



**UNIVERSIDAD
DA VINCI
DE GUATEMALA**

TEMA:

PARCIAL 1 ESTRUCTURA DE DATOS



NOMBRE;

Isaías Mejía Morente

CARNET:

202503928

Universidad:

Universidad Da Vinci de Guatemala

Facultad:

Facultad de Ingeniería

Tutor:

Ing. Brandom C

Fecha de entrega:

30 de enero del 2026

INDICE

1. Introducción	1
2. ANÁLISIS COMPARATIVO DE ALGORITMOS ITERATIVOS Y RECURSIVOS.....	2
Hipótesis:.....	2
2.1. FACTORIAL.....	2
2.1.1. Definición	2
2.1.2. FACTORIAL ITERATIVO	2
Versión Iterativa	2
Complejidad:.....	2
2.1.3. FACTORIAL RECURSIVO	3
Versión Recursiva	3
Complejidad:.....	3
conclusión	4
2.2 SERIE DE FIBONACCI.....	6
Definición.....	6
Versión Iterativa	6
Complejidad:.....	6
Análisis	6
Versión Recursiva Simple.....	7
Complejidad:.....	7
Análisis:	7
2.3 BUSQUEDA LINEAL.....	10
Definición	10
Versión Iterativa.....	10
Complejidad:	10
Análisis:	10
Versión Recursiva	10
2.4 ORDENAMIENTO DE BURBUJA	13
Definición.....	13
Versión Iterativa	13

Complejidad:.....	13
Análisis:	14
2.5. Bibliografía	18



UNIVERSIDAD
DA VINCI
DE GUATEMALA

1. Introducción

En el desarrollo de software, los algoritmos pueden implementarse principalmente de dos formas: **iterativa** y **recursiva**. Ambas estrategias permiten resolver problemas computacionales, pero presentan diferencias significativas en términos de eficiencia temporal y uso de memoria.

El presente documento tiene como objetivo analizar y comparar las versiones iterativas y recursivas de los siguientes algoritmos fundamentales:

- Factorial
- Serie de Fibonacci
- Búsqueda Lineal
- Ordenamiento de Burbuja

Se evaluará su definición, funcionamiento y eficiencia computacional mediante el análisis de complejidad temporal y espacial.



UNIVERSIDAD
DA VINCI
DE GUATEMALA

2. ANÁLISIS COMPARATIVO DE ALGORITMOS ITERATIVOS Y RECURSIVOS

Hipótesis:

Algoritmos a Implementar

Implementa cada uno de los siguientes 4 algoritmos en sus versiones ITERATIVA y RECURSIVA (8 implementaciones en total). La columna 'Big-O esperado' es una guía: tu trabajo es CONFIRMAR esa notación experimentalmente con las graficas.

#	Algoritmo	Descripción	Iterativo	Recursivo
A1	Factorial	Calcula $n!$. Caso base: $0! = 1$. Limita n a 20 máximo.	$O(n)$	$O(n)$
A2	Serie de Fibonacci	Calcula el n -ésimo término. Limita n a 30 en la versión recursiva.	$O(n)$	$O(2^n)$
A3	Busqueda Lineal	Busca un valor en un arreglo. Retorna el índice o -1 si no existe.	$O(n)$	$O(n)$
A4	Ordenamiento Burbuja	Ordena un arreglo de n enteros de menor a mayor usando el método burbuja.	$O(n^2)$	$O(n^2)$

2.1. FACTORIAL

2.1.1. Definición

La factorial de un número entero positivo n se define como el producto de todos los números enteros desde 1 hasta n .

$$n! = n \times (n - 1) \times (n - 2) \cdots \times 1$$

2.1.2. FACTORIAL ITERATIVO

Versión Iterativa

Se implementa utilizando una estructura repetitiva (for o while), multiplicando progresivamente los valores hasta alcanzar n .

