PRÁCTICA 2
ALGORÍTMICA

Integrantes:

Raúl Granados López
Hossam El Amraoui Leghzali
Javier Montaño Rubio

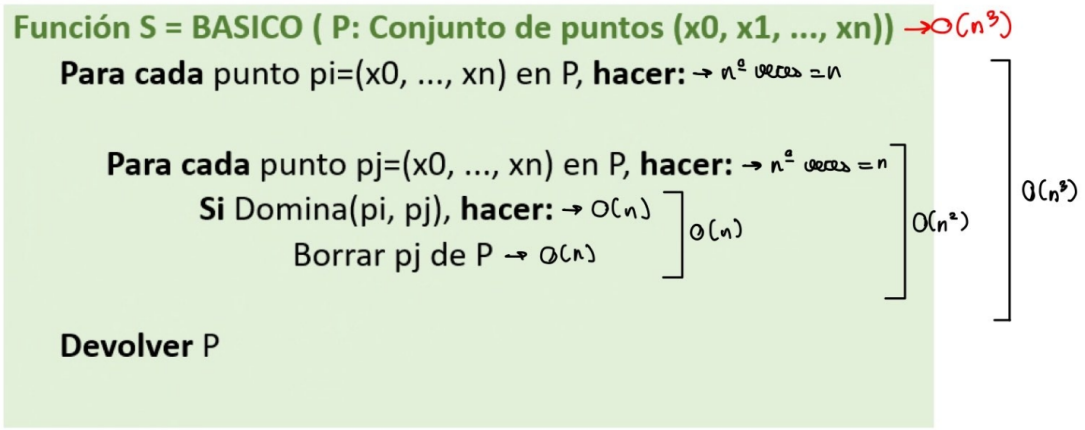
Cómo compilar y ejecutar los archivos:

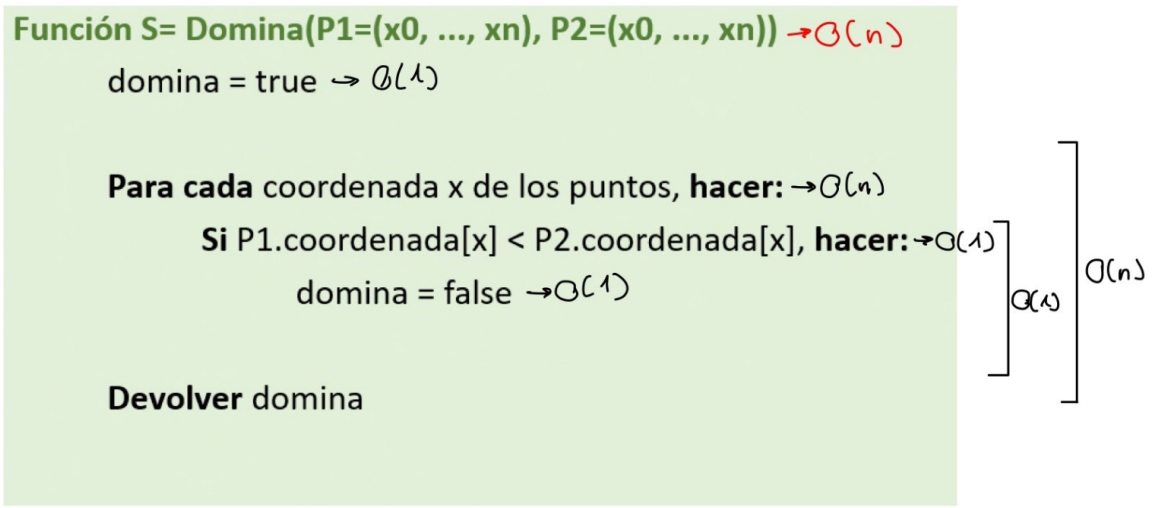
- **Para compilar:**
  g++ -o Practica2 Practica2.cpp
  También se incluye un makefile con el que se puede realizar la compilación. Para utilizarlo, ejecutar en la terminal *make*.

- **Para ejecutar:**
  ./Practica2 <FicheroConPuntos>
  Si se desean ver los resultados de la ejecución, descomentar la línea 138 de "Practica2.cpp".

1. ***Método básico***

El método básico consiste en que, dado un vector de puntos, se coge un punto y se va comprobando si domina a todos los demás uno a uno. En caso de que el punto cogido domine a otro punto, el punto dominado se borra del vector; en caso contrario, se sigue realizando la comprobación con los puntos siguientes.





2. ***Método divide y vencerás 1***

En el método divide y vencerás 1, dado un vector de puntos como argumento, comprueba el tamaño de éste para realizar su ejecución, en caso de ser de tamaño 1, simplemente devuelve el vector dado como argumento.

