UNIVERSIDAD DE GRANADA

Algorítmica

2º Grado en Ingeniería Informática Y ADE

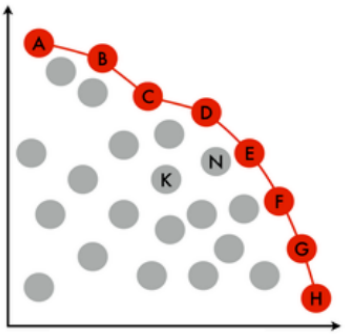
PRÁCTICA 2: Algoritmo Divide y Vencerás

ÍNDICE

INTRODUCCIÓN

Tal y como vimos en la práctica anterior, la eficiencia de un programa es algo muy importante que se debe tener en cuenta a la hora de realizar un proyecto.

Vamos a trabajar con la técnica de Divide y Vencerás, que consiste en dividir el problema en otros más pequeños y de una más fácil resolución, para acabar juntando todo lo obtenido.

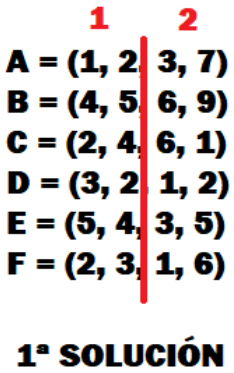


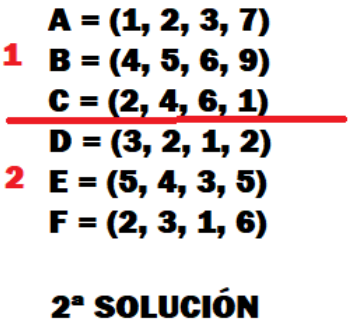
En esta práctica se nos ha planteado elaborar un algoritmo para buscar los puntos dominantes de un conjunto (frente pareto).

Primero, deberemos realizarlo sin tener en cuenta la eficiencia, con un método básico o de Fuerza Bruta. Además, analizaremos su eficiencia teórica.

En segundo lugar, vamos a resolver el mismo problema a través de la utilización del algoritmo de Divide Y Vencerás, para el cual, vamos a plantear dos soluciones.

La **primera solución**, sería dividir el problema en k/2, es decir, dividir las coordenadas de cada punto a la mitad, y la **segunda solución** sería dividirlo en n/2, es decir, dividir el número total de puntos a la mitad. Implementaremos, tras un análisis, la solución que consideremos más eficiente.





Compararemos las eficiencias, tanto las teóricas como las prácticas, de ambos algoritmos para ver cómo de bueno es el Divide y Vencerás frente a la Fuerza Bruta, y obtendremos una serie de conclusiones finales.

FUERZA BRUTA (EJERCICIO 1)

En este apartado, vamos a plantear cómo haríamos el algoritmo de fuerza bruta, es decir, sin dividir el problema en subproblemas. Después, vamos a poner un ejemplo sencillo que nos ayudó a comprender bien cómo funcionaba el algoritmo y, finalmente, vamos a analizar su eficiencia teórica.

DESARROLLO (Pseudocódigo)

Utilizaremos una estructura, denominada **Punto**, la cual será un *vector de double* de la biblioteca STL.

esDominante:

Función esDominante(const Punto &p1, const Punto &p2):

// Inicializamos dos variables booleanas para verificar si p1 es mayor o igual y estrictamente mayor que p2

mayorOIgual <- verdadero
estrictamenteMayor <- falso

// Iteramos sobre todas las coordenadas del punto p1 y comparamos con las correspondientes coordenadas en p2

Para i <- 0 hasta longitud(p1.coords) - 1:

// Si alguna coordenada de p1 es menor que la correspondiente coordenada en p2, entonces p1 no es mayor o igual a p2

Si p1.coords[i] < p2.coords[i] entonces:
retornar **falso**
Fin Si

// Si alguna coordenada de p1 es mayor que la correspondiente coordenada en p2, entonces p1 es estrictamente mayor que p2

Si p1.coords[i] > p2.coords[i] entonces:
estrictamenteMayor <- verdadero
Fin Si

Fin Para
// Si p1 es mayor o igual a p2 y p1 es estrictamente mayor que p2, entonces p1 es dominante sobre p2

retornar **mayorOIgual y (and) estrictamenteMayor**

Fin Función

puntosDominantes:

Función puntosDominantes(vector<Punto> &puntos):
// Creamos un vector vacío para almacenar los puntos dominantes

dominantes <- vector vacío

// Iteramos sobre todos los puntos

Para i <- 0 hasta longitud(puntos) - 1:

// Inicializamos una variable para verificar si el punto actual es dominante
esDominanteActual <- verdadero

// Iteramos sobre todos los demás puntos y verificamos si el punto actual es dominante
Para j <- 0 hasta longitud(puntos) - 1 y esDominanteActual:

// Si encontramos un punto dominante, actualizamos la variable
Si i != j y esDominante(puntos[j], puntos[i]) entonces:
esDominanteActual <- falso
Fin Si
Fin Para