Complejidad:

- Tiempo: $O(n)$

- Espacio: $O(1)$

```
package algorithms;

public class Factorial {

    // Factorial iterativo  $O(n)$ 
    public static long factorialIterativo(int n) {
        long resultado = 1;
        for (int i = 2; i <= n; i++) {
            resultado *= i;
        }
        return resultado;
    }
}
```

2.1.3. FACTORIAL RECURSIVO

Versión Recursiva

Se basa en la definición matemática:

$$n! = n \times (n - 1)!$$

Con caso base:

$$0! = 1$$

Complejidad:

- Tiempo: $O(n)$
- Espacio: $O(n)$

```
• // Factorial recursivo  $O(n)$ 
• public static long factorialRecursivo(int n) {
•     if (n <= 1) {
•         return 1;
•     }
•     return n * factorialRecursivo(n - 1);
• }
• }
```

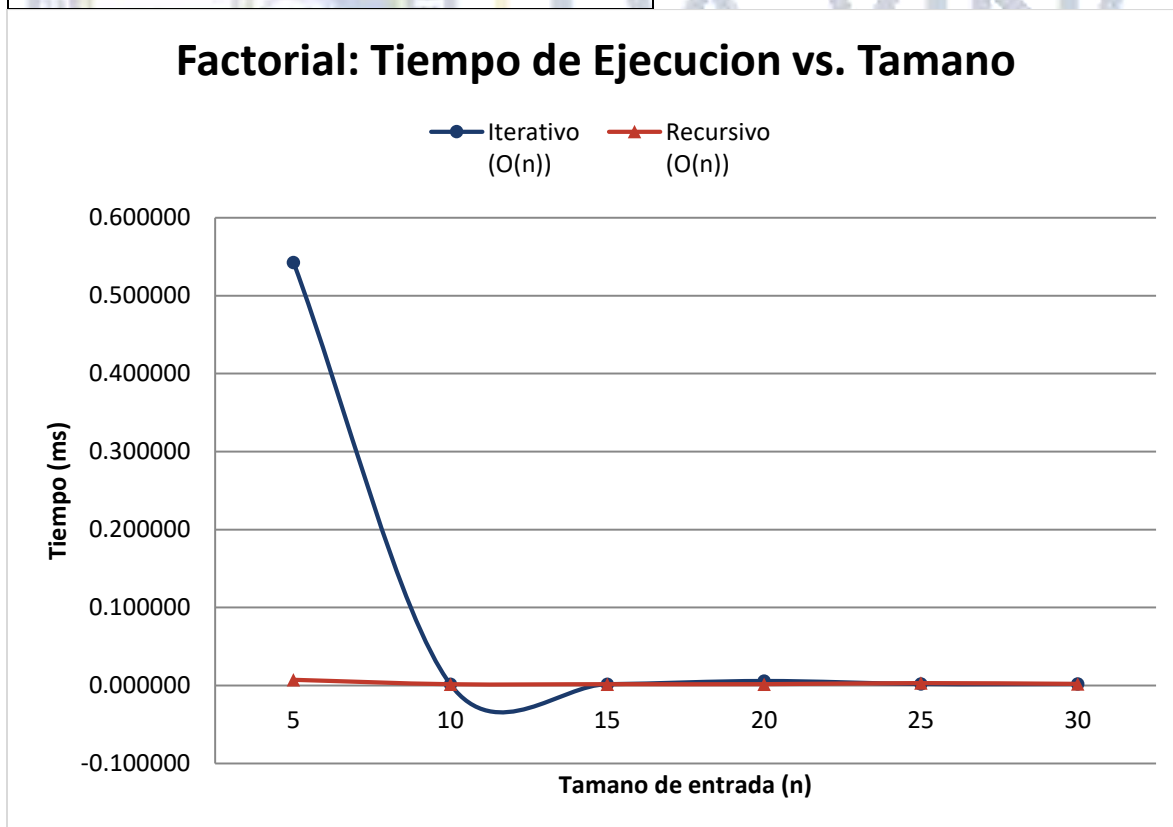
HOJA 1 — Datos Crudos Tiempos de Ejecucion (ms)								
Algoritmo	Version	n	Ej. 1 (ms)	Ej. 2 (ms)	Ej. 3 (ms)	Ej. 4 (ms)	Ej. 5 (ms)	PROMEDIO (ms)
A1 - Factorial	Iterativo	5	0.5505	0.4865	0.6916	0.4671	0.5177	0.54268
A1 - Factorial	Iterativo	10	0.0013	0.0012	0.0019	0.0013	0.002	0.00154
A1 - Factorial	Iterativo	15	0.0017	0.0015	0.0012	0.0025	0.0016	0.0017
A1 - Factorial	Iterativo	20	0.0018	0.0089	0.0013	0.015	0.0021	0.00582
A1 - Factorial	Iterativo	25	0.0017	0.0023	0.0017	0.0015	0.0025	0.00194
A1 - Factorial	Iterativo	30	0.0016	0.0018	0.0018	0.0024	0.0017	0.00186
A1 - Factorial	Rekursivo	5	0.0045	0.0053	0.0044	0.0057	0.0168	0.00734
A1 - Factorial	Rekursivo	10	0.0017	0.0016	0.0018	0.0015	0.0018	0.00168
A1 - Factorial	Rekursivo	15	0.0017	0.0016	0.0014	0.0018	0.0021	0.00172
A1 - Factorial	Rekursivo	20	0.0014	0.0022	0.0018	0.0017	0.0017	0.00176
A1 - Factorial	Rekursivo	25	0.0074	0.0019	0.0017	0.0023	0.0023	0.00312
A1 - Factorial	Rekursivo	30	0.0021	0.002	0.0018	0.0025	0.002	0.00208

