

**FACULTA DE CIENCIAS E INGENIERÍAS FÍSICAS Y FORMALES.**

**Escuela Profesional de Ingeniería de Sistemas.**

**Curso:**

**ALGORITMIA Y ESTRUCTURA DE DATOS.**

**COMPLEJIDAD ALGORÍTMICA**

**ACTIVIDADES – EJERCICIOS – TAREA.**

**Diaz Huacasi Sebastian Salvador**

**Ugarte Rodriguez Eduardo**

**Docente**

**MBA Mg. Ing. Rene Alonso Nieto Valencia.**

**Propuesta Trabajo**

**Versión 1.0**

**29 de marzo del 2025**

**Historial de Revisiones**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fecha de Elaboración** | **Versión** | **Elaborado por:** | **Descripción** | **Revisado por:** | **Fecha de Revisión** |
| 15/05/2024 | 1.0 |  | Versión preliminar como propuesta. | Rene Alonso Nieto Valencia | 15/05/2024 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# **ÍNDICE**

[**ÍNDICE 3**](#_w5olaivb11xr)

[**1. ACTIVIDAD 01. 4**](#_kn83bvuex8yg)

[**1.1. ENUNCIADO. 4**](#_yywihnfcjtv8)

[**1.2. ENLACE GITHUB/GITLAB. 4**](#_99j894co95sy)

[**1.3. RESULTADOS. 4**](#_yceu4gt5d0ac)

[2. EXPLICACIÓN. 4](#_jrysler3s5n8)

[**3. EJERCICIO 1. 6**](#_1nufq9ish1lp)

[**3.1. ENUNCIADO. 6**](#_v8b9xtxu6kl2)

[**3.2. RESPUESTA. 8**](#_rhlvvhqujk7z)

# **ACTIVIDAD 01.**

## **ENUNCIADO.**

* Analice los siguientes procedimientos, funciones y obtenga su orden computacional correspondiente aplicando las reglas asintóticas y las pautas para el análisis de algoritmos.

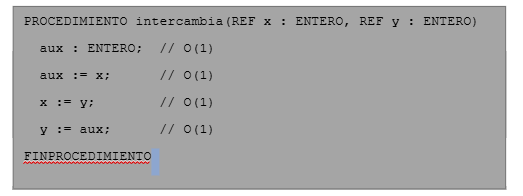
## **ENLACE GITHUB/GITLAB.**

* <https://github.com/SebastianDiazHuacasiCR7Siu/AED_Grupo_8.git>

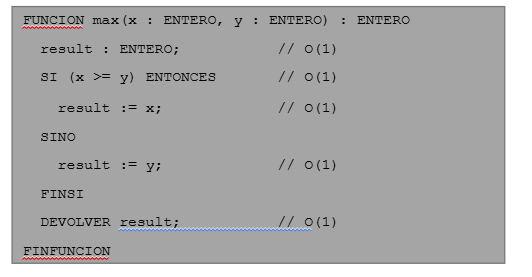
## **RESULTADOS.**

## **EXPLICACIÓN.**

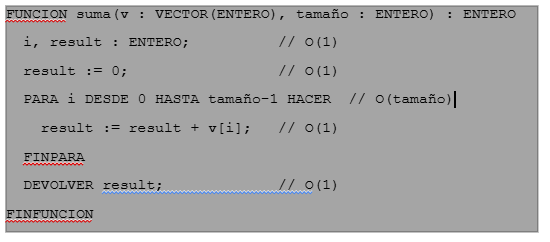
* Explicación clara y concreta de la actividad, ejercicio, tarea realizada.



