Universidad Nacional de San Agustín, Escuela de Ingenieria de Sistemas

ADA - Lab 03

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 Closest Pair, así como las respuestas a otras interrogantes relacionadas.

2. Implementación

La implementacion del algoritmo inicia por la definición de una clase Coordenada. Esta tiene como únicos atributos un entero denotando una posicion x, y otro para una posicion y. Estos son finales y protegidos para así evitar toda la ceremonia de crear getters, setters, y manejar el state.

```
class Coordinate {
    final int x;
    final int y;

    Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }

}
```

Luego, la implementación de la clase propiamente dicha. Esta consta de los métodos run y describe requeridos por la interfaz IAlgorithm, así como métodos para el cálculo del

algoritmo con fuerza bruta y eficientemente, un método para clonar ArrayList, y otro para separar y ordenar el ArrayList.

```
public class ClosestPairProblem implements IAlgorithm {

    private ArrayList<Coordinate> coordinates;
    double result;

    ClosestPairProblem(ArrayList<Coordinate> coordinates) {
        this.coordinates = coordinates;
    }

    @Override
    public boolean run() {
        result = calc();
        return true;
    }

    double calc() {
        int size = coordinates.size();
        ArrayList<Coordinate> coordinates_x = extractElements(coordinates,
            0, size);
        coordinates_x.sort((c1, c2) -> c1.x - c2.x);
        ArrayList<Coordinate> coordinates_y = extractElements(coordinates,
            0, size);
        coordinates_y.sort((c1, c2) -> c1.y - c2.y);
        return efficientClosestPair(coordinates_x, coordinates_y);
    }

    private double bruteForceClosestPair(ArrayList<Coordinate> coordinates)
        {
        double minimunDistance = Double.MAX_VALUE;

        for (int i = 0; i < coordinates.size(); i++) {
            Coordinate c1 = coordinates.get(i);
            for (int j = i + 1; j < coordinates.size(); j++) {
                Coordinate c2 = coordinates.get(j);
                double distance = Math.sqrt(
                    Math.pow(c1.x - c2.x, 2) +
                    Math.pow(c1.y - c2.y, 2)
                );
                minimunDistance = Math.min(minimunDistance, distance);
            }
        }

        return minimunDistance;
    }
```

```
private ArrayList<Coordinate> extractElements(ArrayList<Coordinate>
    elements,
        int start, int end) {
    ArrayList<Coordinate> result = new ArrayList<>();
    for (int i = start; i < end; i++) {
        result.add(elements.get(i));
    }
    return result;
}

private double efficientClosestPair(ArrayList<Coordinate> coordinates_x,
        ArrayList<Coordinate> coordinates_y) {
    if (coordinates_x.size() <= 3) return
        bruteForceClosestPair(coordinates_x);

    int size = coordinates_x.size();
    int half_limit = (int) Math.ceil(coordinates_x.size() / 2);
    ArrayList<Coordinate> c_x_left = extractElements(coordinates_x, 0,
        half_limit);
    ArrayList<Coordinate> c_y_left = extractElements(coordinates_y, 0,
        half_limit);

    ArrayList<Coordinate> c_x_right = extractElements(coordinates_x,
        half_limit, size);
    ArrayList<Coordinate> c_y_right = extractElements(coordinates_y,
        half_limit, size);

    double minimum_left = efficientClosestPair(c_x_left, c_y_left);
    double minimum_right = efficientClosestPair(c_x_right, c_y_right);

    double minimum = Math.min(minimum_left, minimum_right);
    Coordinate mediumPoint = coordinates_y.get(half_limit - 1);

    ArrayList<Coordinate> mediumElements = new ArrayList<>();
    for (Coordinate c: coordinates_x) {
        if (Math.abs(c.x - mediumPoint.x) < minimum) {
            mediumElements.add(c);
        }
    }

    double dminsq = minimum * minimum;

    for (int i = 0; i < mediumElements.size() - 2; i++) {
        int k = i + 1;
        Coordinate c1 = mediumElements.get(i);
        Coordinate c2 = mediumElements.get(k);
        double value = Math.pow(c1.y - c2.y, 2);
        while (k <= size - 1 && value < dminsq) {
            double result = Math.pow(c1.x - c2.x, 2) + Math.pow(c1.y -
```

```
                    c2.y, 2);
                dminsq = Math.min(result, dminsq);
                k++;
            }
        }

        return Math.sqrt(dminsq);
    }

    @Override
    public String description() {
        return "Calculates the closes pair from a list of coordinates.";
    }

}
```