conclusión

La versión iterativa es eficiente en memoria, ya que no utiliza llamadas adicionales en la pila del sistema.

Factorial recursivo, aunque mantiene la misma complejidad temporal que la versión iterativa, utiliza mayor memoria debido al almacenamiento de llamadas en la pila.

HOJA — A1 - Factorial Iterativo vs. Recursivo		
Big-O Iterativo:	$O(n)$	Big-O Recursivo: $O(n)$
n	Iterativo ($O(n)$)	Recursivo ($O(n)$)
5	0.542680	0.007340
10	0.001540	0.001680
15	0.001700	0.001720
20	0.005820	0.001760
25	0.001940	0.003120
30	0.001860	0.002080
Observacion: Ambas versiones son $O(n)$: el tiempo crece linealmente con n .		



2.2 SERIE DE FIBONACCI

Definición

La sucesión de Fibonacci se define mediante la siguiente relación:

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

Con condiciones iniciales:

$$F(0) = 0, F(1) = 1$$

Versión Iterativa

Se implementa utilizando variables auxiliares que almacenan los valores previos dentro de un ciclo.

Complejidad:

- Tiempo: $O(n)$
- Espacio: $O(1)$

Análisis:

Es considerablemente más eficiente, ya que evita cálculos repetidos.

```
public class Fibonacci {  
  
    // Fibonacci iterativo O(n)  
    public static long fibonacciIterativo(int n) {  
        if (n <= 1) {  
            return n;  
        }  
        long a = 0, b = 1;  
        for (int i = 2; i <= n; i++) {  
            long temp = a + b;  
            a = b;  
            b = temp;  
        }  
        return b;  
    }  
}
```

Versión Recursiva Simple

Aplica directamente la fórmula matemática realizando múltiples llamadas recursivas.

Complejidad:

- Tiempo: $O(2^n)$
- Espacio: $O(n)$

Análisis:

Es altamente ineficiente debido a la repetición innecesaria de cálculos. Su uso se recomienda únicamente con fines académicos.

```
// Fibonacci recursivo  $O(2^n)$  – sin memoización
public static long fibonacciRecursivo(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacciRecursivo(n - 1) + fibonacciRecursivo(n - 2);
}
```