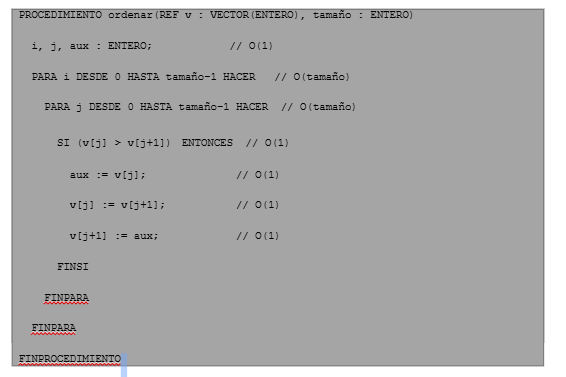
* Cada operación del procedimiento tiene un costo constante de O(1). Dado que no hay bucles ni operaciones recursivas, la complejidad total del procedimiento es: O(1) + O(1) + O(1) + O(1) = O(1), El orden computacional es O(1). Se debe a que todas las operaciones dentro del procedimiento tienen una complejidad constante y no dependen del tamaño de entrada.



* Cada operación de la función max tiene un costo constante de O(1)**.** Dado que no hay bucles ni operaciones recursivas, la complejidad total de la función es:
* O(1)+O(1)+O(1)+O(1)+O(1)=O(1)
* El orden computacional es O(1)**.** Esto se debe a que todas las operaciones dentro de la función tienen una complejidad constante y no dependen del tamaño de las entradas x y y**.**

****

* Cada operación del procedimiento suma tiene un costo constante de O(1). Sin embargo, debido al bucle PARA, la complejidad total depende del tamaño de la entrada, ya que el bucle recorre todos los elementos del vector. La complejidad del bucle es O(tamaño), y el resto de las operaciones fuera del bucle tienen un costo constante de O(1).
* Por lo tanto, la complejidad total del procedimiento es:
* O(1)+O(1)+O(tamaño)+O(1)=O(tamaño)O(1) + O(1) + O(tamaño) + O(1) = O(tamaño)
* El orden computacional es O(tamaño), ya que la complejidad del procedimiento depende del número de elementos en el vector v, y el tiempo de ejecución aumenta linealmente con el tamaño de la entrada.



* Cada operación dentro del procedimiento ordenar tiene un costo constante de O(1), como las comparaciones y las asignaciones dentro de los bucles. Sin embargo, debido a los dos bucles anidados, la complejidad total depende del tamaño de la entrada, ya que cada bucle recorre tamaño elementos.

Por lo tanto, la complejidad total del procedimiento es:

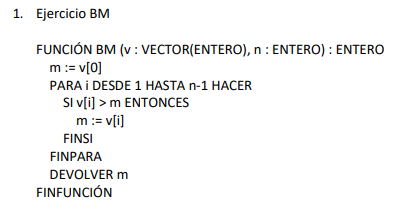
O(tamaño)×O(tamaño)=O(tamaño²)

* El orden computacional es O(tamaño²), ya que la ejecución del procedimiento depende del número de elementos en el vector v, y el tiempo de ejecución aumenta de manera cuadrática con el tamaño de la entrada.

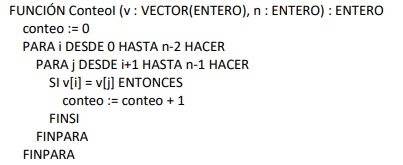
# **EJERCICIO 1.**

## **ENUNCIADO.**

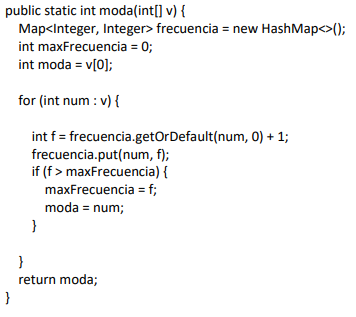
* Ejercicio BM



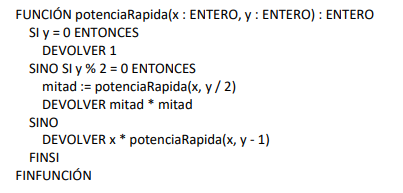
* Ejercicio ConteoI



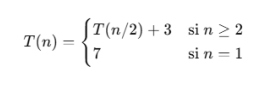
* Dado un arreglo, encuentra el número más frecuente usando un diccionario.