//Si el punto actual es dominante, lo agregamos al vector de puntos dominantes

Si esDominanteActual entonces:
agregar puntos[i] al final de dominantes

Fin Si

Fin Para

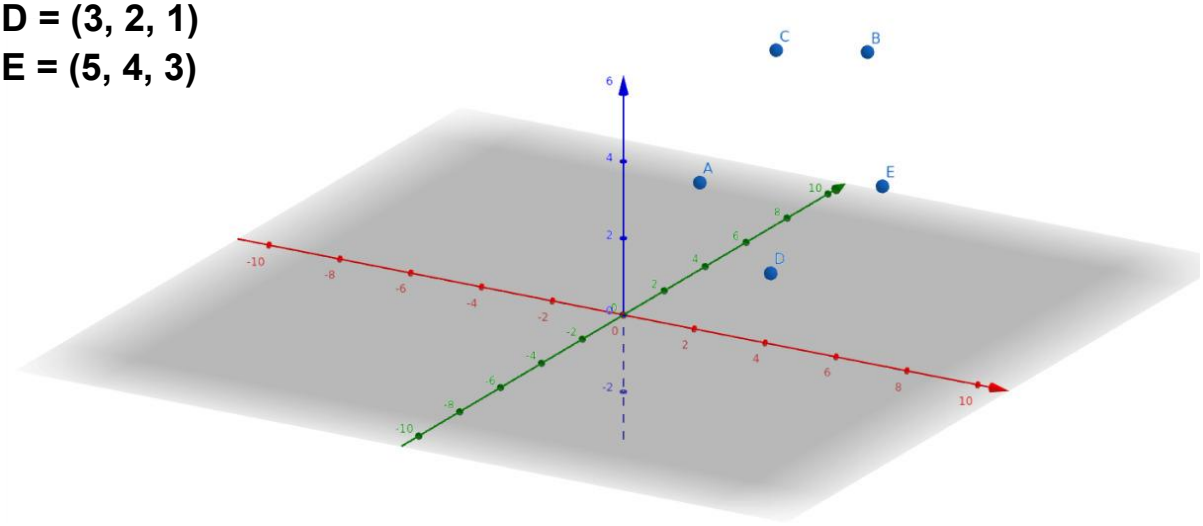
// Retornamos el vector de puntos dominantes
retornar dominantes

Fin Función

EJEMPLO

Hemos usado un ejemplo con **K = 3**, para entender mejor el funcionamiento de este programa, ya que viéndolo de forma gráfica era mucho más sencillo ver a simple vista cuáles eran los puntos dominantes y ver que la salida de nuestro ejercicio era la correcta.

A = (1, 2, 3)
B = (4, 5, 6)
C = (2, 4, 6)
D = (3, 2, 1)
E = (5, 4, 3)



"Los puntos dominantes son el B y el E"

EFICIENCIA TEÓRICA

Vamos a analizar la eficiencia teórica de los métodos “**esDominante**” y “**puntosDominantes**” que tendrá nuestro programa.

Respecto al método **“esDominante”** obtenemos una eficiencia O(K), siendo K el número de coordenadas de las que está formada cada punto.

Dicho análisis se realiza línea por línea de la siguiente manera:

```
Funcion esDominante(const Punto &p1, const Punto &p2): // O(K)
siendo K el tamaño de coordenadas

mayorOIgual <- verdadero //O(1)
estrictamenteMayor <- falso //O(1)

Para i <- 0 hasta longitud(p1.coords) - 1:  // O(K)

Si p1.coords[i] < p2.coords[i] entonces://O(1)
retornar falso //O(1)
Fin Si

Si p1.coords[i] > p2.coords[i] entonces: //O(1)
estrictamenteMayor <- verdadero //O(1)
Fin Si

Fin Para

retornar mayorOIgual y (and) estrictamenteMayor

 Fin Funcion
```

Por tanto, el método **“puntosDominantes”** nos quedaría una eficiencia O(N^2 * K), siendo N el número de puntos que se encuentran en el vector de puntos, y siendo K el número de coordenadas que contiene cada punto.

Dicho análisis se realiza línea por línea:

```
Funcion puntosDominantes(vector<Punto> &puntos): // O(N² * K)
 dominantes <- vector vacío //O(1)

 Para i <- 0 hasta longitud(puntos) - 1: //O(N)

 esDominanteActual <- verdadero //O(1)

Para j <- 0 hasta longitud(puntos) - 1 y esDominanteActual: //O(N)

Si i != j y esDominante(puntos[j], puntos[i]) entonces: //O(K)
esDominanteActual <- falso //O(1)
Fin Si
 Fin Para

Si esDominanteActual entonces: //O(1)
agregar puntos[i] al final de dominantes

Fin Si

Fin Para

 retornar dominantes

Fin Funcion
```

Nuestro método de fuerza bruta realizado para buscar los puntos dominantes de un conjunto tiene una eficiencia teórica de:

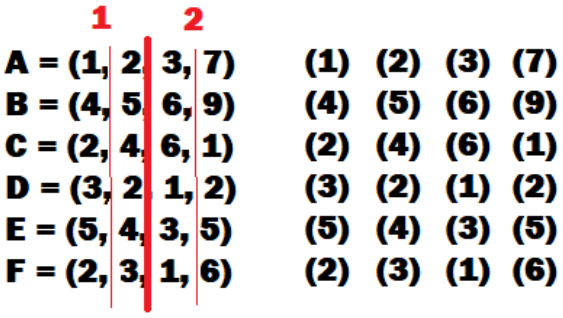
$$O(N^2 * K)$$

DIVIDE Y VENCERÁS (EJERCICIO 2)

ESTRATEGIAS POSIBLES

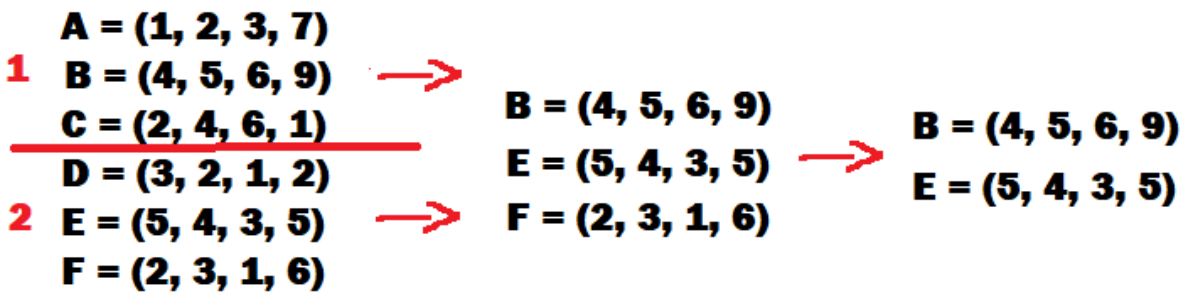
Estrategia I: Dividir en K/2

1. Sea K la dimensión de todos los puntos.
2. Seleccionamos un valor d con d = K/2
3. Dividimos el conjunto de puntos en dos subconjuntos, uno con los puntos con las coordenadas de la 1 a la d, y otro de la d+1 a K
4. Resolvemos el problema de los puntos dominantes para cada subconjunto dado haciendo uso de la recursividad.
5. Combinamos las soluciones obtenidas de los subconjuntos (eliminando los puntos dominados por otros puntos en la solución combinada).



Estrategia II: Dividir en N/2

1. Sea N el número total de puntos.
2. Dividimos el conjunto de puntos en dos subconjuntos, de manera que cada subconjunto contenga aproximadamente la mitad de los puntos.
3. Resolvemos el problema de los puntos dominantes para cada subconjunto empleando de nuevo la recursividad.
4. Combinamos las soluciones que hemos obtenido en cada caso (eliminando los puntos dominados por otros puntos en la solución combinada) para obtener el conjunto final de puntos dominantes.



SELECCIÓN DE LA MEJOR: N/2

En el caso de nuestro problema de encontrar los puntos dominantes, la división por la mitad de puntos (N/2) es más eficiente, porque se tiene en cuenta los conjuntos de puntos totalmente definidos.

Este método funciona bien en problemas donde las soluciones se pueden dividir fácilmente en partes iguales y las características de los datos son uniformes.

Si se divide el conjunto de puntos por la mitad, es probable que se reduzca significativamente el número de puntos que deben ser comparados para determinar si un punto domina a otro.

Por otro lado, si dividimos por la mitad las coordenadas de cada punto puede no ser tan eficiente en términos de reducción del número de comparaciones necesarias debido a que aunque la primera coordenada de un punto sea mayor que la primera coordenada de otro punto, eso no quiere decir que lo domine, ya que puede que comparando el resto de coordenadas lleguemos a la conclusión de que el segundo punto tenga alguna coordenada mayor que el primero y, por tanto, ningún punto dominaría al otro.

Además, el dividirlo por N/2, nos facilitará que se pueda hacer muchas más veces la recurrencia y, por tanto, nuestro algoritmo será más eficiente. Ya que si, por ejemplo, nosotros tenemos 252 puntos de 6 coordenadas, podremos dividirlo en dos de 126, estos, a su vez, en cuatro de 63 y así sucesivamente. Sin embargo, si dividieramos por coordenadas, al tener solo 6, en pocas divisiones llegaríamos al límite y no podríamos hacer tanta recursividad.

Por todos estos motivos, el método que es mucho más eficiente para el programa que se nos plantea de buscar los puntos dominantes es el de división por la mitad de los puntos, debido a que reduce considerablemente el número de comparaciones necesarias para determinar si un punto domina a otro.

