Practica1:Estrategias Algorítmicas

Algoritmos y Modelos de Computación

Wadadi Sidelgaum limam

Índice:

Introducción:

En esta práctica vamos a estudiar cómo se comportan los algoritmos Divide y Vencerás, y los algoritmos Voraces teóricamente como empíricamente. Estos algoritmos recibirán un array en el caso de DyV o un grafo en caso de los Voraces. Utilizaremos Java como lenguaje y NetBeans como compilador para ver la eficiencia de los algoritmos. Acompañaremos dicho estudio con gráficas para hacerlo más entendible y didáctico, para que cualquiera que lea esta memoria entienda este algoritmo a la perfección.

Esta práctica está compuesta de dos partes. La primera tratará de encontrar, dado una serie de puntos, la linea de menor perímetro y mayor área. La segunda dado un grafo deberá calcular el árbol de recubrimiento mínimo.

Para las pruebas utilizaremos un pequeño “main”, creado exclusivamente para la práctica y que ya se ha eliminado de la entrega, puesto que no se pide para esta práctica.

2.Exhaustivo

- 1.Código

```
public static Parpuntos Exhaustivo(Punto p[], int in,int f) {
      int c=0;
    Parpuntos l = new Parpuntos(p[0], p[1]);
    double min = l.distancia();
    for (int i = in; i < f; i++) {
        for (int j = i + 1; j < f; j++) {
            Parpuntos aux = new Parpuntos(p[i], p[j]);
            c++;
            if (aux.distancia() < min) {
                l.setLinea(p[i], p[j]);
                min = aux.distancia();
            }
        }
    }
    l.setCalculadas(calculadas: c);
    return l;
}
```

Siendo c el número de veces que calculamos la distancia y l es una instancia de la clase Parpuntos que va a contener los dos puntos con la distancia más corta entre ellos. Y min es una variable para ir guardando la menor distancia que hay en el momento del array de puntos.

Lo que hacemos es un bucle que va a recorrer todos los puntos y para cada punto irá comparando su distancia con el resto de los puntos.
Finalmente devolvemos la solución.