Esta implementación sigue como base aquella brindada en clase. Algunas peculiaridades son que para satisfacer la interfaz IAlgorithm, el método run() simplemente ejecuta el método calc(), y almacena el resultado de este en la propiedad de clase resultado, la cual el usuario deberá usar.
El método calc() clona el ArrayList brindado via constructor en uno ordenado según el eje x y otro ordenado según el eje y. Para realizar el ordenamiento se utilizó el método sort de ArrayList, y expresiones lambda.

3. Tiempo de ejecución

El algoritmo, debido a su implementación, tiene un tiempo de ejecución corto. Con 100 elementos tarda 71ms, con 1000 elementos tarda 76ms, con 10000 tarda 116ms, con 100000 elementos aumenta a 360ms, y con 1000000 elementos alcanza los 1531ms.

4. Gráficos

Los gráficos del plano se encuentran a continuación:

5. Descripción de los gráficos

A partir de los gráficos provistos anteriormente, y usando como referencia el tiempo de ejecución del algoritmo, podemos observar que a pesar de que la cantidad de elementos en cada caso aumenta de manera muy significativa, el tiempo de ejecución no lo hace, indicando una vez más la eficiencia del algorítmo.

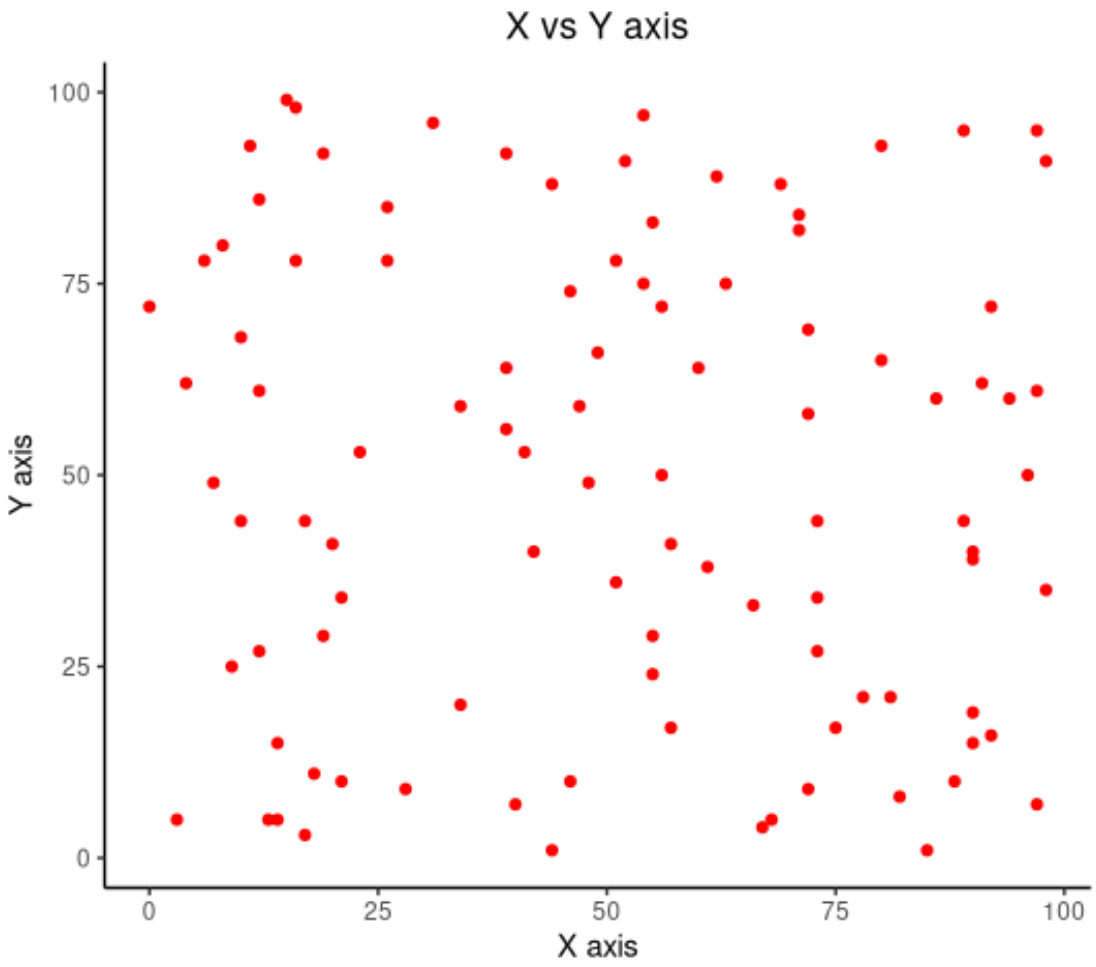

Figura 4.1: 100 elementos.

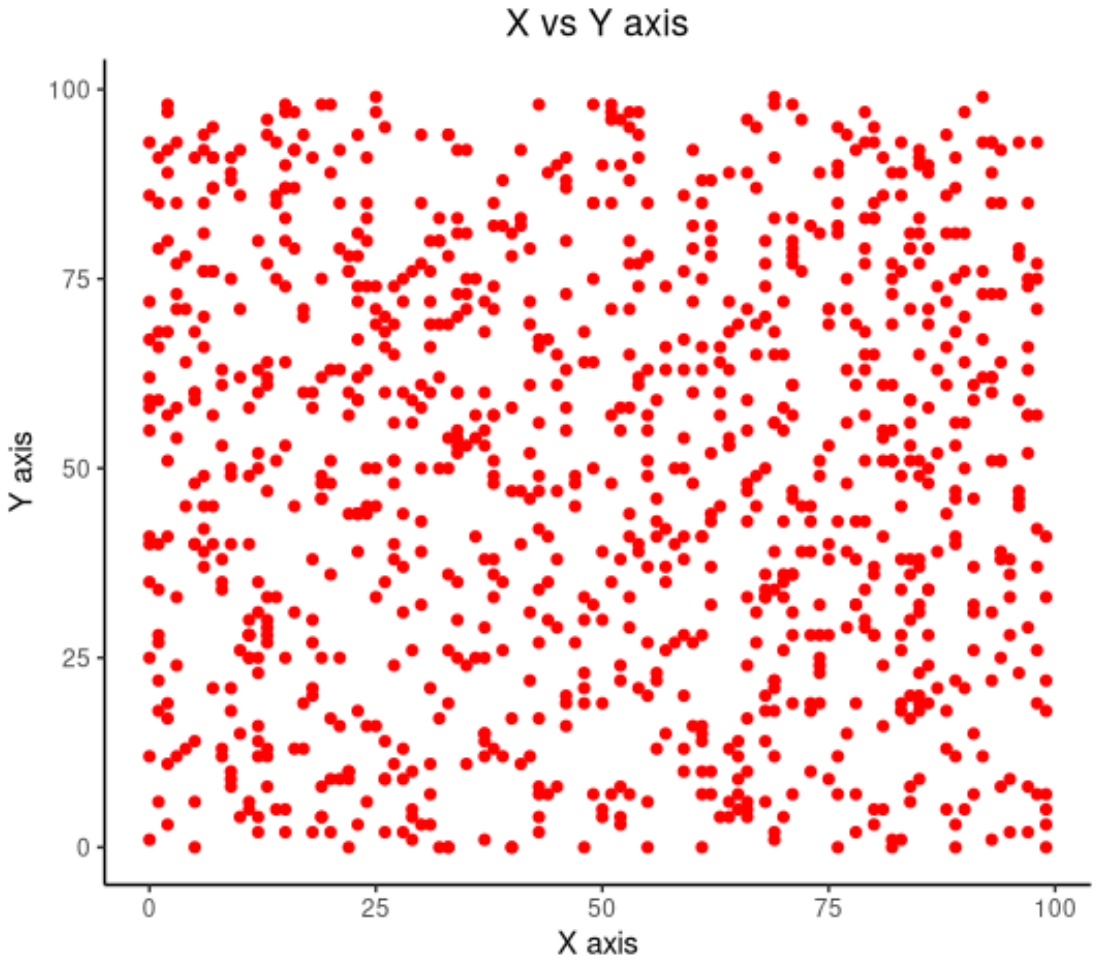

Figura 4.2: 1000 elementos.