DESARROLLO (Pseudocódigo)

Utilizaremos una estructura, denominada **Punto,** la cual será un *vector de double* de la biblioteca STL.

esDominante:

// Comprueba si p1 domina a p2

Función esDominante(const Punto &p1, const Punto &p2):

// Inicializamos dos variables booleanas para verificar si p1 es mayor o igual y estrictamente mayor que p2

mayorOIgual <- verdadero
estrictamenteMayor <- falso

// Iteramos sobre todas las coordenadas del punto p1 y comparamos con las correspondientes coordenadas en p2

Para i <- 0 hasta longitud(p1.coords) - 1:

// Si alguna coordenada de p1 es menor que la correspondiente coordenada en p2, entonces p1 no es mayor o igual a p2

Si p1.coords[i] < p2.coords[i] entonces:
retornar **falso**
Fin Si

// Si alguna coordenada de p1 es mayor que la correspondiente coordenada en p2, entonces p1 es estrictamente mayor que p2

Si p1.coords[i] > p2.coords[i] entonces:
estrictamenteMayor <- verdadero
Fin Si

Fin Para

// Si p1 es mayor o igual a p2 y p1 es estrictamente mayor que p2, entonces p1 es dominante sobre p2

retornar **mayorOIgual y (and) estrictamenteMayor**

Fin Función

```
// Combina las soluciones parciales de los subproblemas

Funcion combinar(&puntosIzquierda, &puntosDerecha)

//Crear un vector de puntos vacío

   combinado = []
//Verificar si los puntos de izquierda dominan a los de derecha

   Para cada puntoIzquierda en puntosIzquierda
      esDominanteActual = verdadero
      Para cada puntoDerecha en puntosDerecha
         Si esDominante(puntoDerecha, puntoIzquierda) entonces
            esDominanteActual = falso
            salir del bucle interno
         Fin Si
      Fin Para
      Si esDominanteActual entonces
         agregar puntoIzquierda a combinado
      Fin Si
   Fin Para

//Verificar si los puntos de derecha dominan a los de izquierda

   Para cada puntoDerecha en puntosDerecha
      esDominanteActual = verdadero
      Para cada puntoIzquierda en puntosIzquierda
         Si esDominante(puntoIzquierda, puntoDerecha) entonces
            esDominanteActual = falso
            salir del bucle interno
         Fin Si
      Fin Para
      Si esDominanteActual entonces
         agregar puntoDerecha a combinado
      Fin Si
   Fin Para

//Devuelve el vector combinado el cual contiene los puntos dominantes

   retornar combinado
Fin Función
```

```
// Encuentra los puntos dominantes usando divide y vencerás
funcion puntosDominantesDivideYVenceras(&puntos, inicio, fin)
    Si inicio es igual a fin entonces
        devuelve vector con un solo elemento: [puntos[inicio]]
    Fin Si

    medio = (inicio + fin) / 2
    puntosIzquierda = puntosDominantesDivideYVenceras(puntos, inicio, medio)

    puntosDerecha = puntosDominantesDivideYVenceras(puntos, medio + 1, fin)

    devuelve combinar(puntosIzquierda, puntosDerecha)

Fin Función


Funcion puntosDominantes(&puntos) devuelve vector de Punto
    Si puntos esta vacío entonces
        devuelve puntos
    Fin Si
    devuelve puntosDominantesDivideYVenceras(puntos, 0, tamaño de puntos - 1)
Fin Función
```

Eficiencia teórica:

La eficiencia teórica de un algoritmo es una medida de cuánto tiempo tarda en ejecutarse en términos de la entrada de datos que recibe. En el caso del algoritmo **puntosDominantesDivideYVenceras**, su eficiencia teórica se puede calcular utilizando ecuaciones de recurrencia.

La ecuación de recurrencia para este algoritmo se obtiene al dividir el vector de entrada en dos subvectores de tamaño n/2 y luego combinar las soluciones parciales de los subproblemas en tiempo O(n^2 * k). La ecuación de recurrencia es T(n) = 2 * T(n/2) + O(n^2 * k), donde T(n) representa el tiempo de ejecución de la función **puntosDominantesDivideYVenceras** en un vector de tamaño n.

T(N) = 2T(N/2) + O(N^2*K)

Calculamos la eficiencia para la parte izquierda de la suma y aplicamos la propiedad de la suma

$$T(2^K) = 2T(2^{(K-1)}) + O(N^2 * K)$$
$$(x-2) = 0$$
$$tk = c1 * 2^K = c1*N = O(N)$$

$$max(O(N), O(N^2 *K)) = O(N^2 *K)$$

En resumen, la eficiencia teórica de **puntosDominantesDivideYVenceras** se calcula mediante ecuaciones de recurrencia y se obtiene que es:

$$O(N^2 * K)$$

(Ejercicio 3)

Implementación método Fuerza Bruta

Para solucionar este problema, nos hemos decantado por utilizar una estructura denominada “Punto”, que será un vector de doubles, de la biblioteca STL. Para dicha estructura, hemos desarrollado dos funciones, “**esDominante**” y “**puntosDominantes**”.

La función “**esDominante”** se basa en comparar dos puntos para saber si un punto domina a otro. Para ello, debemos de tener en cuenta que un punto domina a otro si todas sus coordenadas (k) son **mayores o iguales** que las del otro punto, además de tener una coordenada **estrictamente mayor.**

Para su implementación, hemos utilizado dos variables booleanas, una para comprobar si cumple la condición de que todas las coordenadas son mayores o iguales y, otra para comparar si hay una coordenada estrictamente mayor. Dicho método devuelve un **and** de ambas variables booleanas, debido a que sólamente devolverá verdadero si ambas variables son verdaderas.

