Universidad Nacional de San Agustín, Escuela de Ingenieria de Sistemas

ADA - Lab 02

Fernando Enrique Araoz Morales - 20173373

20 de octubre de 2019

1. Introducción

En el presente documento se presenta un análisis del tiempo de ejecución del algoritmo Bubble Sort, con el fin de demostrar el crecimiento exponencial de su tiempo de ejecución.

2. Implementación

La implementacion del algoritmo es la misma provista en la carpeta compartida de Google Drive.

```
public class BubleSort implements IAlgorithm {

    private int length;

    BubleSort(int length) {
        this.length = length;
    }

    @Override
    public String description() {
        return "Bubble Sort Problem";
    }

    @Override
    public boolean run() {
        // Creates a random array
```

```
        int[] data = new int[length];
        for (int i = 0; i < length; i++) {
            data[i] = (int) (Math.random() * 10000);
        }
        System.out.print(length + "i terminado. ");

        // explosive bubble sort
        for (int i = 0; i < length - 1; i++) {
            for (int j = 1; j < length; j++) {
                if (data[j - 1] > data[j]) {
                    int temp = data[j - 1];
                    data[j - 1] = data[j];
                    data[j] = temp;
                }
            }
        }
        System.out.print("Orden terminado. ");

        // how to validate??
        for (int i = 1; i < length; i++) {
            if (data[i - 1] > data[i]) return false;
        }
        return true;
    }

}
```

Para facilitar la creacion de datos, el tamaño del array de enteros es dinámico, mediante la incorporación de un atributo length.
La siguiente clase se encarga de ejecutar el algoritmo con diferente numero de elementos, y escribe los datos revelantes en un archivo data.csv. Estos datos se describen con mayor detalle en una sección posterior.

```
import java.io.FileWriter;
import java.io.IOException;

public class BubbleSortDataWriter {

    private String getData(int iteration) {
        long startTime = System.currentTimeMillis();
        boolean result = new BubleSort(iteration).run();
        long endTime = System.currentTimeMillis();
        System.out.println("(" + (endTime - startTime) + "ms)");

        return iteration + ", " + startTime + ", " + endTime + ", " +
        (endTime - startTime) + ", " + result + "\n";
    }
```

```
    public void writeData(int iterations, int multiplier, int sum) throws
        IOException {
        FileWriter fout = null;
        try {
            String writeData;

            fout = new FileWriter("./data.csv");
            fout.write("number_of_elements, start_time, end_time, duration,
                result\n");
            for (int i = 1; i <= iterations; ++i) {
                writeData = getData(i * multiplier + sum);
                fout.write(writeData);
            }

        } catch (IOException e) {
            System.err.println("Error al abrir el archivo para escritura.\n"
                + e.getMessage());
            e.printStackTrace();
        } finally {
            if (fout != null) {
                fout.close();
            }
        }
    }

}
```

Los parámetros multiplier y sum del método writeData se usan para controlar la cantidad de elementos a ordenar, e iterations es la cantidad de iteraciones a realizar y escribir en el archivo data.csv

3. Tiempos de ejecución

El algoritomo se ejecutó en diferentes intervalos con diferentes cantidades de datos, para poder tener la mayor cantidad de información usando la menor cantidad de tiempo.
En primer lugar, se ejecuto el algoritmo 8000 veces, con desde 1 elemento hasta 8000 elementos, aumentando 1 elemento en cada iteración.
El siguiente segmento realizó iteraciones desde los 8000 elementos hasta los 50000 elementos, incrementado en intervalos de 1000 elementos por iteración.
Luego, iniciando en los 50000 elementos se ejecuto el algoritmo en intervalos de 10000 elementos hasta alcanzar los 100000 elementos.
Finalmente, en intervalos de 50000 elementos se alcanzó la cifra de 400000 elementos.
Los resultados de estas pruebas se encuentran ilustradas en el suguiente gráfico:
Los tiempos de ejecución alcanzan los 400000ms para el caso con 500000 elementos.

