



FACULTAD DE
INGENIERÍA

UNIVERSIDAD DA VINCI
DE GUATEMALA

Universidad Da Vinci De Guatemala

Facultad de Ingeniería

Catedrático: Ing. Brandon Antony Chitay Coutiño

Curso: Estructuras de datos



**FACULTAD DE
INGENIERÍA**

UNIVERSIDAD DA VINCI
DE GUATEMALA

“Primer Examen Parcial”

Bryan Alexander Gonzalez Maddaleno

Número de carné: 202503096

Guatemala, febrero de 2026



Introducción

La notación Big-O es una métrica matemática utilizada en ciencias de la computación para describir el comportamiento de un algoritmo. No mide el tiempo exacto en segundos, sino cómo aumenta el número de operaciones a medida que el tamaño de la entrada (n) crece. Su propósito principal es predecir la eficiencia de un software antes de su implementación en sistemas de producción.



Desarrollo del trabajo

Descripción de Algoritmos

- **Factorial:** Calcula el producto de todos los números desde 1 hasta n. La versión iterativa solo es un ciclo que hace la multiplicación por el siguiente número y almacena el resultado. La recursiva se llama a sí misma reduciendo el problema en 1 cada vez hasta llegar al valor inicial (1).
- **Fibonacci:** Genera una secuencia donde cada número es la suma de los dos anteriores. La versión iterativa es simple y lineal solo haciendo las sumas hasta llegar a n. La recursiva, me demore un poco en entenderla, pero tiene el problema que trabaja de más por así decirlo. Repite cálculos intermedios de forma innecesaria.
- **Búsqueda Lineal:** Revisa uno por uno los elementos de una lista hasta encontrar el valor objetivo. El iterativo solo avanza posición por posición hasta encontrar el valor. El recursivo hace lo mismo, pero tiene la ventaja de poder dar un “índice” y saltarse cierta parte de la lista de ser necesario.
- **Ordenamiento Burbuja:** Un algoritmo de ordenamiento que compara elementos adyacentes y los intercambia si están en el orden incorrecto. La iterativa hace esto usando dos ciclos, el primero se asegura de dejar el valor más pequeño al inicio mientras que el segundo hace los intercambios. La recursiva hace el ciclo completo de la burbuja cada vez y luego se llama a si mismo para hacer un ciclo más pequeño dejando el valor más grande al final.



Fragmentos de Código

Fibonacci

```
public static long iterativo(int n) {  
    if (n < 0) throw new IllegalArgumentException(s: "n no puede ser negativo");  
  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
  
    long anterior = 0;  
    long actual = 1;  
  
    for (int i = 2; i <= n; i++) {  
        long siguiente = anterior + actual;  
        anterior = actual;  
        actual = siguiente;  
    }  
    return actual;  
}
```

```
public static long recursivo(int n) {  
    if (n < 0) throw new IllegalArgumentException(s: "n no puede ser negativo");  
    if (n > 30) throw new IllegalArgumentException(s: "Limita n ≤ 30 en versión recursiva (O(2^n))");  
  
    // --- CASOS BASE ---  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
  
    // --- DOS LLAMADAS RECURSIVAS = crecimiento exponencial ---  
    return recursivo(n - 1) + recursivo(n - 2);  
}
```



Factorial

```
public static long iterativo(int n) {  
    long resultado = 1;  
    for (int i = 1; i <= n; i++) {  
        resultado *= i;  
    }  
    return resultado;  
}  
  
public static long recursivo(int n) {  
    if (n <= 1) {  
        return 1; // Caso base  
    }  
    return n * recursivo(n - 1);  
}
```