La función “**puntosDominantes”** se basa en recorrer el vector de puntos, para ir comparando punto por punto cada coordenada. Para ello, hemos determinado una variable booleana, **inicializada a true**, que usaremos para determinar cuando el punto es dominante.

Nos fijamos un punto con el primer bucle for para compararlo con el resto de puntos, ponemos la condición en el if de que i != j, para no compararlo con el mismo punto.

Posteriormente, vemos si hay algún punto que lo domine (>), en el caso de que *ninguno* lo domine, el bool **seguirá en true** y meteremos ese punto en el vector solución de dominantes; en caso de que *alguno* lo domine, **pasará a ser false** y se saldrá del bucle por la condición de parada, para seguir comprobando el resto de puntos.

CÓDIGO FUERZA BRUTA

```
#include <iostream>
#include <vector>

using namespace std;

struct Punto {
    vector<double> coords;
};

bool esDominante(const Punto &p1, const Punto &p2) {
    bool mayorOIgual = true;
    bool estrictamenteMayor = false;
    for (int i = 0; i < p1.coords.size(); i++) {
        if (p1.coords[i] < p2.coords[i]) {
            return false;
        }
        if (p1.coords[i] > p2.coords[i]) {
            estrictamenteMayor = true;
        }
    }
    return mayorOIgual and estrictamenteMayor;
}

vector<Punto> puntosDominantes(vector<Punto> &puntos) {
vector<Punto> dominantes;
    for (int i = 0; i < puntos.size(); i++) {
        bool esDominanteActual = true;
        for (int j = 0; j < puntos.size() && esDominanteActual; j++) {

            if (i != j && esDominante(puntos[j], puntos[i])) {
                esDominanteActual = false;
            }
        }
        if (esDominanteActual) {
            dominantes.push_back(puntos[i]);
        }
    }
    return dominantes;
}
```

```cpp
int main() {
    //EJEMPLO INT CON K=10
    Punto p1 = {{1, 2, 3, 6, 1, 5, 4, 1, 8, 2}};
    Punto p2 = {{4, 5, 6, 7, 2, 2, 1, 8, 5, 3}};
    Punto p3 = {{2, 4, 6, 4, 2, 1, 1, 7, 3, 1}};
    Punto p4 = {{3, 2, 1, 7, 1, 4, 2, 1, 9, 2}};
    Punto p5 = {{5, 4, 3, 8, 3, 6, 4, 2, 9, 3}};

    vector<Punto> puntos = {p1, p2, p3, p4, p5};
    vector<Punto> dominantes = puntosDominantes(puntos);

    cout << "Puntos dominantes: " << endl;
    for (int i = 0; i < dominantes.size(); i++) {
        cout << "(";
        for (int j = 0; j < dominantes[i].coords.size(); j++) {
            cout << dominantes[i].coords[j];
            if (j < dominantes[i].coords.size() - 1) {
                cout << ", ";
            }
        }
        cout << ")" << endl;
    }

    return 0;
}
```

Salida del código:

```
/home/flo/Desktop/ALG/ALG_PRACTICA_2/cmake-build-debug/ALG_PRACTICA_2
Puntos dominantes:
(4, 5, 6, 7, 2, 2, 1, 8, 5, 3)
(5, 4, 3, 8, 3, 6, 4, 2, 9, 3)

Process finished with exit code 0
```

Implementación método DyV

- Para solucionar este problema, nos hemos decantado por utilizar una estructura denominada “**Punto**”, que será un vector de doubles, de la biblioteca STL. Para dicha estructura, hemos desarrollado cuatro funciones, “**esDominante**”, “**combinar**”, “**puntosDominantesDivideyVenceras**” y “**puntosDominantes**”.

- Crear una función llamada "**esDominante**" que tome como parámetros dos puntos y devuelva un valor booleano que indique si el primer punto domina al segundo.

- La función **"esDominante"** compara las coordenadas de ambos puntos y devuelve verdadero si todas las coordenadas del primer punto son mayores o iguales que las del segundo y al menos una coordenada es estrictamente mayor

- La función "**combinar**" toma dos vectores de puntos (puntosIzquierda y puntosDerecha) y devuelve un nuevo vector de puntos (combinado) que contiene aquellos puntos que no son dominados por ningún otro punto en ninguno de los dos vectores originales. Para lograr esto, primero verifica si los puntos de la izquierda dominan a los de la derecha, y luego verifica si los puntos de la derecha dominan a los de la izquierda. Si un punto no es dominado por ningún otro punto en ninguno de los dos vectores originales, se agrega al vector combinado.

- Crear una función llamada "**puntosDominantesDivideYVenceras**" que tome como parámetros un vector de puntos, el índice de inicio y el índice de fin, y devuelva un vector con los puntos dominantes.