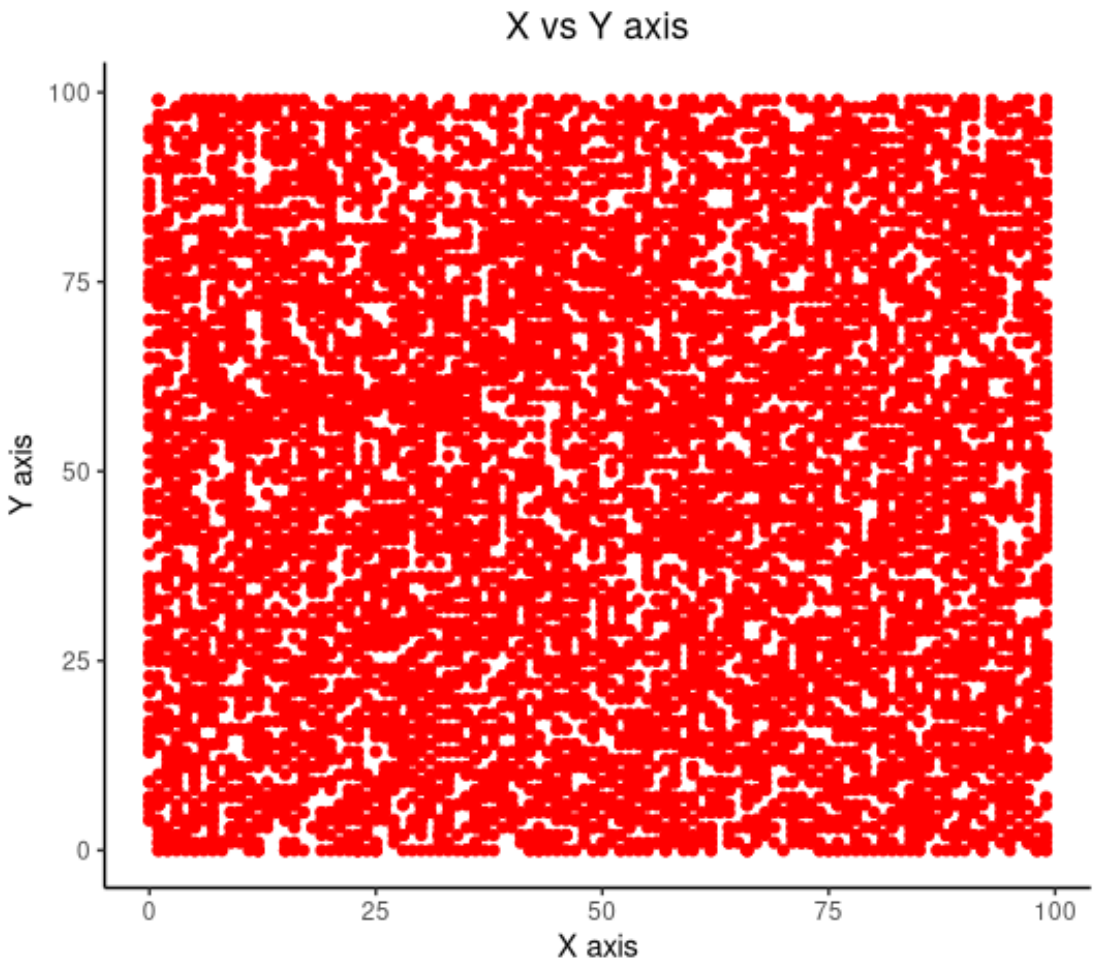

Figura 4.3: 10000 elementos.