Búsqueda Lineal

```
public static long iterativo(int[] arr, int busqueda) {  
    for (int i = 0; i < arr.length; i++) {  
        if (arr[i] == busqueda) {  
            return i; // Elemento encontrado  
        }  
    }  
    return -1; // No se encontró  
}  
  
public static long recursivo(int[] arr, int busqueda,int indice) {  
    // Caso base 1: El índice llegó al final (no encontrado)  
    if (indice >= arr.length) {  
        return -1;  
    }  
    // Caso base 2: Encontramos el valor  
    if (arr[indice] == busqueda) {  
        return indice;  
    }  
    // Llamada recursiva: Pasamos al siguiente índice  
    return recursivo(arr, busqueda, indice + 1);  
}
```



Ordenamiento Burbuja

```
public static void iterativo(int[] arr) {  
    int n = arr.length;  
    for (int i = 0; i < n - 1; i++) {  
        for (int j = 0; j < n - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                // Intercambio (swap)  
                int temp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = temp;  
            }  
        }  
    }  
}
```

```
public static void recursivo(int[] arr, int n) {  
    // Caso base: si el tamaño del arreglo es 1, ya está ordenado  
    if (n == 1) return;  
  
    // Una pasada completa de la "burbuja"  
    for (int i = 0; i < n - 1; i++) {  
        if (arr[i] > arr[i + 1]) {  
            int temp = arr[i];  
            arr[i] = arr[i + 1];  
            arr[i + 1] = temp;  
        }  
    }  
  
    // Llamada para el resto del arreglo (excluyendo el último elemento ya fijado)  
    recursivo(arr, n - 1);  
}
```



Generador de Arrays con valores aleatorios

```
/**  
 * Genera un arreglo de enteros aleatorios entre 0 y 10,000.  
 */  
  
public static int[] generarArregloAleatorio(int n) {  
    int[] arr = new int[n];  
    Random rand = new Random();  
  
    for (int i = 0; i < n; i++) {  
        arr[i] = rand.nextInt(bound: 10000); // Números de 0 a 9999  
    }  
    return arr;  
}
```

Main

La parte principal del código es bastante repetitiva ya que se hace lo mismo varias veces para poder correr todos los algoritmos en una sola ejecución.

Corremos el iterativo una vez y medimos el tiempo como referencia

```
// ---- ORDENAMIENTO BURBUJA ITERATIVO ----  
System.out.println("n ORDENAMIENTO BURBUJA ITERATIVO [O(n^2)]");  
Medidor.imprimirEncabezado();  
  
for (int n : ARRTAMANOS) {  
    final int fn = n;  
    int [] resultado=GeneradorDatos.generarArregloAleatorio(fn);  
    // Calcular resultado una vez (solo para mostrarlo)  
    OrdenamientoBurbuja.iterativo(resultado);  
  
    // Medir solo el algoritmo puro (sin I/O ni inicialización)  
    double tiempoMs = Medidor.medir(() -> OrdenamientoBurbuja.iterativo(GeneradorDatos.generarArregloAleatorio(fn)));  
  
    Medidor.imprimirFila(algoritmo: "ORDENAMIENTO BURBUJA", version: "Iterativo", n, tiempoMs);  
    csv.append(String.format(format: "ORDENAMIENTO BURBUJA,Iterativo,%d,%f%n", n, tiempoMs));  
}
```



Luego hacemos lo mismo con el recursivo:

```
// ---- ORDENAMIENTO BURBUJA RECURSIVO ----
System.out.println(x: "\n ORDENAMIENTO BURBUJA RECURSIVO [O(n^2)]");
Medidor.imprimirEncabezado();

for (int n : ARRTAMANOS) {
    final int fn = n;
    int [] resultado=GeneradorDatos.generarArregloAleatorio(fn);
    OrdenamientoBurbuja.recursivo(resultado,fn);
    double tiempoMs = Medidor.medir(() -> OrdenamientoBurbuja.recursivo(GeneradorDatos.generarArregloAleatorio(fn),fn));

    Medidor.imprimirFila(algoritmo: "ORDENAMIENTO BURBUJA", version: "Recursivo", n, tiempoMs);
    csv.append(String.format(format: "ORDENAMIENTO BURBUJA,Recursivo,%d,%6f%n", n, tiempoMs));
}
```

