# PRÁCTICA 2: ALGORITMOS DIVIDE Y VENCERÁS

Algorítmica

2º Grado en Ingeniería Informática

UNIVERSIDAD DE GRANADA

JAVIER BUENAVENTURA MARTÍN JIMÉNEZ
EMILIO GUILLÉN ÁLVAREZ
ALBERTO RODRÍGUEZ FERNÁNDEZ

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.

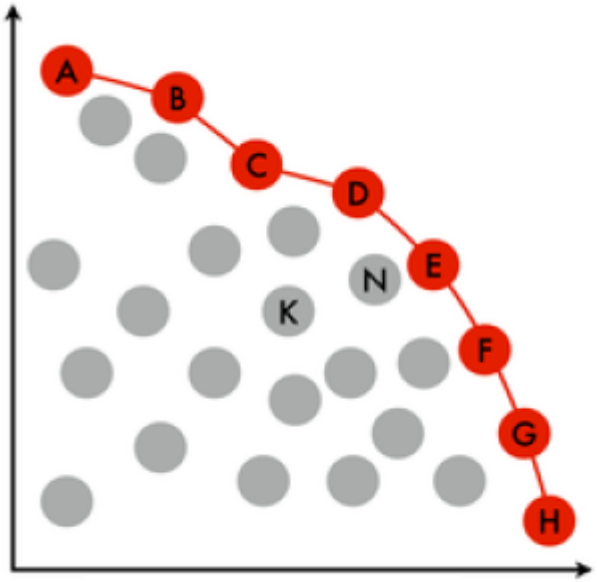
//Precondición: Se ofrece un conjunto de puntos en el hiperplano de dimensión K.

//Postcondición: Unos puntos dominan a otros debido a sus características.

**Begin algoritmo**

/*Debemos comparar los distintos jugadores unos con otros en función de estas distintas características, de forma que encontremos aquellos que en alguna o varias de ellas no sean superados por ningún otro. Esa comparación y elección se realiza con tres bucles de la siguiente manera:

n:*Número total de jugadores a comparar.*
*l: Jugador con el cual se está comparando.*
*i:Jugador que comparamos con el resto.*
*j: Características distintas de cada jugador.*
*k: Número total de características que posee un jugador. */



*Recorremos los distintos jugadores principales para añadir a un vector el conjunto de las soluciones buscadas*

**sol[i] = v[i];**
*Para ir añadiendo a un vector el conjuntos de las soluciones buscadas*

*Comparamos al jugador principal con cada uno de los otros jugadores.*

*Comparamos característica por característica*

*Si se cumple la condición de que sea menor que otra, incrementamos el contador*

**cont++;**
*Incrementamos mientras sea menor*

*Si coincide el índice con el contador*

**sol[i] = nullptr;**
*Dejamos en la solución solo aquellas que cumplen de verdad todos los requisitos*

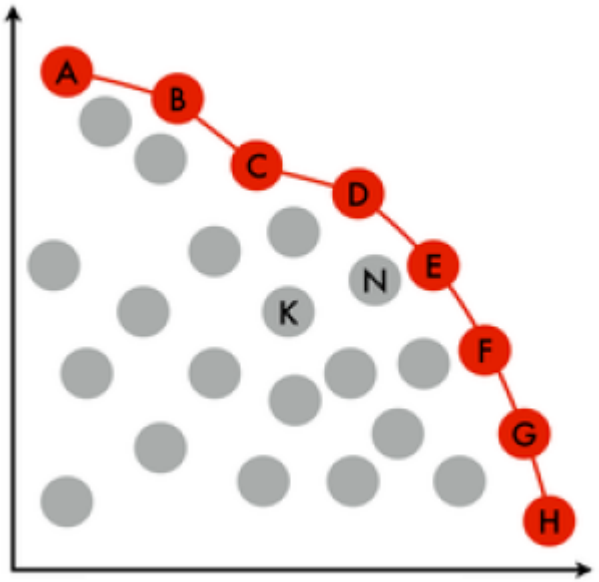
Analizamos ahora la eficiencia teórica de este algoritmo con el cual hemos hecho frente al problema planteado. Nos encontramos ante tres bucles for anidados, cada uno de ellos con una eficiencia individual O(n). Esto nos llevaría a deducir que la eficiencia total sería de O(n^3), sin embargo, llegamos a la conclusión tras un análisis más profundo de que realmente sería O(n^2*K). Viene a ser prácticamente lo mismo solo que esta pequeña diferencia es debida a que el último bucle no recorre n, sino que recorre k, por lo que no se podría emplear la otra notación incluyendo solamente n.