En cualquier otro caso, el vector se divide en 2 mitades del mismo tamaño (“n/2”) en caso de ser pares y en caso de ser impares en tamaño “n/2” y “n-n/2”, para posteriormente llamarse de forma recursiva a sí misma.

Después, una vez obtenidos los resultados de las llamadas recursivas, fusionamos los vectores en un solo vector, posteriormente, comprobamos si los puntos de un vector dominan a los del otro y viceversa.

Para comprobar si unos puntos dominan a otros entre ellos se comprueba para cada punto si es dominado por alguno de los demás. Si no es dominado por ninguno se añade a un vector “S” (solución).

Como este método nos ha resultado el más eficiente en cuanto a tiempos de ejecución, va a ser al que le vamos a realizar el estudio de la eficiencia teórica.

División del problema en subproblemas
El problema P de tamaño N consiste en obtener el subconjunto de puntos que sean puntos no dominados. Divido el problema en 2 subproblemas:

P_1: $\{p_1, p_2, \ldots, p_{N/2}\}$, de tamaño N/2.
P_2: $\{p_{N/2+1}, p_{N/2}, \ldots, p_N\}$, de tamaño N/2.

Combinación de soluciones
Sea S_1 la solución de P_1 y S_2 la solución de P_2. La solución del problema viene dada por el conjunto de puntos no dominados de del subconjunto $S_1 \cup S_2$.

Caso base
Se para de dividir cuando N<=1, donde la solución es el punto restante.

Algoritmo DyV1(puntos)

Función S= DyV1(P: Conjunto de puntos (x0, x1, ..., xn)) $\rightarrow O(n^3)$

Si |P| =1, **hacer:**
Devolver S= BASICO(P)
En otro caso:
Partir P en dos subconjuntos de igual tamaño P1 y P2

S1= DyV1(P1) $\rightarrow T(n/2)$
S2= DyV1(P2) $\rightarrow T(n/2)$
S= Fusionar1(S1, S2) $\rightarrow O(n^3)$
Devolver S

$\Big]\ T(n) = 2T(n/2) + n^3$

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

$n = 2^m$; $m = \log_2(n)$

$T(2^m) = T(2^{m-1}) + 2^{3m}$

PH: $T(2^m) - T(2^{m-1}) = 0$

$P_H(x) = x - 2$

PNH: $2^{3m} = b^m \cdot q(m)$ $\begin{cases} b = 8 \\ q(m) = 1 \\ d = 0 \end{cases}$

$P(x) = (x-2)\cdot(x-b)^{d+1} = (x-2)(x-8)$ $\begin{cases} r_1 = 2 & M_1 = 1 \\ r_2 = 8 & M_2 = 1 \end{cases}$

$t_m = c_{1a} \cdot 2^m \cdot m^0 + c_{2a} \cdot 8^m \cdot m^0$

$t_n = c_{1a} \cdot n + c_{2a} \cdot 2^{3\log_2(n)} = c_{1a} \cdot n + c_{2a} \cdot 2^{\log_2(n^3)} = c_{1a} \cdot n + c_{2a} \cdot n^3 \Rightarrow$

$\Rightarrow \boxed{O(n^3)}$

Función S= Fusionar1(S1, S2) $\rightarrow O(n^3)$

AUX = S1 + S2 (Unir vectores en vector auxiliar) $\rightarrow O(n)$
S=[]

Para cada punto pi(x0, ..., xn) en AUX, **hacer:** $\rightarrow O(n)$
dominado = false
Para cada punto pj=(x0, ..., xn) en AUX y !dominado, **hacer:** $\rightarrow O(n)$
Si Domina(pj, pi) y pi!=pj, **hacer:** $\rightarrow O(n)$
dominado = true

$\Big]\ O(n^2)$

$\Big]\ O(n^3)$

Si !dominado, **hacer:**
Añadir pi a S

Devolver S

3. ***Método divide y vencerás 2***

El método divide y vencerás 2, sigue la misma metodología que el primer divide y vencerás con la diferencia de la fusión entre distintos vectores, es decir, simplemente llama otra función para fusionar.

La forma de unir dos vectores es añadiéndolos todos en uno y llamar al método básico para eliminar los puntos de los vectores que se dominan entre sí.