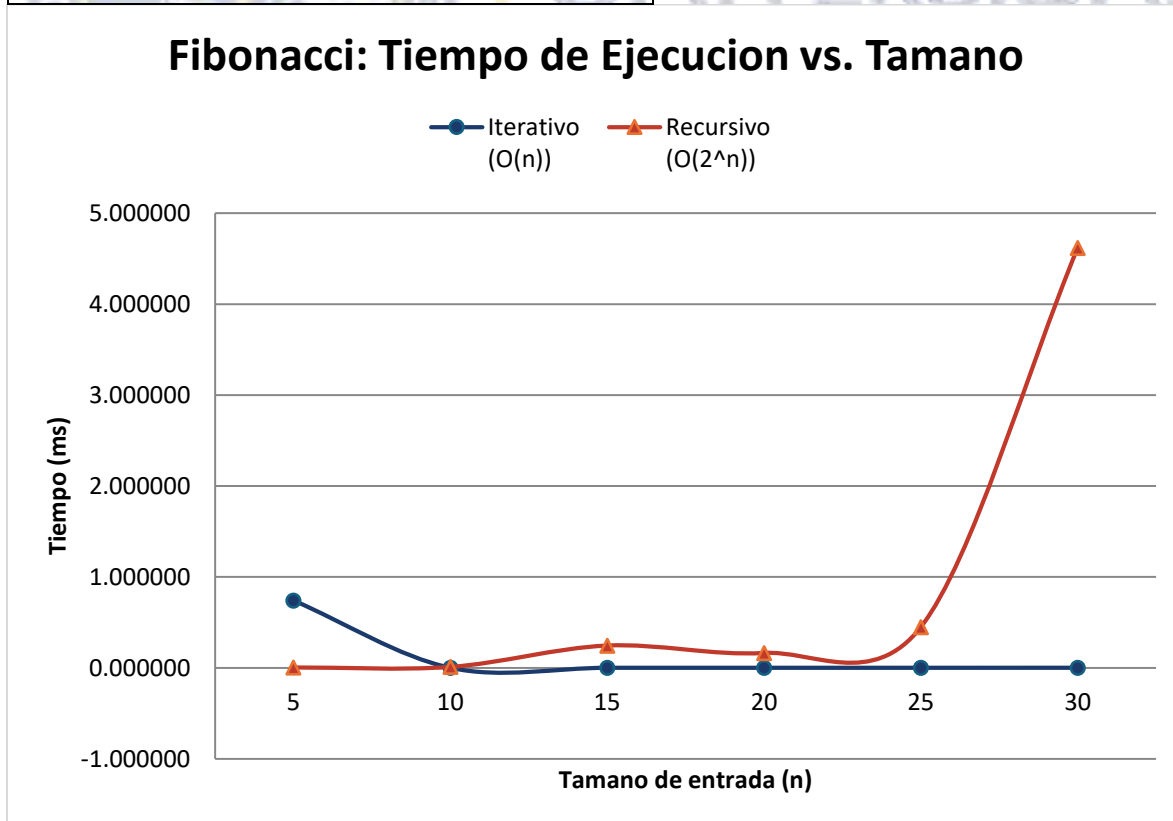


HOJA 1 — Datos Crudos Tiempos de Ejecucion (ms)								
Algoritmo	Version	n	Ej. 1 (ms)	Ej. 2 (ms)	Ej. 3 (ms)	Ej. 4 (ms)	Ej. 5 (ms)	PROMEDIO (ms)
A2 - Fibonacci	Iterativo	5	0.5908	1.0298	0.7502	0.6851	0.6507	0.74132
A2 - Fibonacci	Iterativo	10	0.0016	0.0013	0.0013	0.0015	0.0018	0.0015
A2 - Fibonacci	Iterativo	15	0.0014	0.0026	0.0018	0.0014	0.0013	0.0017
A2 - Fibonacci	Iterativo	20	0.0022	0.0018	0.0015	0.0016	0.0018	0.00178
A2 - Fibonacci	Iterativo	25	0.0018	0.0019	0.0018	0.0016	0.0016	0.00174
A2 - Fibonacci	Iterativo	30	0.0018	0.0029	0.0022	0.0017	0.0019	0.0021
A2 - Fibonacci	Recursivo	5	0.0048	0.0048	0.0059	0.0044	0.0045	0.00488
A2 - Fibonacci	Recursivo	10	0.0083	0.0202	0.0111	0.0078	0.0091	0.0113
A2 - Fibonacci	Recursivo	15	0.1691	0.1463	0.1558	0.1698	0.5903	0.24626
A2 - Fibonacci	Recursivo	20	0.181	0.1244	0.1921	0.1519	0.1747	0.16482
A2 - Fibonacci	Recursivo	25	0.4074	0.4409	0.4664	0.4573	0.475	0.4494
A2 - Fibonacci	Recursivo	30	5.6609	4.5276	4.056	4.1078	4.7404	4.61854



