Rocío Barragán Moreno, Mario Casas Pérez, Ángela Maria Garrido Ruiz, Marina Ruiz Palomino D1

PRÁCTICA 2: ALGORITMOS DIVIDE Y VENCERÁS

1. Elaborar un método básico (no Divide y Vencerás) que resuelva el problema de identificar cuáles puntos, entre un conjunto de puntos dado, son no dominados. Analice su eficiencia teórica.

Para desarrollar el método básico hemos implementado dos clases, la clase Punto y la clase Conjunto_puntos.

En la clase Punto podemos encontrar los métodos básicos de una clase y el método domina que es el siguiente:

```
//Función que indica si a un punto le domina otro punto
bool Punto::ledomina(Punto otro){

    //En principio, el otro punto le va a dominar.
    bool ledomina = true;
    for (int i=0; i<coordenadas.size() &&ledomina; i++){

        //Con que una coordenada del punto ya sea mayor o igual que la otra
        //coordenada del otro punto, ya no le dominaría.
        if (coordenadas[i] >= otro.coordenadas[i])
            ledomina = false;
    }

    return ledomina;
}
```

En la clase Conjunto_puntos encontramos como en la anterior, los métodos básicos y el método aniade_dom que es el siguiente:

```
//Añade los puntos no dominados, al vector de no dominados
void Conjunto_puntos::aniade_dom(){
    bool no_dominado;
    for (int i=0; i<puntos.size(); i++){
        //Suponemos que no es dominado
        no_dominado = true;;
        for (int j=0; j<puntos.size() && no_dominado; j++){

            //Compara un punto con el resto de puntos del conjunto
            if (i!=j){
                //Con que uno de los puntos le domine, no_dominado sería false
                //ya que no sería un punto no dominado, pues hay otro que le domina
                if(puntos[i].ledomina(puntos[j]))
                    no_dominado = false;

            }
        }

        //Si ese punto ha mantenido a true el no_dominado al haberlo comparado
        //con el resto de puntos del conjunto, significa que ninguno de ellos
        //lo ha dominado, por tanto se añade al vector de no dominados (resultado)
        if(no_dominado)
            resultado.push_back(puntos[i]);
    }
}
```

Los métodos especificados son la clave para el funcionamiento de la práctica de estos hemos obtenido su eficiencia teórica obteniendo los siguientes resultados:

EFICIENCIA MÉTODO LEDOMINA

- Las 2 primeras sentencias son operaciones elementales → Eficiencia $O(1)$
- Bucle for
  - → Bloque de sentencias → Eficiencia $O(1)$ $F(n)$
  - → Evaluación de la condición → Eficiencia $O(1)$ $g(n)$
  - → El bucle se ejecuta n veces → Eficiencia $O(n)$ $h(n)$
  - → Actualización → Eficiencia $O(1)$ $a(n)$
  - → Inicialización → Eficiencia $O(1)$ $i(n)$

$$O(i(n) + g(n) + h(n) * (g(n) + F(n) + a(n)) =$$
$$= O(1 + 1 + n(1 + 1 + 1) = O(2 + 3n)$$

La eficiencia sería $O(n)$

- Las sentencias de después del for son también operaciones elementales → Eficiencia $O(1)$

Por tanto la eficiencia total de la función sería $\max(O(n), O(1), O(1))$

Eficiencia total del método no dominado ⇒ $O(n)$

EFICIENCIA MÉTODO ANIADE_DOM()

→ Bucle interno

$i(n) : O(1)$
$a(n) : O(1)$
$g(n) : O(1)$
$f(n) : O(1)$ Anidación del condicional
$h(n) : O(n)$

$$O(1+1+n(1+1+1) = O(2+3n) = O(n)$$

→ Bucle externo

$i(n) : O(1)$
$a(n) : O(1)$
$g(n) : O(1)$
$f(n) : \max(O(1), O(n)) : O(n)$
$h(n) : O(n)$

$$O(1+1+n(1+n+1)) = O(2+2n+n^2) = O(n^2)$$

Por tanto la eficiencia total de la función sería $\max(O(n), O(n^2))$

Eficiencia total del método aniade_dom() ⇒ $O(n^2)$

En el main adjunto en la carpeta de la entrega podemos ver como funciona el método básico de nuestro problema.

2. Estudiar si el problema puede ser abordado mediante la técnica Divide y Vencerás. Si es posible, piense en al menos dos estrategias para dividir el problema en subproblemas y fusionar sus soluciones. Seleccione una de ellas como la mejor, de forma justificada, y realice el diseño completo Divide y Vencerás. Analice su eficiencia teórica.

Si podría realizarse mediante el método DyV.

