Análisis de algoritmos : Taller 1 Parte 2

Carlos Escobar [1] Fabián Rojas[1] Julián Tarazona[1]

[1]Pontificia Universidad Javeriana
{escobartc, fabian-rojasm, jdtarazona}@javeriana.edu.co
February 24, 2022

1 Resumen

En este documento se presenta la implementación en el lenguaje de programación Rust del problema del ordenamiento de elementos y su solución a través del algoritmo de TimSort, el cual es una combinación de los algoritmos merge sort y binary insertion sort, que es a su vez una mezcla de insertion sort y binary search. También se basa en el patrón dividir y vencer, además de ser utilizado como el algoritmo de ordenamiento nativo para el lenguaje Python.

2 Pruebas

Para realizar las pruebas se plantea medir el tiempo de ejecución del algoritmo Timsort en el lenguaje de programación Rust de acuerdo a la cantidad de elementos que tiene una secuencia S a ordenar. De acuerdo a esto,se utilizó la plataforma Replit.com para el desarrollo de las pruebas, con una máquina virtual la cual poseía 0.5 vCPUs, 1GiB de memoria RAM y 1 GiB de almacenamiento. Se realizaron múltiples mediciones de tiempo con una cantidad de elementos en concreto, más específicamente diez veces. Entonces se realizan 10 corridas y mediciones de tiempo para los valores de 10, 50, 100, 500, 1000, 5000, 10000, 15000 y 20000 elementos. Una vez realizadas las 10 corridas, se promedia el tiempo de ejecución y se asigna en la tabla. Posteriormente para las pruebas se planteó realizar una recopilación de datos de la tabla que muestren la relación tamaño del arreglo y tiempo de ejecución del algoritmo, para a partir de esto generar una visualización en la herramienta Power BI y ver cómo se comporta este algoritmo entre más elementos deba ordenar de una secuencia. Los resultados se presentan a continuación:

Elementos	Tiempo (Segundos)
10	0.000002418
50	0.000044918
100	0.000166803
500	0.008152296
1000	0.017573997
5000	0.83440774
10000	2.575349031
15000	7.647379633
20000	12.72385904

Figure 1: Tabla del tiempo de ejecución de acuerdo a la cantidad de elementos de una secuencia para el algoritmo TimSort en Rust.

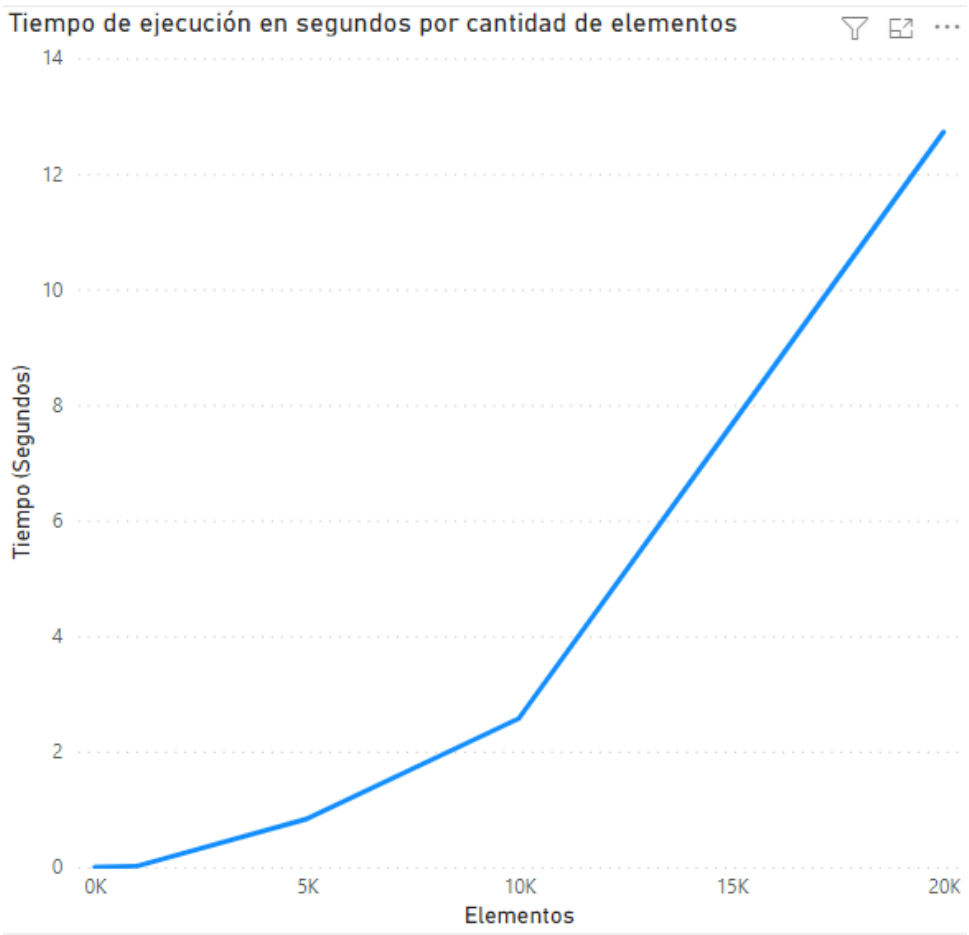

Figure 2: Gráfico de tiempo de ejecución de acuerdo a la cantidad de elementos de una secuencia para el algoritmo TimSort en Rust. (Elaborado con power Bi)

3 Conclusión

Los resultados presentados en este documento muestran que el algoritmo Timsort diseñado en la primera parte de este trabajo no es ideal para el ordenamiento de grandes secuencias de números, a diferencia del algoritmo original que se encuentra implementado dentro del lenguaje Python. Otro inconveniente presentado fue en el uso del lenguaje Rust para la implementación del algoritmo, ya que las constantes limitaciones que presentaba el lenguaje sobre el manejo de la memoria para la asignación de espacio en los arreglos no permitieron realizar pruebas con arreglos con diferentes tamaños, por lo que se opto por realizar dichas pruebas con tamaños fijos.