2.Exhaustivo

- 2.Estudio Teórico

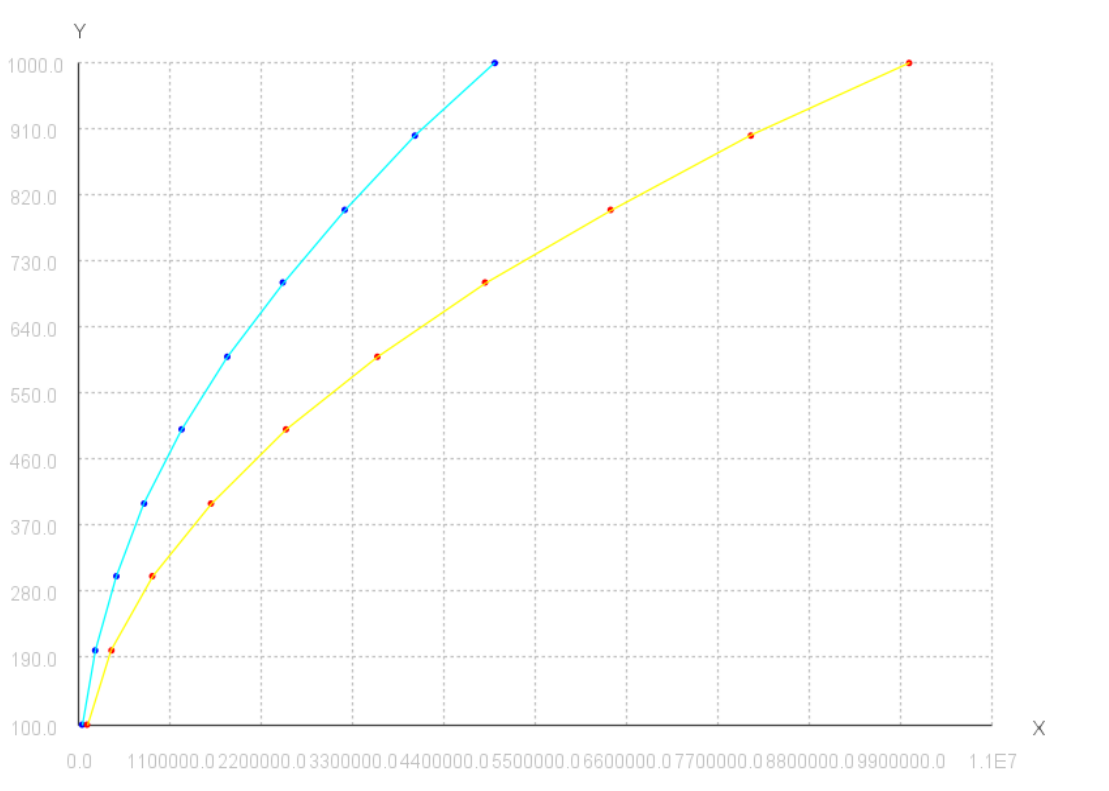
```
public static Parpuntos Exhaustivo(Punto p[], int in,int f) {
      int c=0;                                                    //1OE
    Parpuntos l = new Parpuntos(p[0], p[1]);                      //3OE
    double min = l.distancia();                                   //3OE
    for (int i = in; i < f; i++) {                                //1OE+3OE
        for (int j = i + 1; j < f; j++) {                         //2OE+3OE
            Parpuntos aux = new Parpuntos(p[i], p[j]);            //3OE
            c++;                                                  //2OE
            if (aux.distancia() < min) {                          //3OE
                l.setLinea(p[i], p[j]);                           //5OE
                min = aux.distancia();                            //3OE
            }
        }
    }
    l.setCalculadas(calculadas: c);                               //1OE
    return l;                                                     //1OE
}
```

OE(Exhaustivo) = 1+3+3+2+Sumatorio(5+Sumatorio(2+3+3+8+2+5+3))+1+1
El primero sumatorio va desde 0 hasta n y el segundo desde i+1 hasta n.

- Caso mejor: Si entra una sola vez en el if o sea solo una vez la distania es menor que la que tenimaos antes.
Exhaustivo =9+sumatorio(5+sumatorio(10)+10)=9+15n+10*sumatorio(i) = $5n^2+10n+9$
- Caso peor: siempre se ejecuta el if:
Exhaustivo=9+sumatorio(5+sumatorio(20))=9+15n+20*sumatorio(i) =$10n^2+5n+9$

2.Exhaustivo

- 3.Gráfica



3.Poda

- 1.Código

```
public static Parpuntos Bpoda(Punto p[], int f) {
    int c=0;
    quick_sort(p, in: 0, f);
    Parpuntos l = new Parpuntos(p[0], p[1]);
    double min = l.distancia();
    c++;
    int j;
    for (int i = 0; i < f; i++) {
        j = i + 1;
        while (j < f) {
            Parpuntos aux = new Parpuntos(p[i], p[j]);
            c++;
            if (aux.distancia() > min) {
                break;
            }
            l.setLinea(p[i], p[j]);
            min = aux.distancia();
            j++;
        }
    }
    l.setCalculadas(calculadas: c);
    return l;
}
```

```
public static int partition(Punto p[], int in, int f, Punto pivote) {
    int i = in;
    int j = in;
    while (i <= f) {
        if (p[i].getX() > pivote.getX()) {
            i++;
        } else {
            intercambiar(p, p1: i, p2: j);
            i++;
            j++;
        }
    }
    return j - 1;
}

public static void intercambiar(Punto p[], int p1, int p2) {
    Punto aux;
    aux = p[p1];
    p[p1] = p[p2];
    p[p2] = aux;
}

public static  void  quick_sort(Punto p[], int in, int f) {
    Punto pivote;
    int q;
    if (in < f) {
        pivote = p[f];
        q = partition(p, in, f, pivote);
        quick_sort(p, in, q - 1);
        quick_sort(p, q + 1, f);
    }
}
```