* Dado un arreglo, encuentra el número más frecuente usando un diccionario



* A partir de la siguiente función de recurrencia de un algoritmo, halle el orden de complejidad correspondiente:



* Investigue acerca del algoritmo de ordenación por fusión (Merge Sort) y cómo puede ser utilizado para ordenar un arreglo de enteros. En base a la información obtenida, escriba la implementación en Java de un algoritmo de O(n log n) que permita ordenar un arreglo de enteros utilizando el algoritmo de ordenación por fusión. Explique utilizando las reglas de la notación asintótica el por qué su orden de complejidad
* Analice el algoritmo propuesto del ejercicio anterior de modo que encuentra una versión mejorada. Explique haciendo uso de las reglas asintóticas el orden de complejidad encontrado, así mismo, explique las diferencias con el algoritmo propuesto anteriormente, haciendo énfasis en aquellos aspectos que redundaron en la mejora del algoritmo

## **RESPUESTA.**

**Ejercicio 1:**

|  |
| --- |
| public static int BM(int[] v) {  int m = v[0]; // O(1)  for (int i = 1; i < v.length; i++) { // O(n) - Recorre todo el vector  if (v[i] > m) { // O(1) – Compara una sola vez  m = v[i]; // O(1) – Se ejecuta si encuentra un máximo  }  }  return m; // O(1) - Retorno del resultado  } // Complejidad de algoritmo: O(n) Complejidad lineal |

**Ejercicio 2:**

|  |
| --- |
| public static int ConteoI(int[] v) {  int conteo = 0; // O(1) - Inicialización  for (int i = 0; i < v.length - 1; i++) { // O(n) Bucle  for (int j = i + 1; j < v.length; j++) { // O(n) Bucle  if (v[i] == v[j]) { // O(1) - Comparación  conteo++; // O(1) - Incremento del contador  }  }  }  return conteo; // O(1) - Retorno del resultado  } // Complejidad del algoritmo: O(n²)Complejidad Cuadrática |

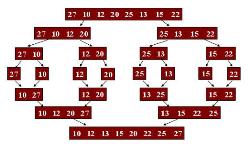
**Ejercicio 3:**

|  |
| --- |
| public static int moda(int[] v) {  Map<Integer, Integer> frecuencia = new HashMap<>(); // O(1) - Creación del HashMap, es algo constante.  int maxFrecuencia = 0; // O(1) - Inicialización  int moda = v[0]; // O(1) - Inicialización    for (int num : v) { // O(n) – Bucle  int f = frecuencia.getOrDefault(num, 0) + 1; // O(1) - Acceso e incremento  frecuencia.put(num, f); // O(1) - Inserción en el HashMap  if (f > maxFrecuencia) { // O(1) – Comparación, son operaciones constantes.  maxFrecuencia = f; // O(1) - Actualización de la moda  moda = num; // O(1) - Asignación  }  }  return moda; // O(1) - Retorno del resultado  } // Complejidad de algoritmo: O(n) Complejidad lineal |

**Ejercicio 4:**

|  |
| --- |
| public static int potenciaRapida(int x, int y) {  if (y == 0) return 1; // O(1) - Caso base  if (y % 2 == 0) {  int mitad = potenciaRapida(x, y / 2); // O(log n) - Llamada recursiva  return mitad \* mitad; // O(1) – Multiplicación solo una vez  } else {  return x \* potenciaRapida(x, y - 1); // O(log n) - Llamada recursiva y multiplicación  }  } // Complejidad de algoritmo: O(log n) Complejidad logaritmica |

**Ejercicio 6:**



O(n): En cada nivel de recursión, el tiempo de fusión es O(n) porque debemos procesar todos los elementos.

O(log n): La cantidad de niveles de recursión es log(n), ya que dividimos el arreglo por la mitad en cada paso.

**Ejercicio 7:**

Uso de in-place para así no generar espacio adicional, como lo hace normalmente el MergeSort, que al dividir el arreglo genera espacio adicional para poder hacer la división de cada elemento.