Su eficiencia teórica sería:

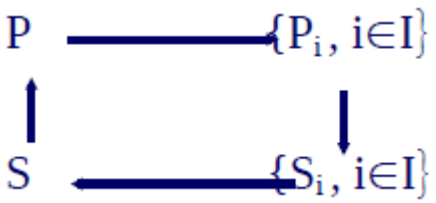
```
int l,cont;                                                    } O(1)
sol = new int *[n];                                            }

for (i = 0; i < n; i++) {                                      }
                                                               }
    sol[i] = v[i];                                             } O(1)
                                                               }
    for (l = 0,cont = 0;(l < n) && (cont != k); l++) {         }
                                                               }
        for (j = 0 ; (j < k) && (cont != k); j++){             }  O(n^2*k)
                                                               }
            if (v[i][j] < v[l][j]) {                           } O(1)
                cont++;                                        }
            }                                                  }
        }                                                      }
                                                               }
    }                                                          }
                                                               }
    if (cont == k) {                                           } O(1)
                                                               }
        sol[i] = nullptr;                                      }
    }
}
```

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.

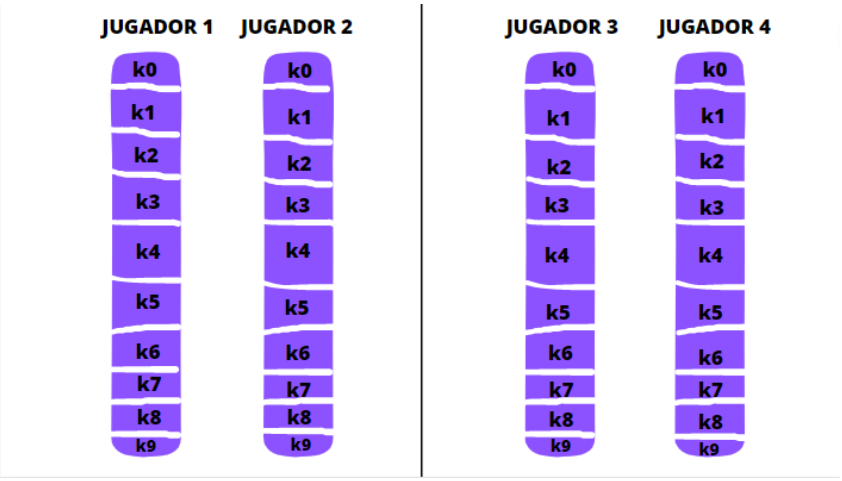



El problema abordado en el ejercicio 1 implementado mediante el algoritmo de fuerza bruta nos hace ver que el algoritmo empleado tiene una eficiencia O(n^2*k), por lo que nos hace plantearnos abordar el problema mediante la técnica Divide y Vencerás, que se basa en dividir el problema (P) en varios subproblemas (Pi), vencer cada uno de los subproblemas y combinar cada una de sus respectivas soluciones (Si) para obtener la solución (S) del problema inicial.