3.Poda

- 2.Estudio Teórico

```
public static Parpuntos Bpoda(Punto p[], int f) {
    int c=0;                                                    //1OE
    quick_sort(p, in: 0, f);                                    //N LOGN
    Parpuntos l = new Parpuntos(p[0], p[1]);                    //3OE
    double min = l.distancia();                                 //3OE
    c++;                                                        //2OE
    int j;
    for (int i = 0; i < f; i++) {                               //2OE+3OE
        j = i + 1;                                              //2OE
        while (j < f) {                                         //1OE
            Parpuntos aux = new Parpuntos(p[i], p[j]);          //3OE
            c++;                                                //2OE
            if (aux.distancia() > min) {                        //3OE
                break;
            }
            l.setLinea(p[i], p[j]);                             //3OE
            min = aux.distancia();                              //3OE
            j++;                                                //2OE
        }
    }
    l.setCalculadas(calculadas: c);
    return l;                                                   //1OE
}
```

OE(PODA) = 1+3+3+2+nlogn+Sumatorio(5+Sumatorio(2+1+3+2+3+3+3+2))+1

El primero sumatorio va desde 0 hasta n y el segundo desde i+1 hasta n en el caso peor y en el médio hasta n/2;

- Caso medio: Si entra la mitad de las veces en el if o sea solo una vez la distania es menor que la que tenimaos antes.

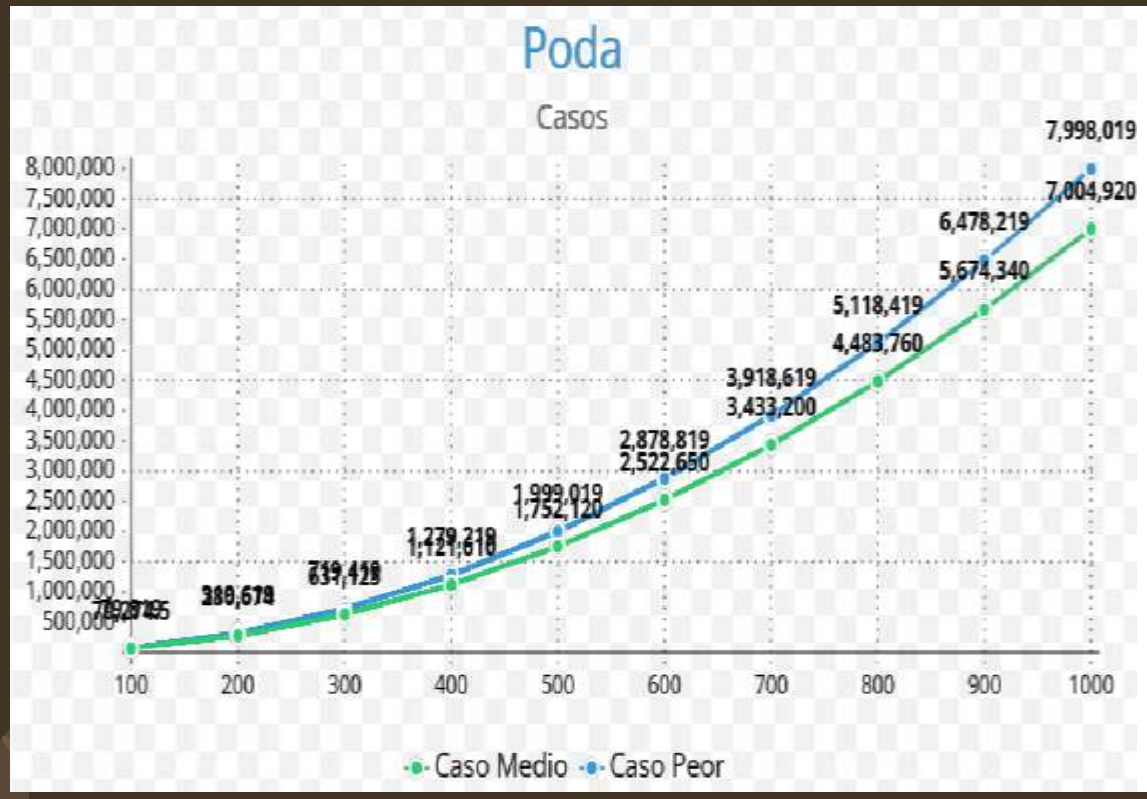
Poda=9+nlogn+sumatorio(5+sumatorio(15)) =$3/2n^2$-n+nlogn+15