Mostramos la diferencia de ambos:

```
// ---- ANÁLISIS DE DIFERENCIA ----
System.out.println(x: "\n COMPARACION ITERATIVO vs RECURSIVO");
System.out.println("-".repeat(count: 60));
System.out.printf(format: "%-8s | %-14s | %-14s | %s%n",
| | ...args: "n", "Iterativo (ms)", "Recursivo (ms)", "Recursivo / Iterativo");
System.out.println("-".repeat(count: 60));
```

Y por último ya ejecutamos ambas versiones del algoritmo para todos los valores de n requeridos:

```
for (int n : ARRTAMANOS) {
    final int fn = n;

    double tIter = Medidor.medir(() -> OrdenamientoBurbuja.iterativo(GeneradorDatos.generarArregloAleatorio(fn)));
    double tRec = Medidor.medir(() -> OrdenamientoBurbuja.recursivo(GeneradorDatos.generarArregloAleatorio(fn),fn));
    double factor = (tIter > 0) ? tRec / tIter : 0;

    System.out.printf(format: "n=%-6d | %-14.6f | %-14.6f | %.1fx mas lento%n",
    | | n, tIter, tRec, factor);
}
```

Este código en el Main.java cambia para cada algoritmo ya que sus funciones son relativamente diferentes (en especial para la búsqueda y el ordenamiento) pero en esencia lo que se hace es lo mismo.

Tomamos ventaja de los métodos ya definidos de Medidor.java por el Ing. Brandon Chitay para medir el tiempo de cada algoritmo, y también del método exportarCSV en Main.java para llevarlo todo a un archivo y poder trasladar eso al Excel.



Tabla de Tiempos Medidos

HOJA 1 — Datos Crudos | Tiempos de Ejecucion (ms)

Algoritmo	Version	n	Ej. 1 (ms)	Ej. 2 (ms)	Ej. 3 (ms)	Ej. 4 (ms)	Ej. 5 (ms)	PROMEDIO (ms)
A1 - Factorial	Iterativo	5	0.00049	0.00035	0.00031	0.00038	0.00038	0.000382
A1 - Factorial	Iterativo	10	0.00032	0.00018	0.00019	0.0002	0.00023	0.000224
A1 - Factorial	Iterativo	15	0.00036	0.00025	0.00023	0.0003	0.00022	0.000272
A1 - Factorial	Iterativo	20	0.00029	0.00026	0.00026	0.00042	0.00027	0.0003
A1 - Factorial	Iterativo	25	0.00032	0.00031	0.00031	0.0005	0.00029	0.000346
A1 - Factorial	Iterativo	30	0.00057	0.0005	0.00042	0.00057	0.00081	0.000574
A1 - Factorial	Recursivo	5	0.00045	0.00038	0.00058	0.00066	0.00045	0.000504
A1 - Factorial	Recursivo	10	0.00045	0.00042	0.00045	0.00052	0.00069	0.000506
A1 - Factorial	Recursivo	15	0.00098	0.00166	0.00105	0.00118	0.00104	0.001182
A1 - Factorial	Recursivo	20	0.00015	0.00023	0.00018	0.00017	0.00026	0.000198
A1 - Factorial	Recursivo	25	0.00022	0.00017	0.00029	0.00017	0.00021	0.000212
A1 - Factorial	Recursivo	30	0.00022	0.00016	0.00024	0.00018	0.00024	0.000208

A2 - Fibonacci	Iterativo	5	0.00077	0.00055	0.00052	0.00051	0.00056	0.000582
A2 - Fibonacci	Iterativo	10	0.00028	0.00025	0.00029	0.00027	0.00024	0.000266
A2 - Fibonacci	Iterativo	15	0.00053	0.00031	0.00049	0.00035	0.00029	0.000394
A2 - Fibonacci	Iterativo	20	0.00038	0.00035	0.00054	0.00039	0.00034	0.0004
A2 - Fibonacci	Iterativo	25	0.00042	0.00041	0.00041	0.00058	0.00039	0.000442