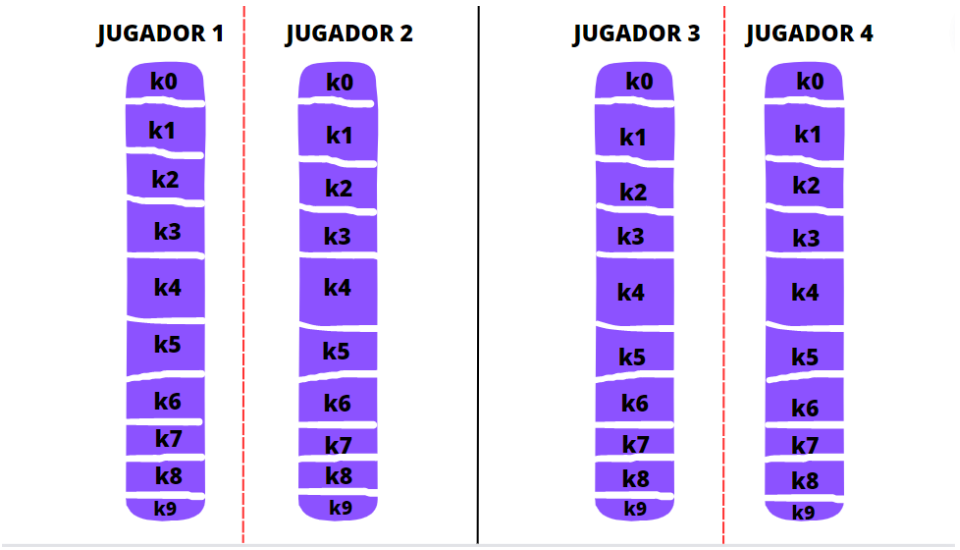


## ☐ ESTRATEGIA DE PARTICIÓN POR EL NÚMERO DE JUGADORES

En este caso, el subproblema sería dividir el número de jugadores.

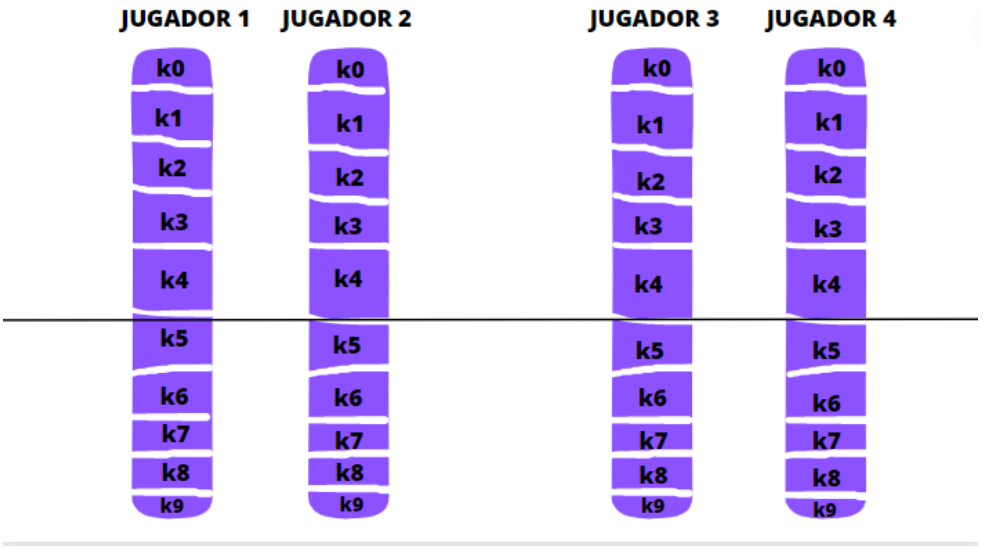



Con este tipo de caso, lo que se busca es dividir el número de jugadores en partes iguales, es decir, n/2, hasta que queden subconjuntos de únicos puntos no dominados. En el ejemplo superior, partimos de 4 jugadores (conjunto de n = 4 jugadores), si aplicamos Divide y Vencerás nos quedan 2 subconjuntos de 2 jugadores y si, sucesivamente volvemos a aplicar Divide y Vencerás, nos quedarán 4 subconjuntos de un único jugador, los cuales podremos interpretar como los 4 puntos no dominados del hiperplano K. Por lo tanto simplemente habrá que comparar las características entre estos 4 para determinar el que más convenga. Por lo tanto, lo que habrá que implementar será un único bucle (for) que compare las características entre los distintos mejores jugadores (aquellos puntos no dominados).