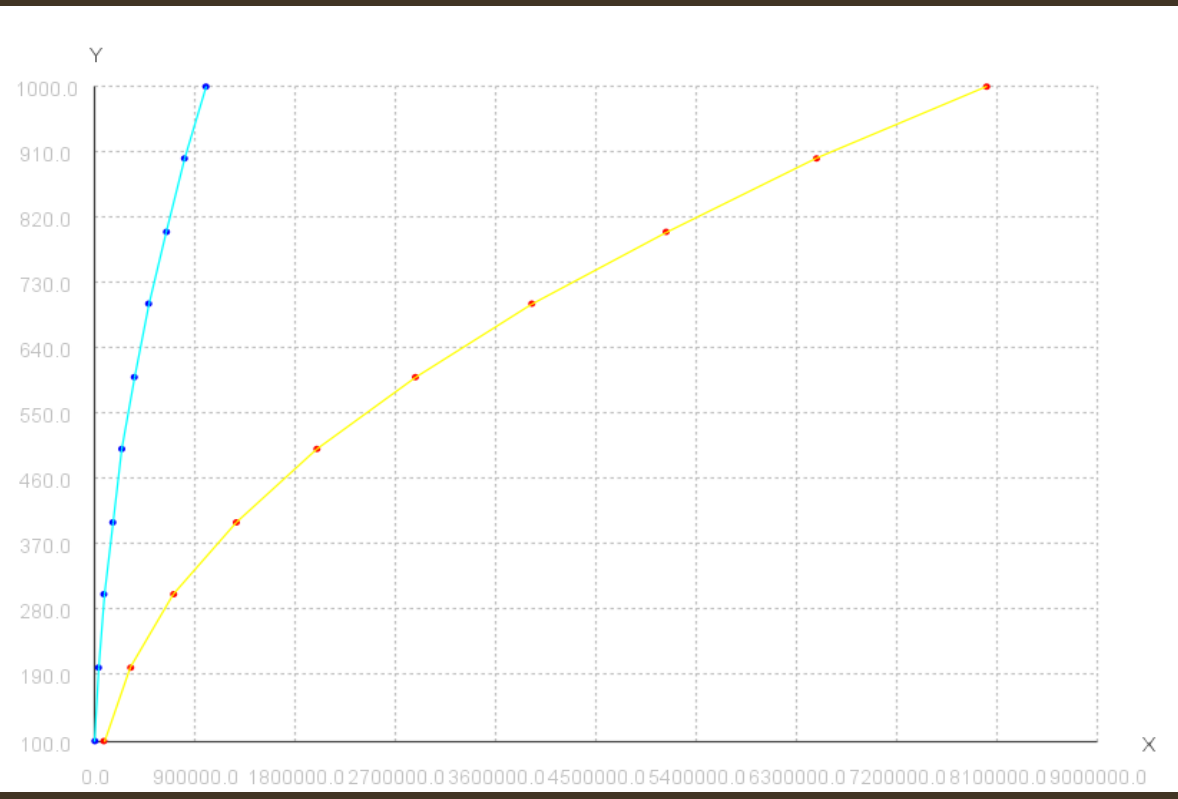
- Caso peor: siempre se ejecuta el if:

Poda= 9+nlogn+sumatorio(5+sumatorio(15)) =$8n^2$-2n+nlogn+19

3.Poda

- 3.Gráfica





4.Divide y Vencerás

```java
public static Parpuntos DyV ( Punto T[], int i, int d){
    Parpuntos l = null;
   int opc = d-i+1;
   if(opc>3){
       int m = (i+d)/2;

       Parpuntos di = DyV(T, i, d: m);
       Parpuntos dd = DyV(T, m+1, d);
       Parpuntos dmin = new Parpuntos();
       dmin.setDistancia(d: Double.min(a: di.distancia(), b: dd.distancia()));

       if(dmin.distancia() == di.distancia())
           dmin = di;
       else
           dmin = dd;
       int a = m;
       while ((a >= i) && ((T[m+1].getX() - T[a].getX()) < dmin.distancia())){
           a--;
       }
       int b = m+1;
       while ((b <= d) && ((T[b].getX() - T[m].getX()) < dmin.distancia())){
           b++;
       }
       l = Exhaustivo(p: T, a+1, b-1);
       if (l.distancia() < dmin.distancia())
               dmin = l;
       return dmin;

   }else{
       l=Exhaustivo(p: T, in: i, f: d);
       return l;
   }
```

El primero sumatorio va desde n/2 hasta 0 y el segundo desde la mitad del array hasta n.

- Caso medio:

OE(DyV) = 1+3+1+3+2*T(n/2)+1+5+1+1+11+Sumatorio(2)+2+11 +Sumatorio(2)+ $5n^2+10n+9$

DyV=2*T(n/2)+$4n^2+11n+49$

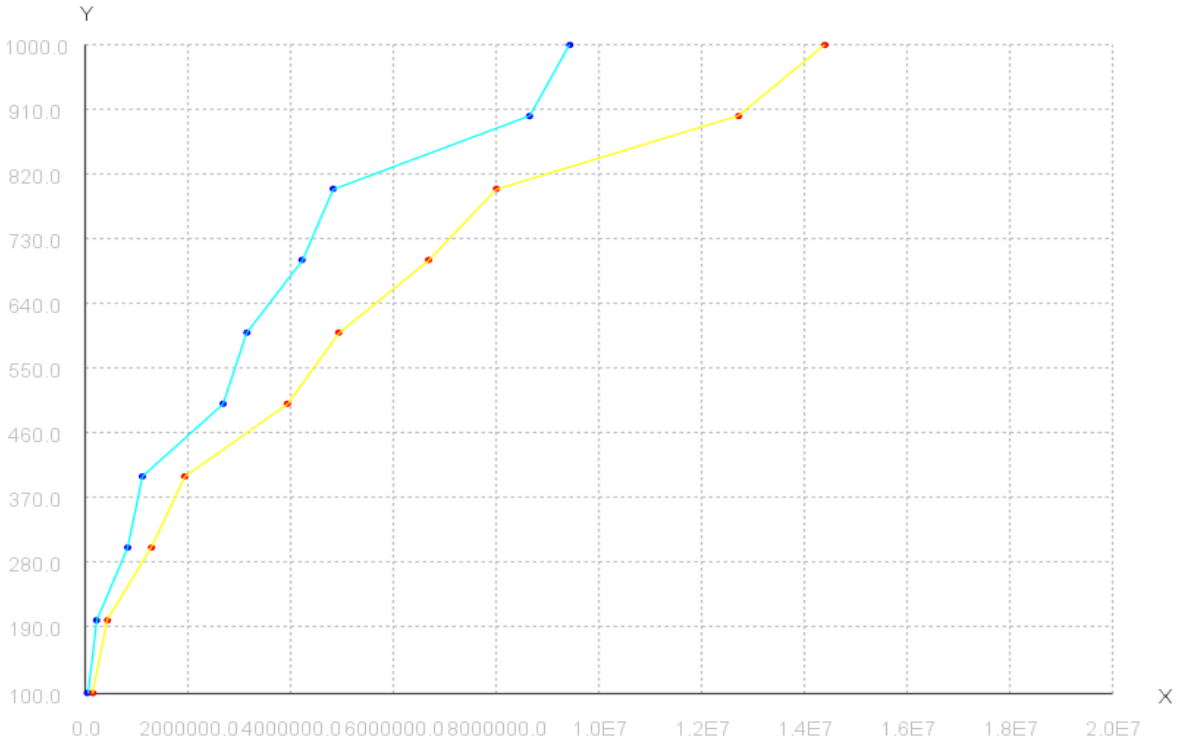
- Caso peor:

OE(DyV) = 1+3+1+3+2*T(n/2)+1+5+7+1+11+Sumatorio(2)+2+11 +Sumatorio(2)+ $10n^2+5n+9$

DyV=2*T(n/2)+$9n^2+6n+55$

4.Divide y Vencerás

- 3.Gráfica



5.DyV Mejorado

```
public static Parpuntos AlgoritmoDyVMejorado(Punto[] p, int in, int f) {

if (f - in <= 2) {
    Parpuntos l=Exhaustivo(p, in, f);
    return l;
}
int medio = (in + f) / 2;
Punto m = p[medio];
Parpuntos di = AlgoritmoDyVMejorado(p, in, f: medio);
Parpuntos dd = AlgoritmoDyVMejorado(p, medio + 1, f);
Parpuntos dmin=new Parpuntos(p1: dd.getP1(),p2: dd.getP2());
if(dmin.distancia() > di.distancia()){
    dmin.setLinea(p1: di.getP1(), p2: di.getP2());
}

List<Punto> puntos = new ArrayList<>();
for (int i = in; i <= f; i++) {
    if (Math.abs(p[i].getX() - m.getX()) < dmin.distancia()) {
        puntos.add(p[i]);
    }
}
puntos.sort(c: Comparator.comparingDouble(pu -> pu.getY()));
for (int i = 0; i < puntos.size(); i++) {
    for (int j = i + 1; j < puntos.size() && (puntos.get(index: j).getY() - puntos.get(index: i).getY()) < dmin.distancia(); j++) {
        Parpuntos l=new Parpuntos(p1: puntos.get(index: i), p2: puntos.get(index: j));
        double distancia = l.distancia();
        dmin.setCalculadas(dmin.getCalculadas()+2);
        if(dmin.distancia() > distancia){
            dmin.setLinea(p1: l.getP1(),p2: l.getP2());
        }
    }
}
return dmin;
}
```

El primero sumatorio va desde n/2 hasta 0 y el segundo desde la mitad del array hasta n.

- Caso medio:

OE(DyV) = 1+3+1+3+2*T(n/2)+1+5+1+1+11+Sumatorio(2)+2+11 +Sumatorio(2)+ $5n^2+10n+9$

DyV=2*T(n/2)+$4n^2+11n+49$

- Caso peor:

OE(DyV) = 1+3+1+3+2*T(n/2)+1+5+7+1+11+Sumatorio(2)+2+11 +Sumatorio(2)+ $10n^2+5n+9$