FACULTAD DE INGENIERÍA

UNIVERSIDAD DA VINCI
DE GUATEMALA

A2 - Fibonacci	Iterativo	30	0.00061	0.00069	0.00059	0.00063	0.00098	0.0007
A2 - Fibonacci	Recursivo	5	0.00086	0.00135	0.00117	0.00087	0.0008	0.00101
A2 - Fibonacci	Recursivo	10	0.00605	0.00895	0.00614	0.00623	0.00612	0.006698
A2 - Fibonacci	Recursivo	15	0.00853	0.00884	0.00885	0.01116	0.01357	0.01019
A2 - Fibonacci	Recursivo	20	0.03841	0.04127	0.05974	0.03429	0.03152	0.041046
A2 - Fibonacci	Recursivo	25	0.32816	0.31592	0.31643	0.31987	0.31581	0.319238
A2 - Fibonacci	Recursivo	30	3.81257	3.77539	3.87319	3.88283	3.92972	3.85474

A3 - Busqueda Lineal	Iterativo	100	0.02217	0.02119	0.03172	0.02253	0.03856	0.027234
A3 - Busqueda Lineal	Iterativo	500	0.03104	0.0335	0.03389	0.02937	0.02425	0.03041
A3 - Busqueda Lineal	Iterativo	1000	0.04449	0.04468	0.05444	0.05928	0.04562	0.049702
A3 - Busqueda Lineal	Iterativo	5000	0.19281	0.21629	0.17663	0.17082	0.21789	0.194888
A3 - Busqueda Lineal	Iterativo	10000	0.18584	0.15157	0.15051	0.14211	0.13756	0.153518
A3 - Busqueda Lineal	Recursivo	100	0.00719	0.00898	0.00643	0.00634	0.00649	0.007086
A3 - Busqueda Lineal	Recursivo	500	0.01672	0.01769	0.01286	0.01297	0.01244	0.014536
A3 - Busqueda Lineal	Recursivo	1000	0.02343	0.02261	0.02203	0.02187	0.02112	0.022212
A3 - Busqueda Lineal	Recursivo	5000	0.09776	0.10344	0.14163	0.10302	0.10968	0.111106
A3 - Busqueda Lineal	Recursivo	10000	0.16171	0.1966	0.19814	0.226	0.18805	0.1941

A4 - Burbuja	Iterativo	100	0.1163	0.1205	0.11571	0.11582	0.11546	0.116758
A4 - Burbuja	Iterativo	500	0.47775	0.48685	0.58266	0.5008	0.65721	0.541054
A4 - Burbuja	Iterativo	1000	0.84695	0.88616	0.97295	0.7785	0.85403	0.867718
A4 - Burbuja	Iterativo	5000	22.8378	21.70362	22.18672	21.05366	23.80289	22.316938