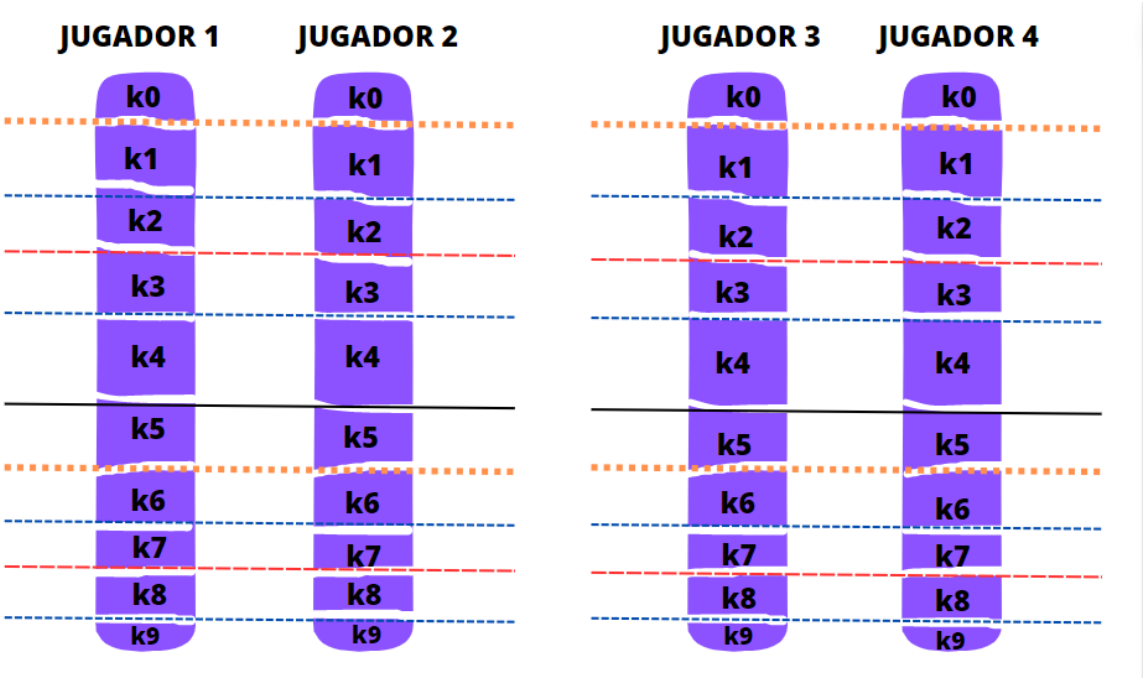
## ☐ ESTRATEGIA DE PARTICIÓN POR LAS CARACTERÍSTICAS DE CADA JUGADOR

En este caso, el subproblema sería dividir el número de características.




Con este tipo de caso, lo que se busca es dividir las características de los jugadores en partes iguales, es decir, K/2, hasta que queden subconjuntos de características únicas. En el ejemplo superior, partimos de 4 jugadores, si aplicamos Divide y Vencerás nos quedan 8 subconjuntos de 5 características y si, sucesivamente volvemos a aplicar Divide y Vencerás, nos quedarán 16 subconjuntos (8 serán de 3 características y los otros 8 serán 2 características). En este punto, tenemos 2 situaciones, una la cual en la que en la situación de las 2 características, aplicamos Divide y Vencerás y nos quedamos con 16 subconjuntos de 1 única característica, y la otra situación es la que tenemos 3 características, en la que entramos en 2 sub posibilidades: una en la que directamente analizamos las 3 características entre ellas mismas (esto sería menos eficiente debido a que habría que comparar K1 con K2, K2 con con K3 y K1 con K3) y la otra sería partir el conjunto de 3 en 2 partes: una de ellas con 2 características (volviendo aplicar Divide y Vencerás nos quedaríamos con 2 subconjuntos de 1 característica) y la otra parte con 1 única característica. (Observamos que esta segunda posibilidad es mucho más eficiente debido a la aplicación de Divide y Vencerás y la sencillez de hacerlo). Finalmente nos quedaríamos con 40 conjuntos de 1 única característica. Por lo