Esta técnica nos dice que es posible dividir un problema inicial en subproblemas independientes que tras resolverlos recursivamente pueden fusionar las soluciones para obtener la solución final.

En este problema los puntos pueden verse como entidades independientes por lo podemos hacer subconjuntos de dichos puntos para estudiarlos por independiente y después fusionar las soluciones de dichas partes.

A continuación vamos a exponer cómo realizaremos esa metodología.

- La primera técnica consiste en dividir el conjunto de puntos dados a la mitad, así tenemos dos subconjuntos que se pueden operar por separado. Cada subconjunto podrá ser dividido a la mitad recursivamente hasta llegar a un conjunto que consta de un único punto que sería nuestro caso base. Tras realizar recursivamente el método obtenemos los resultados parciales de ambas partes. Para el resultado final solo tendremos que combinar esas subsoluciones y así obtendremos el conjunto de puntos que vamos buscando de una manera más eficiente que en la implementación del ejercicio 1.

```
//Método DyV con el conjunto dividido a la mitad
vector<Punto> DominantesMitad(Conjunto_puntos conjunto, int inicio, int final){
    //Solo vamos a dividir el conjunto de puntos cuando tenga mas de una componente
    Conjunto_puntos c;
    if((final-inicio) == 2){
        cout << "Caso base " << endl;
        conjunto.aniade_dom();
        return conjunto.getResultado();
    }
    else if(final-inicio < 2)
        return c.getResultado();

    int medio = (inicio+final+1)/2; //calculamos la mitad del vector
    cout << "Inicio/medio/final " << inicio << " " << medio << " " << final << " " <<endl;
    c.addVector(DominantesMitad(conjunto,inicio,medio));//Buscamos los puntos en la mitad izda

    c.addVector(DominantesMitad(conjunto,medio+1,final));//Buscamos los puntos en la mitad dcha

    c.aniade_dom();
    return c.getResultado();

}
```

La eficiencia de esta función se ha calculado de la siguiente manera:

EFICIENCIA DOMINANTESMITAD

La ecuación resultante sería la siguiente:

$$\boxed{T(n) = 2T(n/2) + n + n^2}$$

Realizamos un cambio de variable $n = 2^m$

$$T(n) - 2T(n/2) = n + n^2$$

$$\underbrace{T(2^m) - 2T(2^{m-1})}_{\text{Parte homogénea}} = \underbrace{2^m + (2^m)^2}_{\text{Parte no homogénea}}$$

Parte homogénea: $T(2^m) - 2T(2^{m-1}) \Rightarrow (x-2)$

Parte no homogénea: $2^m + (2^m)^2$

$$\begin{cases} B_1 = 2 & q_1(n) = 1 & d_1 = 0 & (x-2) \\ B_2 = 4 & q_2(n) = 1 & d_2 = 0 & (x-4) \end{cases}$$

El polinomio característico de la recurrencia es:

$$g(x) = (x-2)^2 (x-4)$$

Por tanto $t(m) = c_1 \cdot 2^m \cdot m^0 + c_2 \cdot 2^m \cdot m^1 + c_3 \cdot 4^m \cdot m^0$

$$t(m) = c_1 \cdot 2^m + c_2 \cdot 2^m \cdot m + c_3 \cdot 4^m$$

Deshacemos el cambio $n = 2^m \rightarrow m = \log_2(n)$

$$t(n) = c_1 \cdot n + c_2 \cdot n \cdot \log_2(n) + c_3 \cdot n^2$$

La eficiencia en el peor caso es $O(n^2)$

- La segunda técnica a realizar será dividir el algoritmo de manera diferente: el total de puntos se dividirá entre 3 y comprobaremos si dominan a otros puntos o no.

```cpp
vector<Punto> DominantesTercios(Conjunto_puntos conjunto, int inicio, int final){
    Conjunto_puntos c;
    //Miro si el conjunto esta compuesto solo por dos componenetes
    if((final-inicio) == 2){ //si solo tiene dos estoy en el caso base
        cout << "Caso base " << endl;
        conjunto.aniade_dom(); //Miro cual es el dominante de los dos
        return conjunto.getResultado(); //devuelvo ese resultado
    }
    else if(final-inicio < 2) //si solo tengo un punto
        return c.getResultado(); //devuelvo ese punto

    //Si mi vector tiene mas de dos puntos
    int primerTercio = (inicio+final+1)/3; //calculamos el primer tercio del vector
    int segundoTercio = 2*primerTercio; //calculamos el segundo tercio

    c.addVector(DominantesTercios(conjunto,inicio,primerTercio));//Buscamos los puntos en la mitad izda
    c.addVector(DominantesTercios(conjunto,primerTercio+1,segundoTercio));//Buscamos los puntos del centro
    c.addVector(DominantesTercios(conjunto,segundoTercio+1,final));//Buscamos los puntos en la mitad dcha

    c.aniade_dom(); //saco los dominantes del resultado de agrupar las subsoluciones
    return c.getResultado(); //los devuelvo

}
```