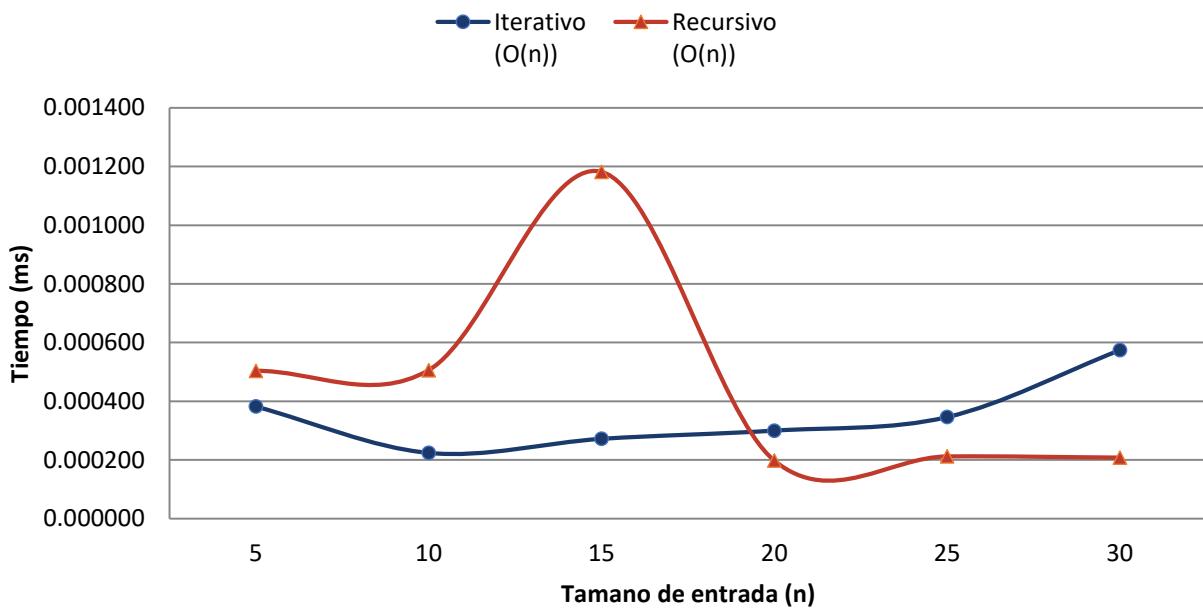
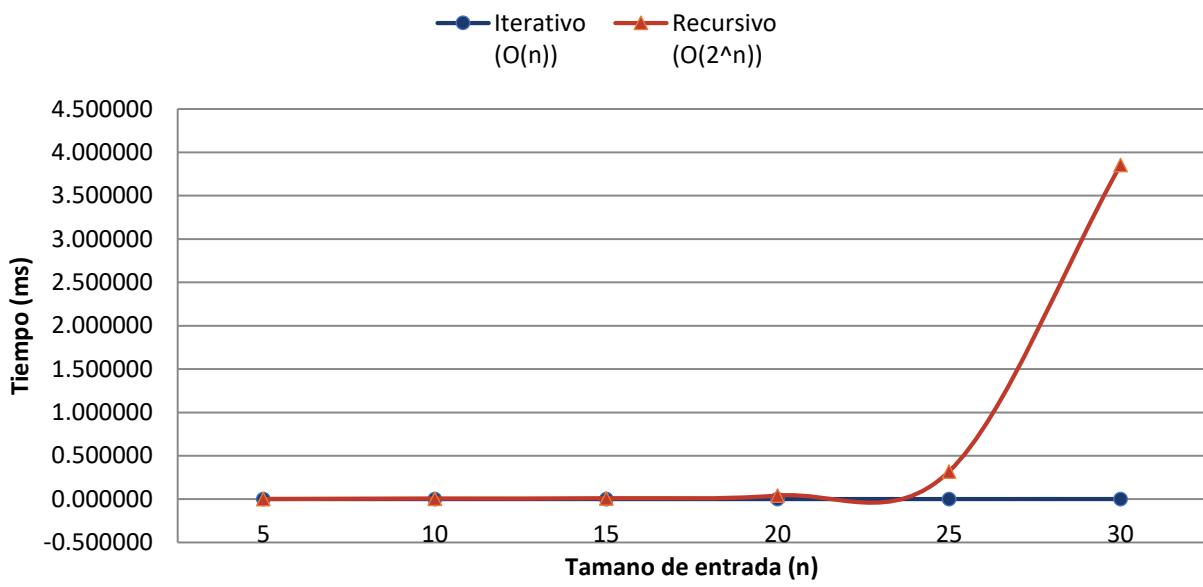
FACULTAD DE INGENIERÍA

