Tarea Corta 1

IC3002 - Análisis de Algoritmos Profe.: Yuen Law Wan I Semestre 2020

Joseph Tenorio Pereira
2019064588

Jose Pablo Muñoz
2019061904

***Abstract*—This documentation demonstrates a comparison between the execution time of two popular sorting algorithms used in computer science, specifically the Bubble-Sort algorithm vs the Quick-Sort algorithm. The results are presented in a comparative graph created in Unity by measuring the execution time of each algorithm several times and constantly increasing the amount of elements to be sorted by each one. Additionally, the algorithm's logic is briefly reviewed.**

I. Introducción

Los algoritmos de ordenamiento nos permiten poner a los elementos de una lista en una secuencia ordenada (por ejemplo una lista numérica de forma ascendente o descendente), lo cual simplifica el trabajo de una amplia variedad de algoritmos que resuelven distintos problemas, como la búsqueda de un elemento en dicha lista. Existen muchos algoritmos de ordenamiento distintos, algunos más complejos que otros, y algunos más eficientes también. Por ende, es importante estar informado del rendimiento de cada uno de ellos para utilizarlos correctamente.

II. Descripción de los algoritmos

A continuación se dará una breve explicación de los algoritmos de ordenamiento analizados por medio de la graficación en *Unity*. Estos corresponden a los métodos de *Bubble Sort* y *Quick Sort*, programados en el lenguaje *C#*. En dicha graficación, se estableció una relación entre la cantidad de elementos a ordenar y la cantidad de microsegundos que le toma al algoritmo el realizar dicho ordenamiento.

A. Algoritmo de ordenamiento Bubble Sort

El primer método de ordenamiendo analizado corresponde al ordenamiento de burbuja, mejor conocido como *Bubble Sort*. Este método es una simple e intuitiva solución al problema de acomodar ascendentemente una serie de números. Dicha solución consiste en comparar cada número del arreglo con el número siguiente (excepto el último número), e intercambiarlos si están en el orden incorrecto. Dicho recorrido, con sus respectivas comparaciones e intercambios, se realiza una vez por cada elemento del arreglo (como es el caso del algoritmo empleado en este trabajo),o bien, hasta que se realiza un recorrido completo sin intercambios.

B. Algoritmo de ordenamiento Quick Sort

Por otro lado, el segundo método de ordenamiento analizado corresponde al ordenamiento rápido, mejor conocido como *Quick Sort*. Este algoritmo pertenece a la categoría de divide y vencerás, en inglés, *Divide and Conquer*, ya que selecciona un elemento del arreglo como pivote y particiona los elementos de dicho arreglo en mayores y menores que el pivote, posteriormente se ubica el pivote en la poisición correcta [1]. El proceso anterior es luego repetido recursivamente sobre los arreglos resultantes de la partición, hasta que todos ellos queden ordenados. El algoritmo empleado en el presente análisis toma como pivote al primer elemento de los arreglos y fue tomado de [1].

III. Análisis de resultados

Como ya se mencionó anteriormente, el gráfico empleado para este trabajo, apreciable en [Fig. 1], establece un relación entre el número de elementos a ordenar (eje X) y el tiempo de ejecución (eje Y, en microsegundos) de los algoritmos de *Bubble Sort* y *Quick Sort*. Cabe resaltar que para cada tamaño de arreglo del eje X se realizaron cinco corridas de los algoritmos, representadas por los puntos verdes y celestes según sea el caso.

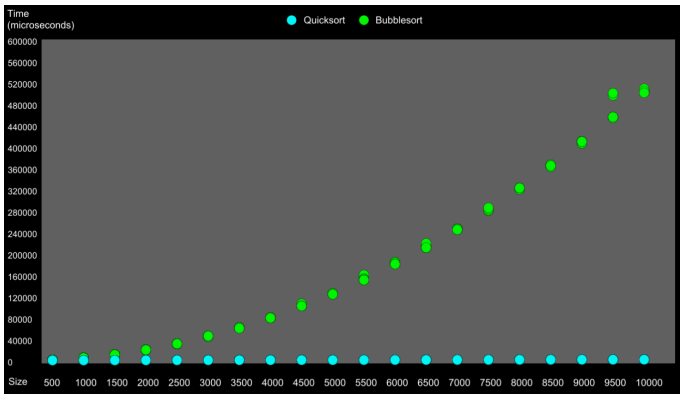

Fig. 1. Gráfico resultante en *Unity*.

El primer detalle a notar es el hecho de que el algoritmo de *Bubble Sort* se vuelve extremedamente ineficiente conforme aumenta el tamaño del arreglo a ordenar, en especial si lo comparamos con el algoritmo de *Quick Sort*. De hecho, la graficación podría dar a entender que el algoritmo de *Quick Sort* mantiene un tiempo de ejecución constante, pero este no es el caso. En realidad, el crecimiento del tiempo de ejecución del algoritmo de *Quick Sort* no se puede observar en las escalas requeridas para representar el crecimiento exponencial del

tiempo de ejecución del algoritmo de *Bubble Sort*. Lo anterior se debe a que el metodo *Quick Sort* presenta un crecimiento de tiempo de ejecución logarítmico, el cual, nuevamente, es inapreciable debido a la escala usada. Otro detalle a notar en los resultados obtenidos es que los tiempos de ejecución del algoritmo de *Bubble Sort* empleado poseen una mayor varianza conforme crece el tamaño del arreglo ordenado.

IV. Acceso al proyecto

El repositorio de GitHub en el cual se ubica el proeycto de *Unity* empleado puede ser accesado mediante el siguiente link: https://github.com/TilapiaBoi/AnalisisTC1.git

Bibliografía

[1] W3resource.com, *C# Sharp Searching and Sorting Algorithm Exercises: Quick sort*, Nov. 2019. Accesado el: Feb. 21, 2020. [Online]. Disponible en: https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-9.php

[2] Code Monkey, *Simple UI Setup (Unity Tutorial for Beginners)*, Jul. 17, 2018. Accesado el: Feb. 23, 2020. [Archivo de video] Disponible en: https://www.youtube.com/watch?v=VHFJgQraVUs

[3] Code Monkey, *Unity Tutorial - Create a Graph*, Jun. 22, 2018. Accesado el: Feb. 18, 2020. [Archivo de video] Disponible en: https://www.youtube.com/watch?v=CmU5-v-v1Qo&t=226s