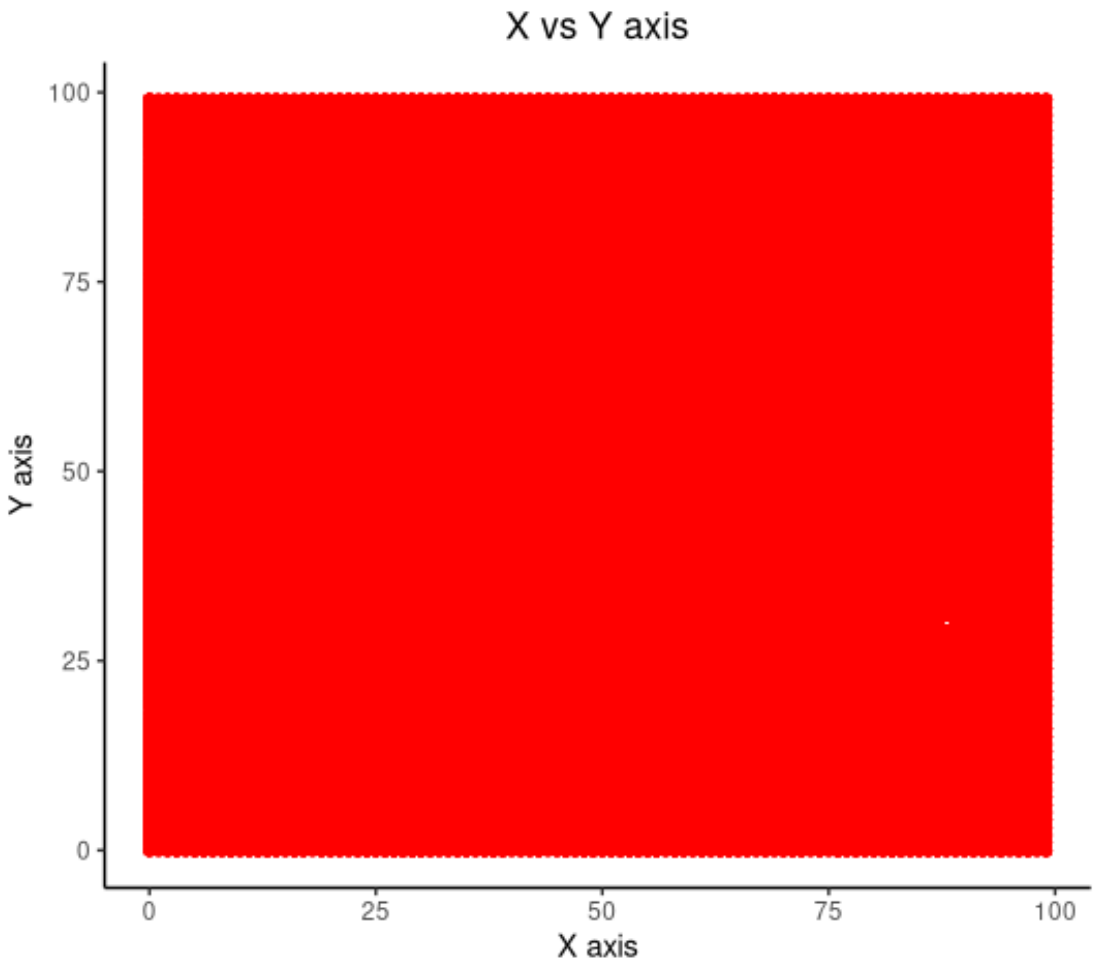

Figura 4.4: 100000 elementos.

6. Límite de puntos para comparación entre fuerza bruta y divide y venceras.

En el pseudocódigo brindado en clase este límite se establece en 3 elementos. Sin embargo, es posible aumentar este sin afectar fuertemente el rendimiento a una gran escala. Hay que notar, sin embargo, que la diferencia de rendimiento es cuadrático vs n log n, por lo que se busca es minimizar este límite al máximo.

7. Costo usando teorema maestro

El costo se puede calcular de la siguiente manera:
T(n) = 2T(n/2) + O(n) + O(nLogn) + O(n)
T(n) = 2T(n/2) + O(nLogn)
T(n) = T(n x Logn x Logn)
En este caso tomamos como base un algoritmo de ordenamiento de orden O(n log n).