UNIVERSIDAD DA VINCI
DE GUATEMALA

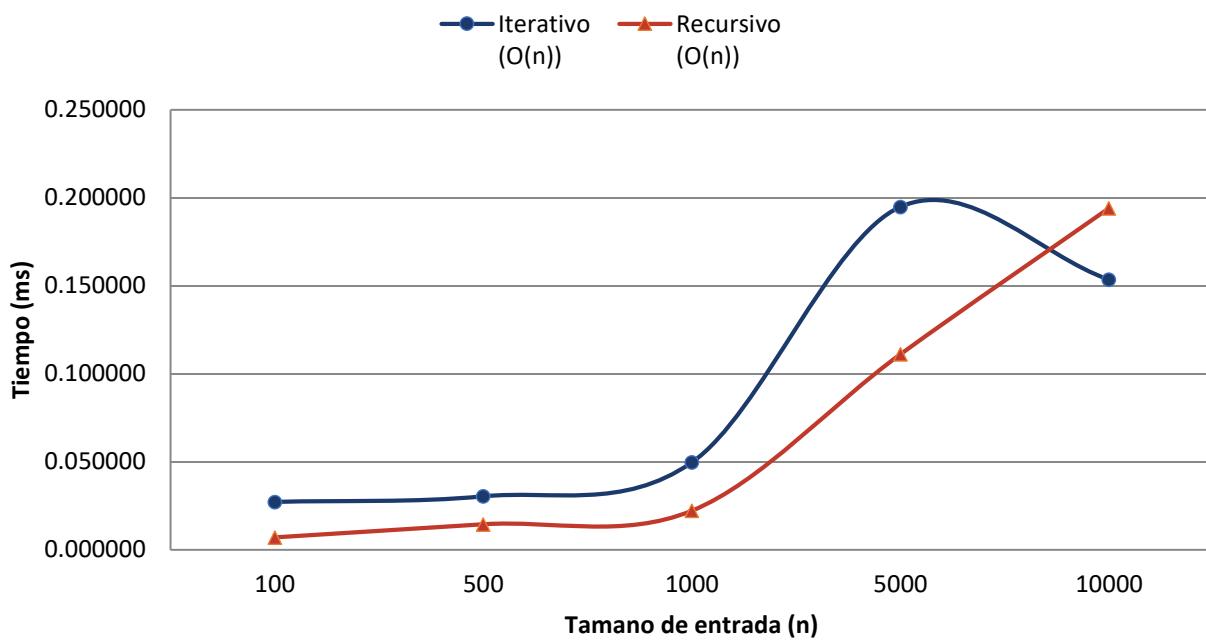
A4 - Burbuja	Iterativo	10000	109.07257	106.92634	108.13277	105.66249	108.83096	107.725026
A4 - Burbuja	Recursivo	100	0.08765	0.04201	0.0375	0.04553	0.03712	0.049962
A4 - Burbuja	Recursivo	500	0.21162	0.38699	0.39518	0.358	0.52258	0.374874
A4 - Burbuja	Recursivo	1000	0.72129	0.68233	0.73286	0.67013	0.64126	0.689574
A4 - Burbuja	Recursivo	5000	17.12645	16.79303	17.25706	17.10143	17.68062	17.191718
A4 - Burbuja	Recursivo	10000	87.5479	84.48316	84.7215	84.74015	85.20222	85.338986

Leyenda: Fondo AMARILLO = ingresa tu medicion real | Fondo VERDE = promedio automatico (no editar) | Fondo NARANJA = version recursiva