Función S= DyV2(P: Conjunto de puntos (x0, x1, ..., xn))

Si |P| =1, **hacer:**
Devolver S= BASICO(P)
En otro caso:
Partir P en dos subconjuntos de igual tamaño P1 y P2

S1= DyV1(P1)
S2= DyV1(P2)
S= Fusionar2(S1, S2)
Devolver S

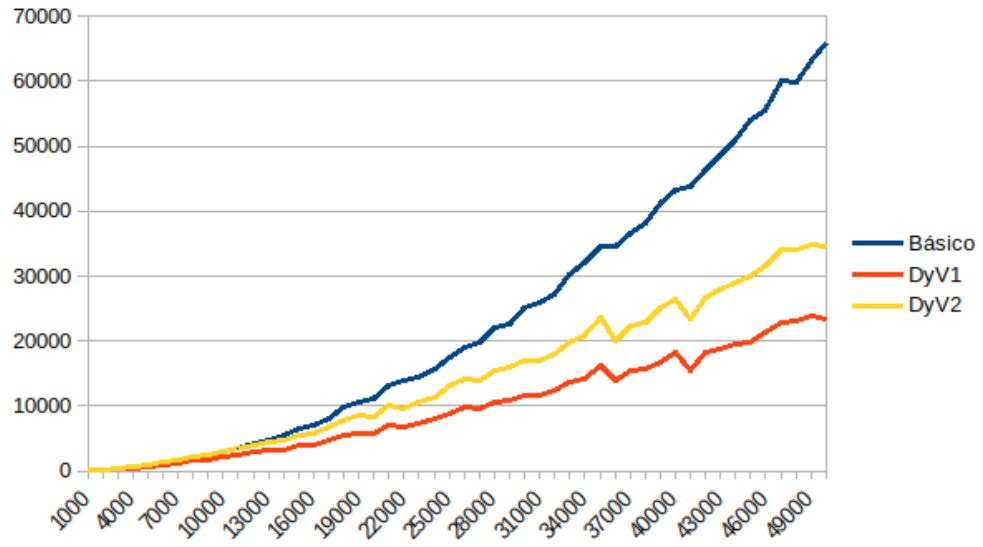
Función S= Fusionar2(S1, S2)

AUX = S1 + S2 (Unir vectores en vector auxiliar)

Devolver BASICO (AUX)

4. ***Estudio de eficiencia***

Para el estudio de eficiencia de esta práctica se ha tenido en cuenta como dimensión para todos los puntos **K=10**, y una cantidad de puntos entre **N=1000** y **N=50000**.



Como podemos observar la medición de los tiempos nos indica que el método básico tiene crece mucho más rápido que el resto de soluciones aunque estas sean de la misma eficiencia (n^3), esto se debe a la k que multiplica esta eficiencia teórica. También vemos que, aún con el mismo orden de eficiencia, el método DyV1 es mejor que el DyV2 en promedio, por eso la elección de este:

- Para el método básico: k = 6,68499989242889E-06
- Para el divide y vencerás 1: k = 4,02587079073035E-06
- Para el divide y vencerás 2: k = 5,68924000710006E-06

A continuación se muestran las capturas del código de cada uno de los métodos utilizados para realizar la práctica con sus respectivas eficiencias teóricas.

```
void Fusiona1 (vector<Punto> & puntosNoDominados, const vector<Punto> & p1, const vector<Punto> & p2) { // O(n^3)

    vector<Punto> aux = p1;
    for (int i=0; i < p2.size(); ++i) { // O(n)
        aux.push_back(p2[i]);
    }

    for (int i=0; i<aux.size(); i++) {  //O(n^3)
        bool dominado=false;

        for (int j=0; j<aux.size() && !dominado; j++) {  //O(n^2)
            if (aux[j].Domina(aux[i]) && i!=j) { //O(n)
                dominado = true;
            }
        }

        if (!dominado)
            puntosNoDominados.push_back(aux[i]);

    }
}
```

```cpp
vector<Punto> DyV1 (vector<Punto> p, int n) { // O(n^3)
    vector<Punto> puntosNoDominados;

    if (n == 1)
        puntosNoDominados.push_back(p[0]);
    else {

        vector<Punto>::const_iterator it = p.cbegin() + n/2;
        vector<Punto> p1(p.cbegin(), it);
        vector<Punto> p2(it, p.cend());

        vector<Punto> temp1 = DyV1(p1, n/2);         // T(n/2)
        vector<Punto> temp2 = DyV1(p2, n - n/2);     // T(n/2)

        Fusiona1 (puntosNoDominados, temp1, temp2); // O(n^3)

    }

    return puntosNoDominados;
}
```

```cpp
struct Punto{
    vector<int> coordenadas;

    // Devuelve si se domina a p2
    bool Domina (const Punto & p2) {

        bool domina = true;

        for (int i=0; i<coordenadas.size() && domina; i++) {
            if (coordenadas[i] < p2.coordenadas[i]) {
                domina = false;
            }
        }

        return domina;
    }
};
```