DyV=2*T(n/2)+$9n^2+6n+55$

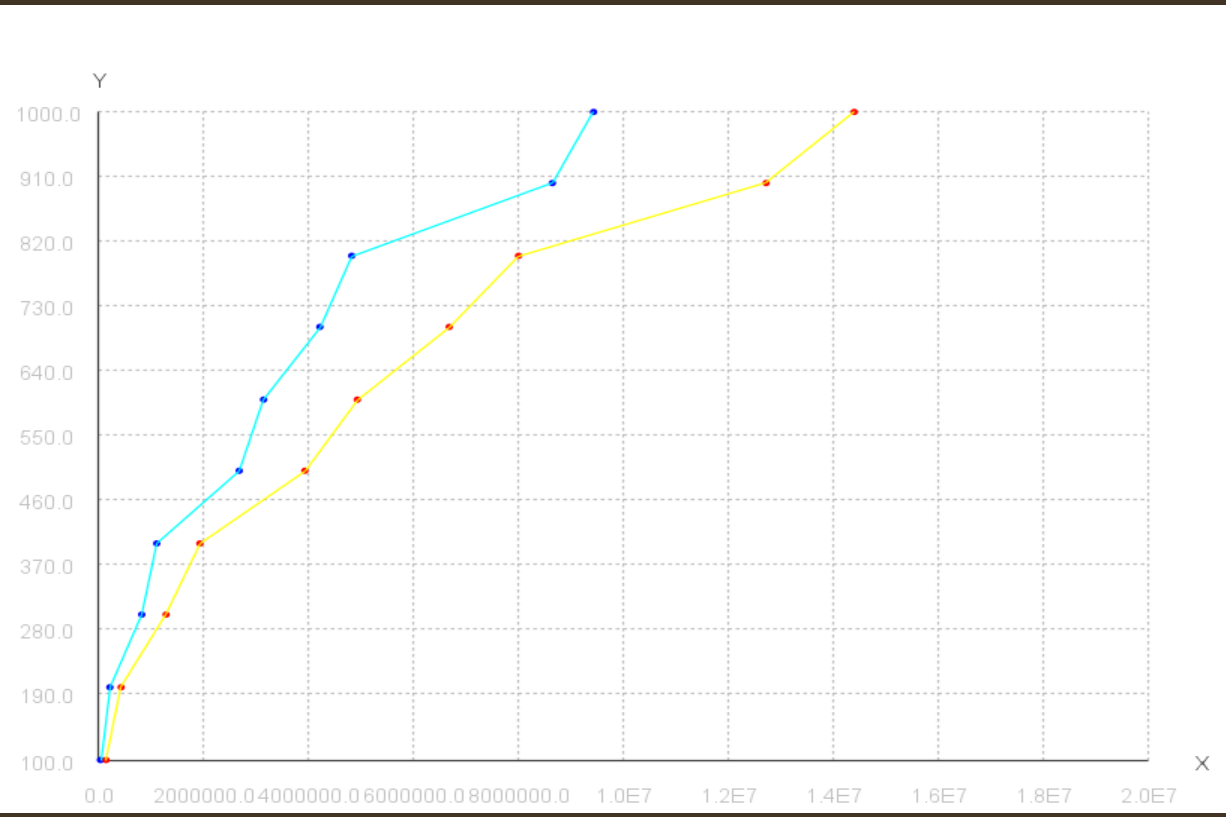
Finalmente nos queda:

Caso médio: O(n^2)

Caso Peor: O(n^2)

5.DyV Mejorado

- 3.Gráfica



6.Comparación de los algoritmos

	Exhaustivo	Poda	DyV	DyV Mejorado
Talla	Tiempo1	Tiempo2	Tiempo3	Tiempo4
1000	11.2591	3.3869	2.5962	8.5107
1500	23.9	5.2275	3.484	4.6191
2000	42.2242	3.7973	4.2206	9.8066
2500	67.0444	3.9287	5.3407	8.0621
3000	104.6591	4.3637	5.945	9.4944
3500	175.8197	5.4284	8.2182	12.3859
4000	221.0574	5.7466	8.3255	13.7553
4500	290.5022	6.8398	10.1053	17.0047
5000	356.5804	8.1018	11.22	17.5271
5500	448.7112	8.1319	11.8658	18.5141
6000	527.1281	9.3295	14.0461	21.745
6500	601.3457	9.9664	14.9129	22.5465
7000	761.5921	10.865	16.1729	25.0494
7500	863.5892	11.5113	17.6818	28.7178
8000	929.9849	12.8168	18.9316	30.0615
8500	1121.009	14.0716	21.1342	38.5563
9000	1302.9173	15.3022	21.6872	33.9596