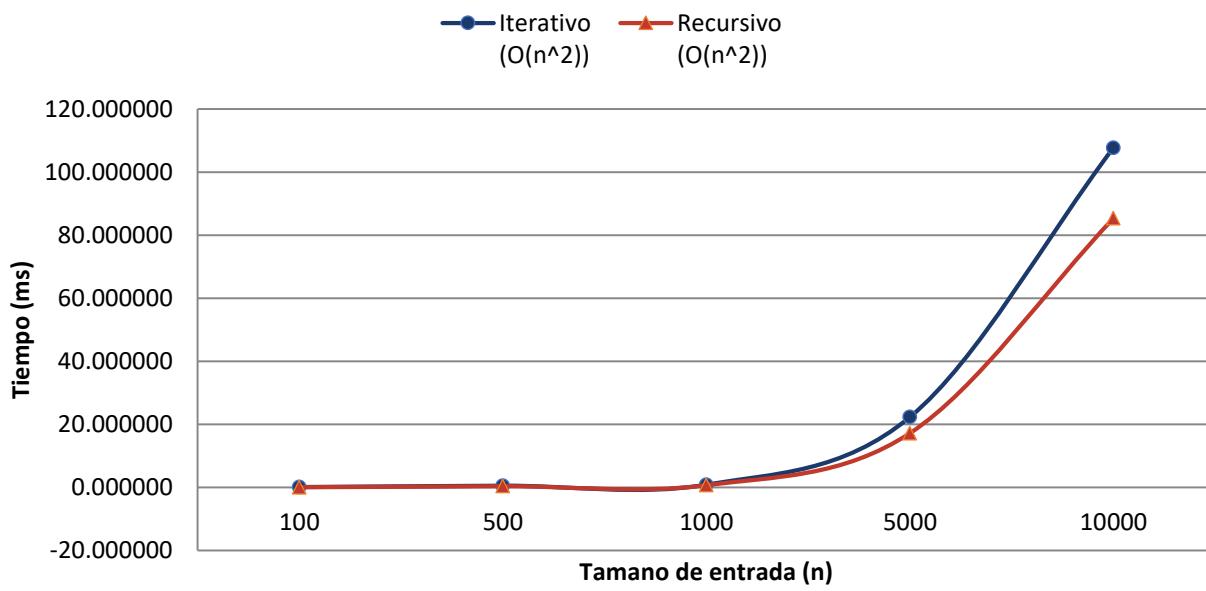
Gráficas

Factorial: Tiempo de Ejecucion vs. Tamano**Fibonacci: Tiempo de Ejecucion vs. Tamano**

Busqueda Lineal: Tiempo de Ejecucion vs. Tamano



Burbuja: Tiempo de Ejecucion vs. Tamano





Análisis y Conclusiones

- **Iterativo vs. Recursivo:** En algoritmos como Factorial y Búsqueda Lineal, la diferencia es mínima. Sin embargo, en Fibonacci, el recursivo es extremadamente ineficiente comparado con el iterativo por la redundancia o doble trabajo de cálculos.
- **Justificación de Big-O:** Los datos experimentales confirman la teoría. Por ejemplo, en el algoritmo de Burbuja, al duplicar n de 5,000 a 10,000, el tiempo aumentó aproximadamente 4.8 veces, lo que confirma un comportamiento Cuadrático $O(n^2)$.
- **Impacto del Sistema:** Se observaron picos de tiempo (como en Factorial Recursivo $n=15$) que no pertenecen a la complejidad del algoritmo, sino al "ruido" del procesador o la gestión de memoria del Sistema Operativo.



Bibliografía (Normas APA)

- *Arrays (Java SE 25 & JDK 25)*. (2026, 20 enero).
<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/Arrays.html>
- *Arrays (The Java™ Tutorials > Learning the Java Language > Language Basics)*. (s. f.). <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>
- *Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell*. (s. f.).
<https://www.bigocheatsheet.com/>
- *Defining Methods (The Java™ Tutorials > Learning the Java Language > Classes and Objects)*. (s. f.). <https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>
- Olawale, J. (2024, 7 noviembre). *Big o cheat sheet – Time Complexity chart*. freeCodeCamp.org. <https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/>