UNIVERSIDAD
DA VINCI
DE GUATEMALA

HOJA — A2 - Fibonacci Iterativo vs. Recursivo		
Big-O Iterativo:	$O(n)$	Big-O Recursivo: $O(2^n)$
n	Iterativo ($O(n)$)	Recursivo ($O(2^n)$)
5	0.741320	0.004880
10	0.001500	0.011300
15	0.001700	0.246260
20	0.001780	0.164820
25	0.001740	0.449400
30	0.002100	4.618540
Observacion: Recursivo crece exponencialmente: duplicar n casi cuadriplica el tiempo.		



2.3 BUSQUEDA LINEAL

Definición

La búsqueda lineal consiste en recorrer secuencialmente un arreglo hasta encontrar el elemento deseado o llegar al final.

Versión Iterativa

Se recorre el arreglo utilizando un ciclo comparando cada elemento.

Complejidad:

- Mejor caso: $O(1)$
- Peor caso: $O(n)$
- Espacio: $O(1)$

Análisis:

Es simple y eficiente en memoria.

```
public class BusquedaLineal {  
  
    // Búsqueda lineal iterativa  $O(n)$   
    public static int busquedaLinealIterativa(int[] arr, int objetivo) {  
        for (int i = 0; i < arr.length; i++) {  
            if (arr[i] == objetivo) {  
                return i;  
            }  
        }  
        return -1;  
    }  
}
```

Versión Recursiva

La función compara un elemento y luego se llama a sí misma con el resto del arreglo.

Complejidad:

- Mejor caso: $O(1)$

- Peor caso: $O(n)$
- Espacio: $O(n)$

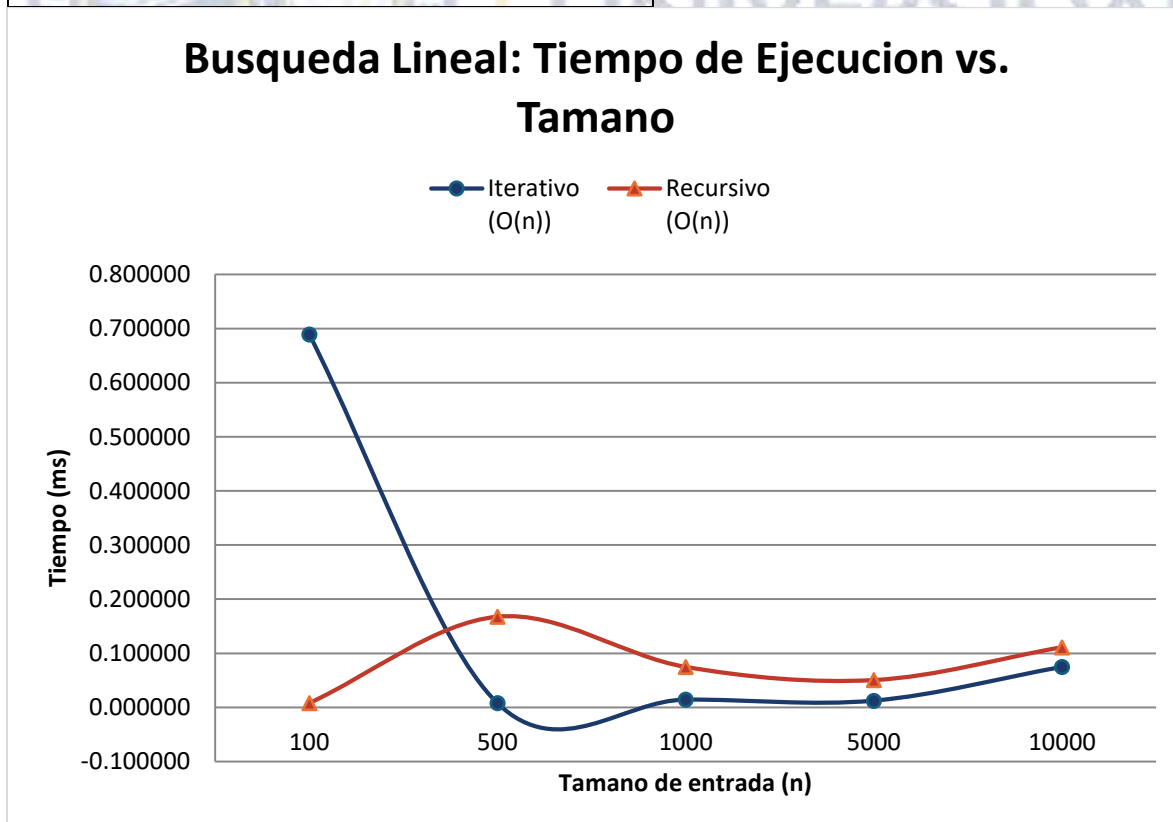
Análisis:

No mejora el rendimiento y aumenta el consumo de memoria.

```
// Búsqueda lineal recursiva  $O(n)$ 
public static int busquedaLinealRecursiva(int[] arr, int objetivo, int
indice) {
    if (indice >= arr.length) {
        return -1;
    }
    if (arr[indice] == objetivo) {
        return indice;
    }
    return busquedaLinealRecursiva(arr, objetivo, indice + 1);
}
```

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)
A3 - Búsqueda Lineal	Iterativo	100	0.532	0.6202	0.9774	0.6246	0.6931	0.68946
A3 - Búsqueda Lineal	Iterativo	500	0.0074	0.0077	0.0087	0.008	0.0079	0.00794
A3 - Búsqueda Lineal	Iterativo	1000	0.0104	0.0183	0.0155	0.0131	0.0152	0.0145
A3 - Búsqueda Lineal	Iterativo	5000	0.0048	0.0202	0.0248	0.0074	0.0056	0.01256
A3 - Búsqueda Lineal	Iterativo	10000	0.0045	0.0593	0.0026	0.0165	0.292	0.07498
A3 - Búsqueda Lineal	Recursivo	100	0.008	0.0091	0.0081	0.0089	0.0061	0.00804
A3 - Búsqueda Lineal	Recursivo	500	0.1658	0.1213	0.1909	0.3493	0.0109	0.16764
A3 - Búsqueda Lineal	Recursivo	1000	0.043	0.015	0.0245	0.0207	0.2713	0.0749
A3 - Búsqueda Lineal	Recursivo	5000	0.0011	0.0769	0.0056	0.0027	0.1666	0.05058
A3 - Búsqueda Lineal	Recursivo	10000	0.0028	0.0129	0.2446	0.2308	0.0654	0.1113

HOJA — A3 - Búsqueda Lineal Iterativo vs. Recursivo		
Big-O Iterativo:	$O(n)$	Big-O Recursivo: $O(n)$
n	Iterativo ($O(n)$)	Recursivo ($O(n)$)
100	0.689460	0.008040
500	0.007940	0.167640
1000	0.014500	0.074900
5000	0.012560	0.050580
10000	0.074980	0.111300
Observacion: Ambas $O(n)$: al multiplicar n x10, el tiempo tambien x10 aprox.		



2.4 ORDENAMIENTO DE BURBUJA

Definición

Algoritmo que compara elementos adyacentes e intercambia sus posiciones si están en orden incorrecto, repitiendo el proceso hasta ordenar completamente el arreglo.

Versión Iterativa

Utiliza dos ciclos anidados para realizar comparaciones e intercambios.