7.Unidireccional

```java
public static Solucion unidireccional(Punto[] ciudades) {
    int n = ciudades.length;
    boolean[] visitadas = new boolean[n];
    double distancia=0;
    ArrayList<Punto> ruta = new ArrayList<>();
    Random rand = new Random(seed: System.currentTimeMillis());
    int ciudadInicial = 4;
    ruta.add(ciudades[ciudadInicial]);
    visitadas[ciudadInicial] = true;
    int ciudadMasCercana = ciudadInicial;
    for (int i = 0; i < n - 1; i++) {
        Punto puntoActual = ciudades[ciudadMasCercana];
        double distanciaMinima = Double.MAX_VALUE;
        ciudadMasCercana = -1;

        for (int j = 0; j < n; j++) {
            if (!visitadas[j]) {
                Parpuntos seg = new Parpuntos(p1: puntoActual, ciudades[j]);
                double d = seg.distancia();
                if (d < distanciaMinima) {
                    distanciaMinima = d;
                    distancia+=distanciaMinima;
                    ciudadMasCercana = j;
                }
            }
        }

        ruta.add(ciudades[ciudadMasCercana]);
        visitadas[ciudadMasCercana] = true;
    }
    ruta.add(ciudades[ciudadInicial]);

    Solucion s=new Solucion(dis:distancia,a: ruta);

    return s;
}
```

```
**************************UNIDIRECCIONAL**********************************

• berlin52: Tamaño 52 ciudades. Coste de la solución óptima: 72479.71704524364
• ch130: Tamaño 130 ciudades. Coste de la solución óptima: 97377.08560914642
• ch150: Tamaño 150 ciudades. Coste de la solución óptima: 110471.50736130221
• d493: Tamaño 493 ciudades. Coste de la solución óptima: 7015515.421850915
• d657: Tamaño 657 ciudades. Coste de la solución óptima: 7187646.461364204
```

```
Distancia berlin52: 72479.71704524364
************************************************************
************************************************************
Distancia ch130: 97377.08560914642
************************************************************
Distancia ch150: 110471.50736130221
************************************************************
Distancia d493: 7015515.421850915
************************************************************
Distancia d657: 7187646.461364204
************************************************************
```

8. Bidireccional

```java
public static Solucion bidireccional(Punto[] ciudades) {
    int n = ciudades.length;
    double distancia=0.0;
    boolean[] visitadas = new boolean[n];
    ArrayList<Punto> ruta = new ArrayList<>();
    Random rand = new Random(seed: System.currentTimeMillis());
    int ciudadInicial = rand.nextInt(bound: ciudades.length);
    int ciudadMasCercana = -1;
    ruta.add(ciudades[ciudadInicial]);
    for (int i = 0; i < visitadas.length; i++) {
        visitadas[i] = false;
    }
    visitadas[ciudadInicial] = true;
    for (int i = 0; i < ruta.size() && ruta.size() < n; i++) {

        double distanciaMinima = Double.MAX_VALUE;
        Punto puntoActual = ruta.get(index: i);
        for (int j = 0; j < n; j++) {
            if (!visitadas[j]) {
                Parpuntos seg = new Parpuntos(p1: puntoActual, ciudades[j]);
                double d = seg.distancia();

                if (d < distanciaMinima) {
                    distanciaMinima = d;
                    distancia+=distanciaMinima;
                    ciudadMasCercana = j;
                }
            }
        }

        ruta.add(ciudades[ciudadMasCercana]);
        visitadas[ciudadMasCercana] = true;
    }
    ruta.add(ciudades[ciudadInicial]);
    Solucion s=new Solucion(dis:distancia,a: ruta);
    return s;
```

- berlin52: Tamaño 52 ciudades. Coste de la solución óptima: 69115.42582022007
- ch130: Tamaño 130 ciudades. Coste de la solución óptima: 100773.31791136939
- ch150: Tamaño 150 ciudades. Coste de la solución óptima: 172275.35259062977
- d493: Tamaño 493 ciudades. Coste de la solución óptima: 7557539.128176931
- d657: Tamaño 657 ciudades. Coste de la solución óptima: 1.4084849666752031E7

```
Distancia berlin52: 69115.42582022007
**********************************************
**********************************************
Distancia ch130: 100773.31791136939
**********************************************
Distancia ch150: 172275.35259062977
**********************************************
Distancia d493: 7557539.128176931
**********************************************
Distancia d657: 1.4084849666752031E7
**********************************************
```