La eficiencia de esta función se ha calculado de la siguiente manera:

EFICIENCIA DOMINANTESTERCIOS

La ecuación resultante sería la siguiente

$$\boxed{T(n) = 3\,T(n/3) + n + n^2}$$

Realizamos un cambio de variable

$$n = 3^m$$

$$T(n) - 3T(n/3) = n + n^2$$

$$\underbrace{T(3^m) - 3(3^{m-1})}_{\text{Parte homogénea}} = \underbrace{3^m + (3^m)^2}_{\text{Parte no homogénea}}$$

Parte homogénea

$$T(3^m) - 3(3^{m-1}) \rightarrow (x-3)$$

Parte no homogénea

$$3^m + (3^m)^2 \left|\begin{array}{lll} B_1 = 3 & g_1(n) = 1 & d_1 = 0 \\ B_2 = 9 & g_2(n) = 1 & d_1 = 0 \end{array}\right.$$

El polinomio característico de la recurrencia es

$$g(x) = (x-3)^2(x-9)$$

Deshacemos el cambio

$$t_n = c_1 \cdot 3^m \cdot m^0 \cdot c_2 \cdot 3^m \cdot m^1 + c_2 \cdot 9^m \cdot m^0$$

$$t_n = c_1 \cdot n + c_2 \cdot n \log_3(n) + c_2 \cdot n^2$$

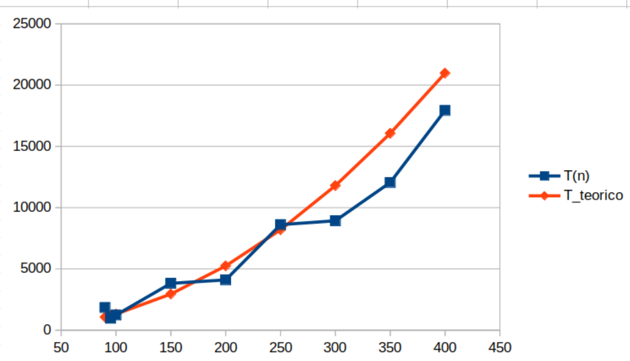
La eficiencia en el peor caso es $O(n^2)$

Finalmente nos hemos decantado por el algoritmo que divide a la mitad los dos conjuntos de puntos ya que ambos algoritmos nos presentan la misma eficiencia. Como podemos observar el método divide y vencerás por si solo es mas eficiente que el básico ya que su eficiencia tiende a ser nlog(n) es al añadirle la función aniade_dom cuando hacemos que su eficiencia crezca.

3. Implemente los algoritmos básicos y Divide y Vencerás. Resuelva el problema del umbral de forma experimental. Aunque los algoritmos deben ser generales y permitir cualquier valor de K, para la práctica asume siempre un valor de K=10.
El algoritmo básico y el DyV escogido están explicados en los ejercicios 1 y 2.
Vamos a proceder a lanzar ejecuciones con distintos tamaños de casos y medir los tiempos que tardan los algoritmos en sacar los resultados.
Cuando tengamos estos tiempos vamos a calcular la constante K y determinaremos un umbral. Este umbral nos indicará para qué tamaños de caso de uso es más conveniente usar el método básico y para cuales el DyV.
El cálculo del T(n), que es el tiempo empírico, lo hemos calculado probando con distintos tamaños de vector y midiendo los tiempos.
El tiempo teorico lo hemos calculado como la constante K multiplicada por la eficiencia del algoritmo -> K*f(n).
La constante K ha sido calculada como T(n)/f(n)

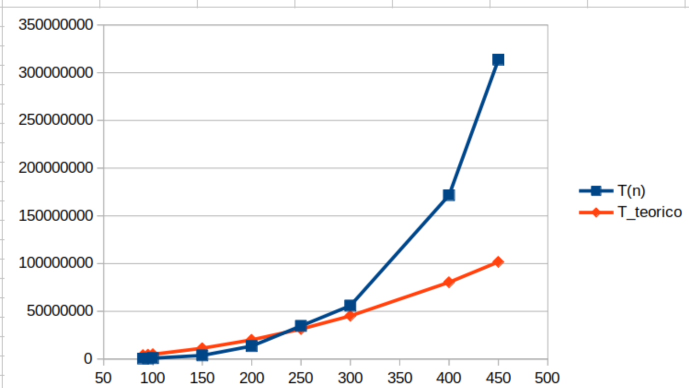
- **Tabla y gráfica del método básico**

n	T(n)	T_teorico	K
90	1849	1061,887065	0,228271605
95	973	1183,151946	0,107811634
100	1237	1310,971686	0,1237
150	3824	2949,686293	0,169955556
200	4105	5243,886743	0,102625
250	8611	8193,573035	0,137776
300	8930	11798,74517	0,099222222
350	12054	16059,40315	0,0984
400	17938	20975,54697	0,1121125
		K_media	**0,131097169**

