta debido a que la n^2 tiene mayor precedencia.
Por lo tanto la complejidad de nuestro algoritmo es de orden cuadrático, O(n^2).

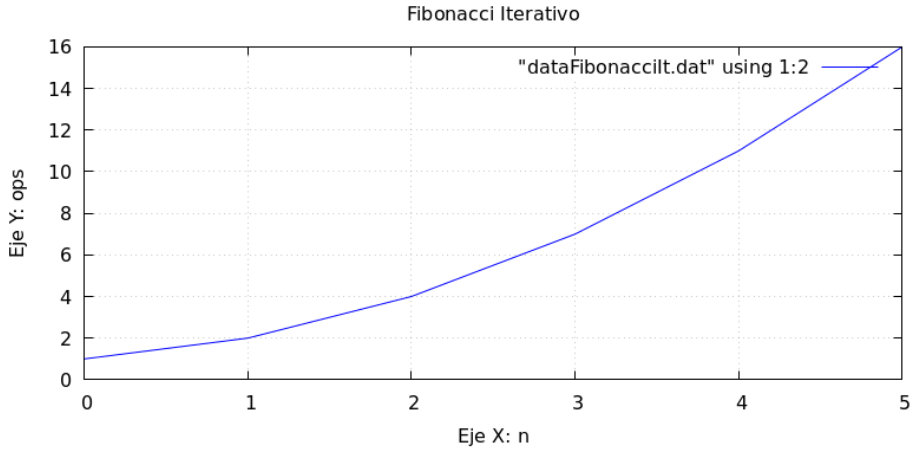


Figure 1: Gráfica generada por GNUPLOT para nuestro algoritmo iterativo de la función Fibonacci.

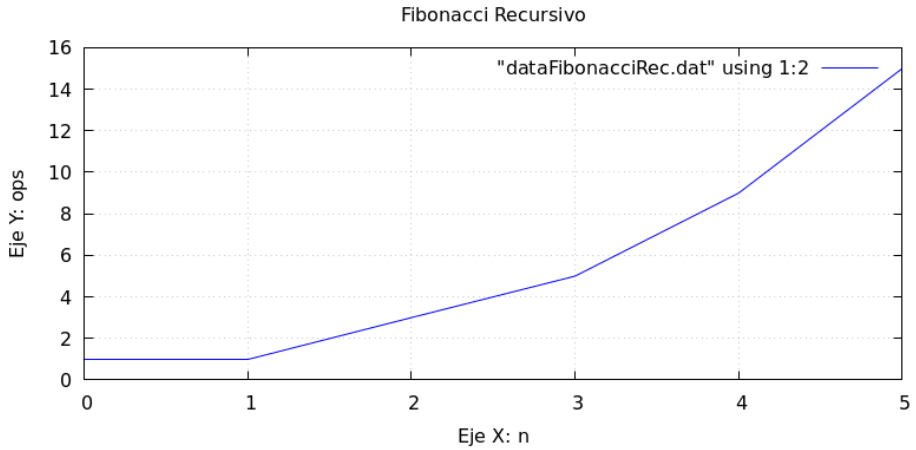


Figure 2: Gráfica generada por GNUPLOT para nuestro algoritmo recursivo de la función Fibonacci.

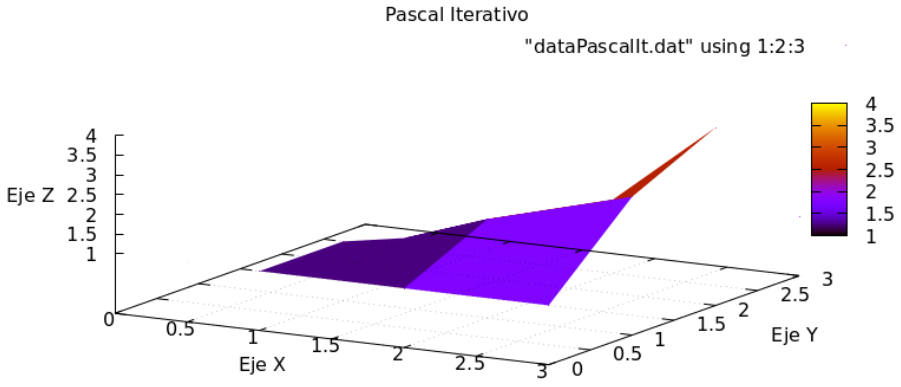


Figure 3: Gráfica generada por GNUPLOT para nuestro algoritmo iterativo de la función Pascal.

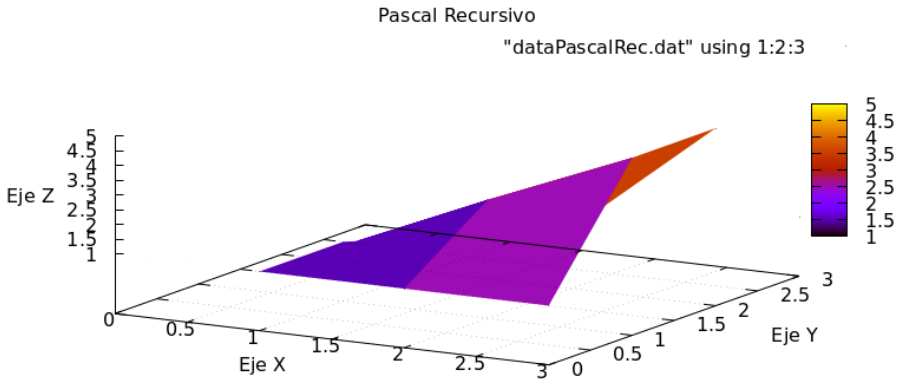


Figure 4: Gráfica generada por GNUPLOT para nuestro algoritmo recursivo de la función Pascal.