- La función "**puntosDominantesDivideYVenceras**" implementa el algoritmo Divide y Vencerás para encontrar los puntos dominantes de un conjunto de puntos en un espacio K dimensional. El algoritmo divide recursivamente el conjunto de puntos en dos mitades hasta llegar a un caso base de una sola punto. Luego, se combina la lista de puntos dominantes de la mitad izquierda con la de la mitad derecha mediante la función "**combinar**", la cual verifica si cada punto en una mitad domina a los puntos en la otra mitad. Finalmente, se retorna la lista de puntos dominantes resultante.

- Crear una función llamada "**puntosDominantes**" que tome como parámetro un vector de puntos y que devuelva otro vector con los puntos dominantes usando la técnica de "divide y vencerás".

- En el programa principal, crear los puntos con sus respectivas coordenadas y almacenarlos en un vector.

CÓDIGO DyV

```
Desarrollo de la estrategia II:
//Comprueba si p1 domina a p2
bool esDominante(const Punto &p1, const Punto &p2) {
   bool mayorOIgual = true;
   bool estrictamenteMayor = false;
   for (int i = 0; i < p1.coords.size(); ++i) {
       if (p1.coords[i] < p2.coords[i]) {
           return false;
       }
       if (p1.coords[i] > p2.coords[i]) {
           estrictamenteMayor = true;
       }
   }
   return mayorOIgual and estrictamenteMayor;
}

//Combina las soluciones parciales de los subproblemas
vector<Punto> combinar(vector<Punto>& puntosIzquierda, vector<Punto>&
puntosDerecha) {
      vector<Punto> combinado;

      //Verificar si los puntos de izquierda dominan a los de derecha
      for (int i = 0; i < puntosIzquierda.size(); ++i) {
       bool esDominanteActual = true;
       for (int j = 0; j < puntosDerecha.size(); ++j) {
            if (esDominante(puntosDerecha[j], puntosIzquierda[i])) {
                  esDominanteActual = false;
                  break;
            }
       }
        if (esDominanteActual) {
            combinado.push_back(puntosIzquierda[i]);
         }
      }

//Verificar si los puntos de derecha dominan a los de izquierda
      for (int i = 0; i < puntosDerecha.size(); ++i) {
        bool esDominanteActual = true;
        for (int j = 0; j < puntosIzquierda.size(); ++j) {
            if (esDominante(puntosIzquierda[j], puntosDerecha[i])) {
                  esDominanteActual = false;
                  break;
            }
```

```
        }
        if (esDominanteActual) {
            combinado.push_back(puntosDerecha[i]);
        }
      }

      return combinado;
}

//Encuentra los puntos dominantes usando divide y vencerás
vector<Punto> puntosDominantesDivideYVenceras(vector<Punto>& puntos, int
inicio, int fin) {
      if (inicio == fin) {
      return vector<Punto>(1, puntos[inicio]);
      }

      int medio = (inicio + fin) / 2;
      vector<Punto> puntosIzquierda =
puntosDominantesDivideYVenceras(puntos, inicio, medio);
      vector<Punto> puntosDerecha =
puntosDominantesDivideYVenceras(puntos, medio + 1, fin);
      return combinar(puntosIzquierda, puntosDerecha);
}

vector<Punto> puntosDominantes(vector<Punto>& puntos) {
      if (puntos.empty()) {
      return puntos;
      }
      return puntosDominantesDivideYVenceras(puntos, 0, puntos.size() -
1);
}
```

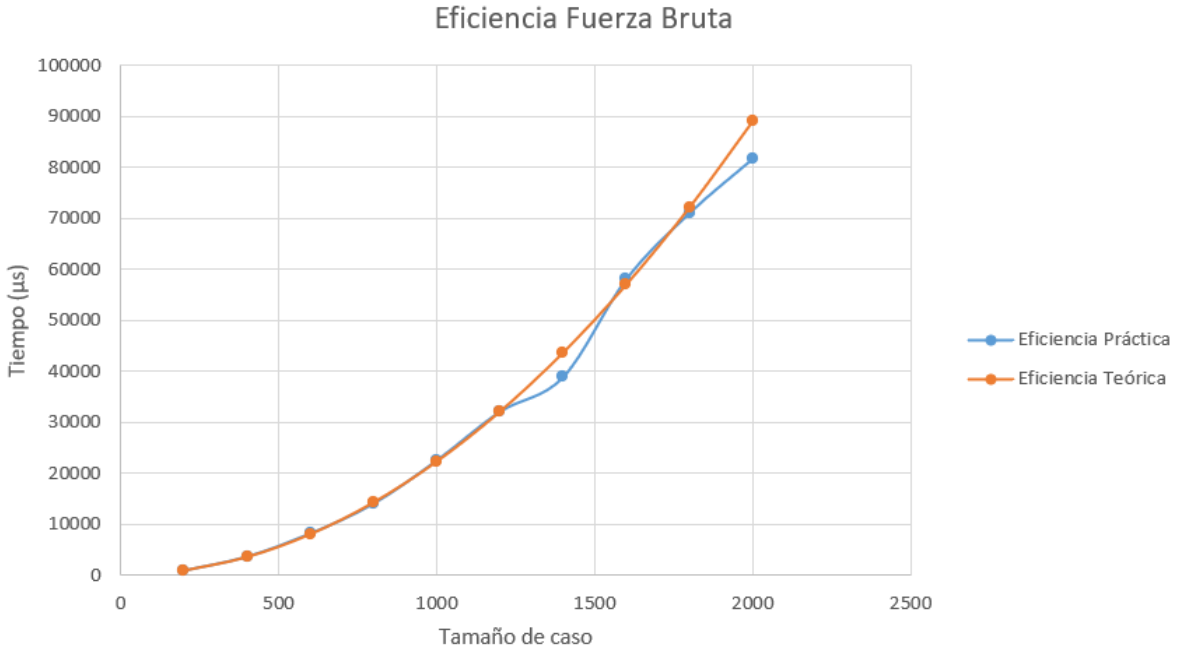
Salida de código:

```
/home/flo/Desktop/ALG/ALG_PRACTICA_2/cmake-build-debug/ALG_PRACTICA_2
Puntos dominantes:
(4, 5, 6, 7, 2, 2, 1, 8, 5, 3)
(5, 4, 3, 8, 3, 6, 4, 2, 9, 3)

Process finished with exit code 0
```

