

August 5, 2023

## **Abstract**

## 0.1 Introducción

### 0.1.1 Comparación de Rendimiento entre Golang, Python y C++

En este trabajo, se llevará a cabo una comparación entre tres lenguajes de programación ampliamente utilizados: Go (Golang), Python y C++. El objetivo de esta comparación es evaluar el tiempo de procesamiento de cuatro programas implementados en cada uno de estos lenguajes.

El tiempo de procesamiento es un factor crucial en el rendimiento de un programa, ya que determina cuánto tiempo lleva ejecutar un conjunto de instrucciones y completar una tarea específica. La eficiencia en el tiempo de procesamiento es especialmente relevante en aplicaciones que manejan grandes cantidades de datos o realizan cálculos intensivos.

Para realizar la comparación, implementaremos cuatro programas con funcionalidades similares en cada uno de los lenguajes mencionados. Estos programas serán diseñados para realizar tareas representativas y representativas de cada lenguaje.

En cada caso, mediremos el tiempo de procesamiento de los programas utilizando herramientas adecuadas para obtener resultados precisos y comparables. La medición se realizará ejecutando cada programa varias veces y tomando el promedio para obtener un tiempo representativo.

Al comparar los resultados obtenidos, podremos evaluar cómo cada lenguaje maneja las tareas específicas y determinar cuál es más eficiente en términos de tiempo de procesamiento. Además, esta comparación nos permitirá identificar las fortalezas y debilidades de cada lenguaje para diferentes tipos de aplicaciones y escenarios de uso.

Es importante destacar que la eficiencia en el tiempo de procesamiento es solo uno de los muchos factores que deben tenerse en cuenta al elegir un lenguaje de programación. Otros aspectos como la facilidad de uso, la legibilidad del código, el ecosistema de bibliotecas y la comunidad de desarrollo también son consideraciones importantes.

En resumen, este estudio comparativo proporcionará información valiosa sobre el rendimiento en tiempo de procesamiento de Go, Python y C++, lo que permitirá a los desarrolladores tomar decisiones informadas al elegir el lenguaje más adecuado para sus proyectos y requisitos específicos.

## 0.2 Algoritmos

**Bubble Sort:** Bubble Sort es un algoritmo simple de ordenación que funciona comparando repetidamente elementos adyacentes y haciendo intercambios si están en el orden incorrecto. El proceso se repite hasta que no se realicen más intercambios, lo que indica que la lista está completamente ordenada. Aunque es fácil de implementar, Bubble Sort no es eficiente para listas grandes.

**Merge Sort:** Merge Sort es un algoritmo de ordenación basado en la técnica

[scale=0.4]C:/Users/Jhoel/Documents/python/homeworks/Reporte/Griffith  
University Thesis Template/images/1.JPG

”divide y vencerás”. Divide la lista no ordenada en sublistas más pequeñas, las ordena de forma recursiva y luego las combina para obtener la lista ordenada final. Merge Sort tiene una complejidad promedio de  $O(n \log n)$ , lo que lo hace mucho más eficiente que Bubble Sort para grandes conjuntos de datos. Es ampliamente utilizado en aplicaciones que requieren una ordenación estable y eficiente.

[scale=0.3]images/2.JPG

Quick Sort: Quick Sort es otro algoritmo de ordenación ”divide y vencerás” que utiliza una estrategia de partición para ordenar los elementos. Selecciona un elemento como ”pivote” y divide la lista en dos subconjuntos: uno con elementos menores que el pivote y otro con elementos mayores. Luego, aplica recursivamente el mismo proceso a los dos subconjuntos. Quick Sort tiene una complejidad promedio de  $O(n \log n)$  y, en general, es más rápido que Merge Sort para conjuntos de datos más pequeños.

[scale=0.4]images/3.JPG

Selection Sort: Selection Sort es un algoritmo de ordenación que busca el elemento más pequeño en cada iteración y lo coloca en su posición correcta. Comienza seleccionando el elemento más pequeño y lo intercambia con el primer elemento. Luego, busca el siguiente elemento más pequeño y lo coloca en la segunda posición, y así sucesivamente. En resumen, cada algoritmo de ordenación tiene sus ventajas y desventajas en términos de eficiencia y facilidad de implementación. La elección del algoritmo de ordenación adecuado dependerá del tamaño del conjunto de datos y los requisitos de rendimiento del problema específico que estás abordando.