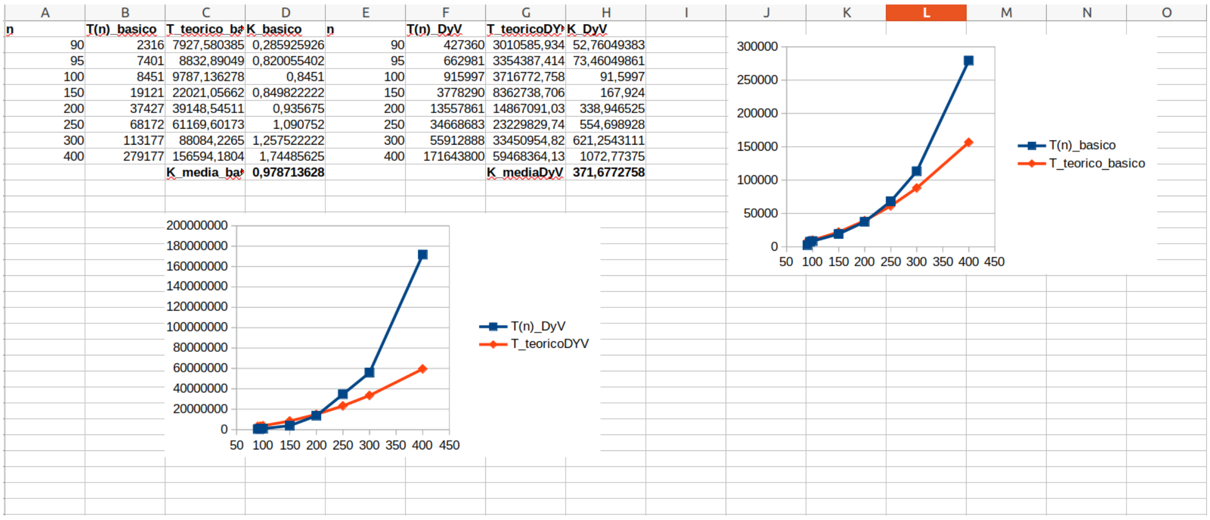


- **Tabla y gráfica del DyV**

n	T(n)	T_teorico	K
90	427360	4069971,275	52,76049383
95	662981	4534751,945	73,46049861
100	915997	5024655,895	91,5997
150	3778290	11305475,76	167,924
200	13557861	20098623,58	338,946525
250	34668683	31404099,34	554,698928
300	55912888	45221903,05	621,2543111
400	171643800	80394494,32	1072,77375
450	313626350	101749281,9	1548,772099
		K_media	**502,4655895**



- **Gráfica comparativa de ambos métodos**



A	B	C	D	E	F	G	H
n	T(n)_basico	T_teorico_ba	K_basico	n	T(n)_DyV	T_teoricoDY	K_DyV
90	2316	7927,580385	0,285925926	90	427360	3010585,934	52,76049383
95	7401	8832,89049	0,820055402	95	662981	3354387,414	73,46049861
100	8451	9787,136278	0,8451	100	915997	3716772,758	91,5997
150	19121	22021,05662	0,849822222	150	3778290	8362738,706	167,924
200	37427	39148,54511	0,935675	200	13557861	14867091,03	338,946525
250	68172	61169,60173	1,090752	250	34668683	23229829,74	554,698928
300	113177	88084,2265	1,257522222	300	55912888	33450954,82	621,2543111
400	279177	156594,1804	1,74485625	400	171643800	59468364,13	1072,77375
		K_media_ba	0,978713628			K_mediaDyV	371,6772758

Hemos modelado las gráficas por separado para la mejoría de su estudio, como podemos observar con estos tamaños de caso el comportamiento del método básico es mucho mejor, ya que el tiempo teórico y empírico se asemejan mucho más que en el caso DyV. Por lo tanto podríamos determinar el umbral en ese punto y tener unos rangos en los que la función DyV debe de entrar en juego que como vemos deben ser tamaños de caso mayores para que merezca la pena aplicar dicho método.

En la carpeta con la entrega se encuentran las hojas de excel empleadas para hacer las gráficas.