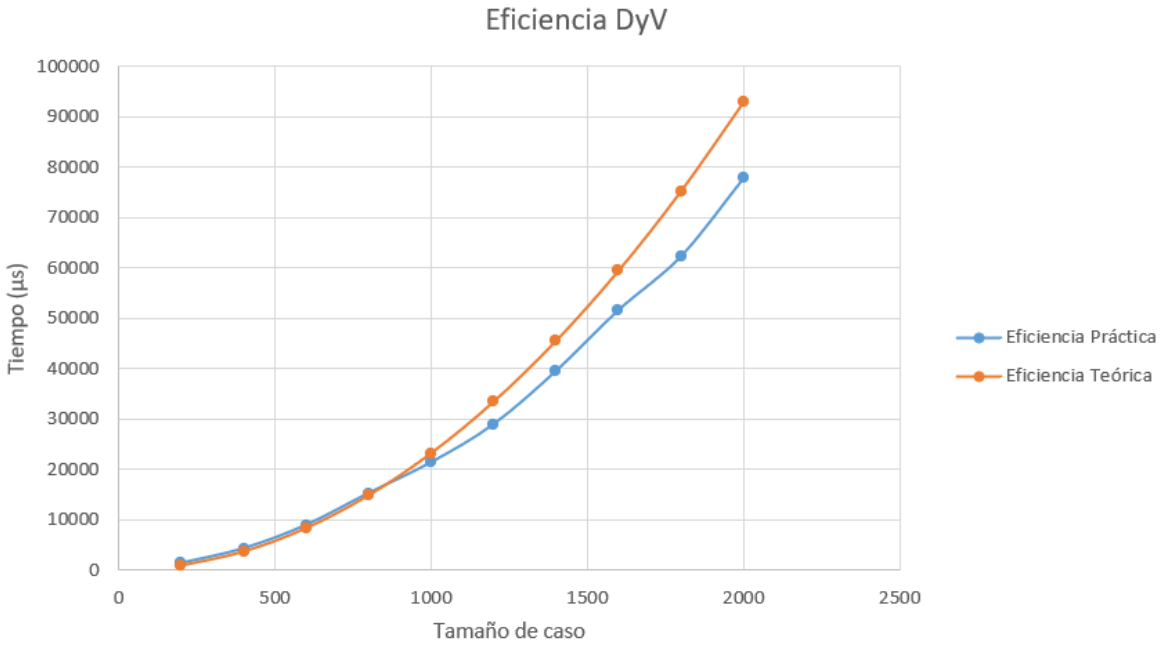
Comparación Eficiencias

Como podemos observar, la eficiencia práctica que nos ha salido tras ejecutar nuestro algoritmo numerosas veces, es similar a la eficiencia teórica estimada.



K =	10			K (media)
Tam. Caso (N)	**f(N) = N^2*K**	**T(n) (µs)**	**K=T(n)/f(n)**	**Tiempo teórico estimado = Kmedia*f(n)**
200	400000	979	0.0024475	890.3814016
400	1600000	3717	0.002323125	3561.525607
600	3600000	8253	0.0022925	8013.432615
800	6400000	14090	0.002201563	14246.10243
1000	10000000	22581	0.0022581	22259.53504
1200	14400000	32148	0.0022325	32053.73046
1400	19600000	38950	0.001987245	43628.68868
1600	25600000	58209	0.002273789	56984.4097
1800	32400000	71180	0.002196914	72120.89353
2000	40000000	81852	0.0020463	89038.14016
	K media	0.002225954		