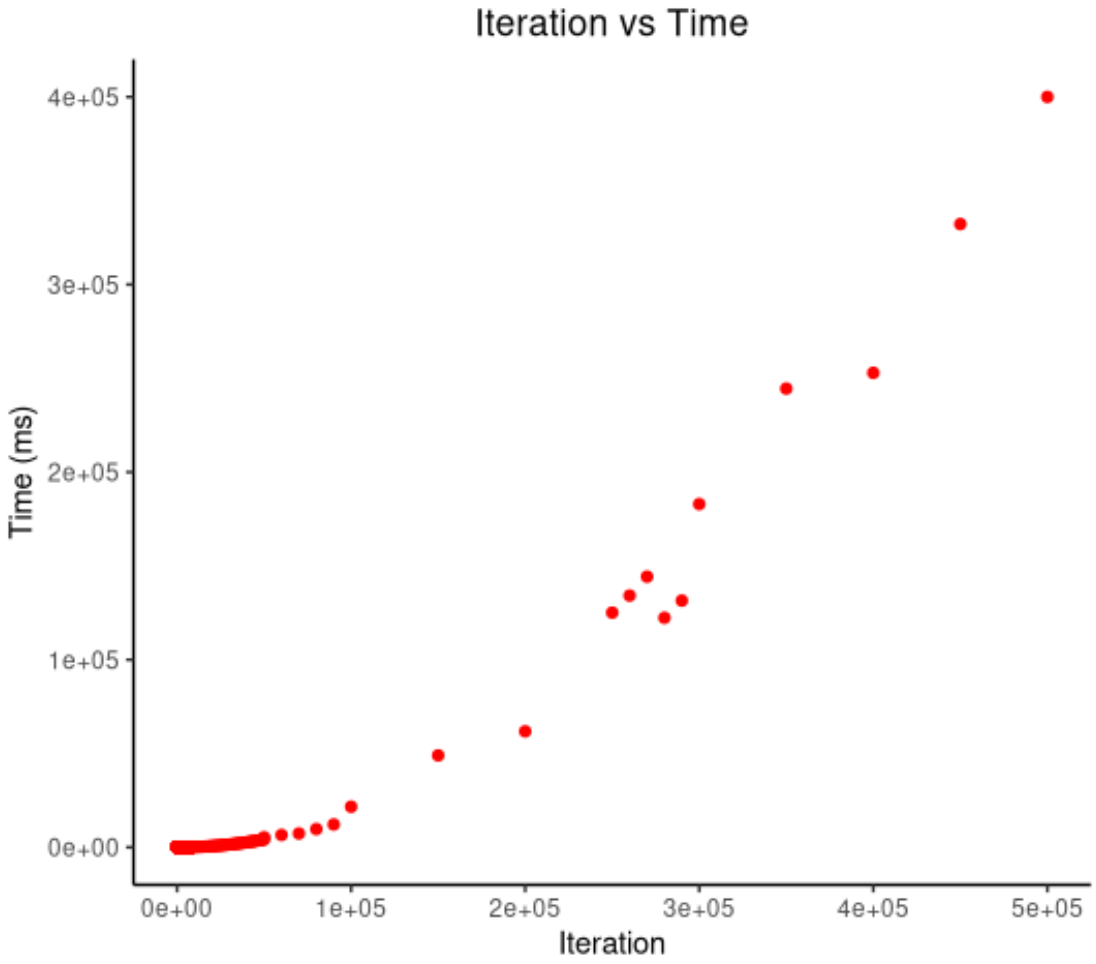

Figura 3.1: Incremento del tiempo de ejecución de Bubble Sort.

4. Conclusión

Al observar el gráfico mostrado, podemos apreciar el tiempo de ejecución exponencial de este algoritmo, y con ello comprender la importancia del análisis y diseño de algoritmos. A través de este, podemos reducir drásticamente el tiempo de ejecución de un algoritmo, mejorando su eficiencia masivamente.