Complejidad:

- Mejor caso (optimizado): $O(n)$
- Promedio: $O(n^2)$
- Peor caso: $O(n^2)$
- Espacio: $O(1)$

Análisis:

Es un algoritmo sencillo, pero ineficiente para grandes volúmenes de datos.

```
// Ordenamiento burbuja iterativo  $O(n^2)$ 
public static void burbujaIterativo(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

Versión Recursiva

Realiza una pasada del algoritmo y luego se llama a sí misma para ordenar el resto del arreglo.

Complejidad:

- Tiempo: $O(n^2)$
- Espacio: $O(n)$

Análisis:

No presenta mejoras respecto a la versión iterativa y consume más memoria.

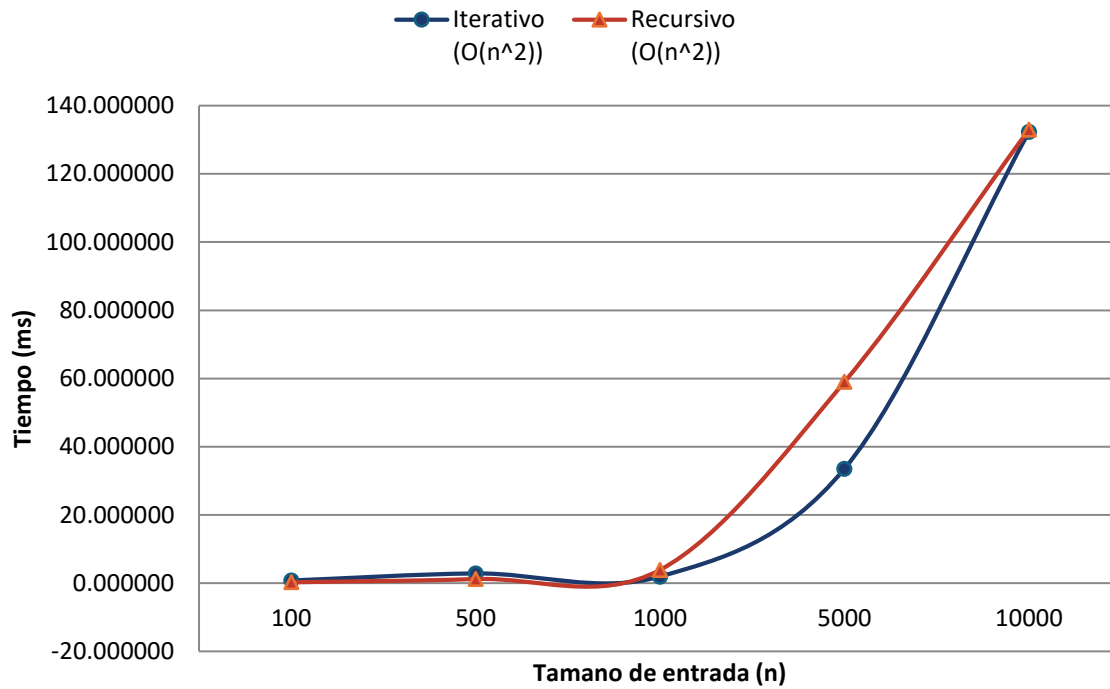
```
// Ordenamiento burbuja recursivo  $O(n^2)$ 
public static void burbujaRecursivo(int[] arr, int n) {
    if (n <= 1) {
        return;
    }
    // Una pasada: burbujear el mayor al final
    for (int j = 0; j < n - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
    // Llamada recursiva con n-1
    burbujaRecursivo(arr, n - 1);
}
```



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)
A4 - Burbuja	Iterativo	100	0.755	0.7074	0.9181	0.7192	0.8121	0.78236
A4 - Burbuja	Iterativo	500	2.6563	3.7433	2.6028	2.7061	2.561	2.8539
A4 - Burbuja	Iterativo	1000	2.6411	1.4109	1.4717	1.5482	2.6638	1.94714
A4 - Burbuja	Iterativo	5000	29.2969	33.9338	36.8948	35.8027	31.8398	33.5536
A4 - Burbuja	Iterativo	10000	129.4178	129.1782	129.6985	136.5942	136.522	132.28214
A4 - Burbuja	Recursivo	100	0.4376	0.3868	0.1741	0.1552	0.1548	0.2617
A4 - Burbuja	Recursivo	500	1.5326	1.2479	1.2471	1.0121	1.0148	1.2109
A4 - Burbuja	Recursivo	1000	6.3521	3.5195	2.5408	3.6099	3.459	3.89626
A4 - Burbuja	Recursivo	5000	74.7009	58.8283	51.384	60.2684	49.8685	59.01002
A4 - Burbuja	Recursivo	10000	134.4186	129.9765	130.2533	134.0562	136.1789	132.9767

HOJA — A4 - Burbuja Iterativo vs. Recursivo		
Big-O Iterativo:	O(n ²)	Big-O Recursivo: O(n ²)
n	Iterativo (O(n ²))	Recursivo (O(n ²))
100	0.782360	0.261700
500	2.853900	1.210900
1000	1.947140	3.896260
5000	33.553600	59.010020
10000	132.282140	132.976700
Observacion: Ambas O(n ²): al x10 n, el tiempo aumenta aprox x100.		

Burbuja: Tiempo de Ejecucion vs. Tamano



DA VINCI
DE GUATEMALA

HOJA 6 — Conclusiones | Big-O Experimental vs. Teorico

Algoritmo	Version	Big-O Experimental (de tu grafica)	Big-O Teorico	Coincide?	Justificacion breve
A1 - Factorial	Iterativo	O(n)	O(n)	Si / No	La versión iterativa es eficiente en memoria, ya que no utiliza llamadas adicionales en la pila del sistema.