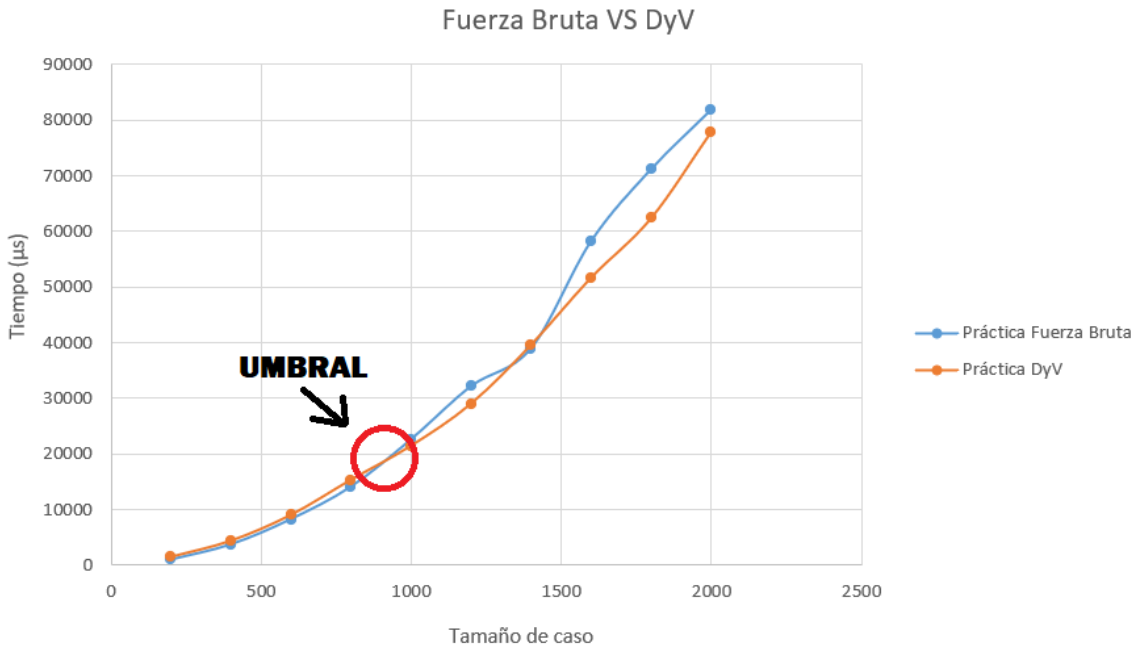
Como podemos observar, la eficiencia práctica que nos ha salido tras ejecutar nuestro algoritmo numerosas veces, varía en menor medida la similitud frente a la eficiencia teórica estimada.



K =	10			K (media)
Tam. Caso (N)	**f(N) = N^2*K**	**T(n) (µs)**	**K=T(n)/f(n)**	**Tiempo teórico estimado = Kmedia*f(n)**
200	400000	1426	0.003565	929.2991805
400	1600000	4340	0.0027125	3717.196722
600	3600000	8989	0.002496944	8363.692624
800	6400000	15313	0.002392656	14868.78689
1000	10000000	21427	0.0021427	23232.47951
1200	14400000	28996	0.002013611	33454.7705
1400	19600000	39583	0.002019541	45535.65984
1600	25600000	51666	0.002018203	59475.14755
1800	32400000	62357	0.001924599	75273.23362
2000	40000000	77869	0.001946725	92929.91805
	K media	0.00232325		

PROBLEMA DEL UMBRAL

Para tamaño de casos menores que 900, el algoritmo de Fuerza Bruta es más eficiente respecto al DyV, sin embargo, cuando sobrepasamos el umbral que se encuentra en torno a los 900 puntos, comienza a ser más eficiente el algoritmo de DyV.



Ejecución

Para la ejecución de los diferentes códigos preparados para la resolución de la práctica 2 de algorítmica, se debe de tener en cuenta que en cada .cpp se encuentra el código entero a probar, incluido el main correspondiente, para colocar dicho código en el main.cpp para ejecutarlo.

- Ejemplo ejecución "**Fuerza_Bruta_K10.cpp**":
  1º Se debe de copiar el código que se encuentra en dicho .cpp y posteriormente colocarlo en el main para ejecutarlo, dicho fichero contiene el main preparado para su ejecución.

  Dicha salida se debería de mostrar por pantalla:

```
/home/flo/Desktop/ALG/ALG_PRACTICA_2/cmake-build-debug/ALG_PRACTICA_2
Puntos dominantes:
(4, 5, 6, 7, 2, 2, 1, 8, 5, 3)
(5, 4, 3, 8, 3, 6, 4, 2, 9, 3)

Process finished with exit code 0
```

Conclusión

Como hemos podido comprobar con la realización de esta práctica es que tanto el algoritmo de Fuerza Bruta como el Divide y Vencerás tienen una misma eficiencia teórica O(N² *K), que a simple vista, diríamos que tardarían lo mismo.

Sin embargo, al haber calculado la eficiencia práctica para cada uno de ellos, hemos llegado a la conclusión de que para un número de puntos mayor (a partir del umbral), el algoritmo Divide y Vencerás tendrá unos tiempos de ejecución menores que el de Fuerza Bruta.