A1 - Factorial	Recursivo	O(n)	O(n)	Si / No	Factorial recursivo, aunque mantiene la misma complejidad temporal que la versión iterativa, utiliza mayor memoria debido al almacenamiento de llamadas en la pila.
A2 - Fibonacci	Iterativo	O(n)	O(n)	Si / No	Es considerablemente más eficiente, ya que evita cálculos repetidos.
A2 - Fibonacci	Recursivo	O(2 ⁿ)	O(2 ⁿ)	Si / No	Es altamente ineficiente debido a la repetición innecesaria de cálculos. Su uso se recomienda únicamente con fines académicos.
A3 - Búsqueda Lineal	Iterativo	O(n)	O(n)	Si / No	Es simple y eficiente en memoria.
A3 - Búsqueda Lineal	Recursivo	O(n)	O(n)	Si / No	No mejora el rendimiento y aumenta el consumo de memoria.
A4 - Burbuja	Iterativo	O(n ²)	O(n ²)	Si / No	Es un algoritmo sencillo, pero ineficiente para grandes volúmenes de datos.
A4 - Burbuja	Recursivo	O(n ²)	O(n ²)	Si / No	No presenta mejoras respecto a la versión iterativa y consume más memoria.

Guía para interpretar tus graficas:

Linea recta en grafica	→ Big-O es probablemente O(1) o O(n)
Curva suave ascendente	→ Big-O es probablemente O(n) o O(log n)
Curva que sube rapido	→ Big-O es probablemente O(n ²)
Curva que explota (vertical)	→ Big-O es probablemente O(2 ⁿ)

2.5. E grafía

BitBoss. (2022, 30 noviembre). *Complejidad Algorítmica sin llorar - Notación Big O* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=UPDjjuz1Hkw>

Programación Desde Cero. (2020, 8 enero). *Recursividad - Qué son las funciones recursivas - Diferencia: algoritmos recursivos e iterativos* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=0NBPd81uhJE>

JDLC - Programación y Software. (2022, 22 noviembre). *Algoritmo factorial resuelto con FOR | En JAVA* 🍰 [Vídeo]. YouTube. <https://www.youtube.com/watch?v=8xDLdcxTmrA>

Xaca Rana. (2017, 15 julio). 16 Curso Programación JAVA Solución algoritmo: serie de fibonacci #serie #fibonacci #java #programa [Vídeo]. YouTube. https://www.youtube.com/watch?v=_RWKeMs_8og

CodigoMentor. (2015, 26 diciembre). *Curso de Java [Tutorial Java Básico] 30 - Búsqueda lineal* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=X1ozAfTwNP4>

CodigoMentor. (2015b, diciembre 26). *Curso de Java [Tutorial Java Básico] 30 - Búsqueda lineal* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=X1ozAfTwNP4>

codigofacilito. (2012, 24 marzo). *Ordenamiento de Burbuja (Bubble Sort) en Java* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=NVuQWFYIXm8>