tanto, lo que habrá que implementar será dos bucles (for) que comparen las características entre todos y cada uno de los puntos (jugadores)..




EXPLICACIÓN DE IMAGEN PARA UN JUGADOR :

- LÍNEA NEGRA CENTRAL: dividir el conjunto en 2 partes iguales: (k0, k1, k2, k3, k4) y (k5, k6, k7, k8, k9).
- LÍNEA ROJA DISCONTINUA: subdividimos el conjunto superior en 2 partes: (k0, k1, k2), (k3, k4) || (k5, k6, k7), (k8, k9).
- LÍNEA NARANJA DISCONTINUA: subdividimos los conjuntos de 3 características superior en 2 partes: (k0), (k1,k2) || (k5), (k6,k7).
- LÍNEA AZUL DISCONTINUA: los subconjuntos restantes de 2 características los subdividimos en 2 partes: (k1), (k2), (k3), (k4), (k6), (k7), (k8), (k9).

Finalmente, agrupando todo nos quedaría:
k0, k1, k2, k3, k4, k5, k6, k7, k8, k9.

Podemos observar claramente que serás más eficiente, dividir por el número de jugadores, debido a que pasaríamos a una eficiencia teórica del total: O(n^2*k) a:

- Si partimos por el número de jugadores: O(k).
- Si partimos por las características: O(n^2).

El diseño completo del Algoritmo sería aproximadamente el siguiente:

```
Sol = D y V (P, N, K) {

    Comprobar si N es suficientemente pequeño para n = 2
    (Empleo del algoritmo de Fuerza Bruta)

        Sol = comparamitad (P, N, K)
        Return Sol

    En caso de N superiores a 2

        Sol1 = D y V (P, N/2 (mitad izquierda), K)
        Sol2 = D y V (P, N/2 (mitad derecha), K)

    Sol = Fusion (S1, S2)

    En caso de N = 1

        Añadimos como punto no dominado

    Return Sol
}
```

Donde Fusion será una función auxiliar que fusione las soluciones de los conjuntos S1 y S2.

La eficiencia teórica del Algoritmo Divide y Vencerás será:

```
int ** fusion(int ** s1, int ** s2, int tam1, int tam2, int k, int &tam){
    int cont,j,l ;
    tam = tam1 + tam2;
    int pos = 0;
    int ** s = new int * [tam];
    for (int f = 0; f < tam1 ; f++) {
        for (int m = 0 ; m < tam2 &&  s1[f]!= nullptr; m++) {
            for (j = 0, cont = 0; j < k && cont != k ; j++) {
                if (s1[f][j] < s2[m][j]) {
                    cont++;
                }
            }
            if (cont == k) {
                s1[f] = nullptr;
                tam--;
            }
        }
        if (s1[f] != nullptr) {
            s[pos]= s1[f];
            pos++;
        }
    }
    for (int f = 0; f < tam2 ; f++) {
        for (int m = 0 ; m < tam1 && s2[f]!= nullptr && s1[m]!= nullptr; m++) {
            for (j = 0, cont = 0; j < k  && cont != k; j++) {
                if (s2[f][j] < s1[m][j]) {
                    cont++;
                }
            }
            if (cont == k) {
                s2[f] = nullptr;
                tam--;
            }
        }
        if (s2[f] != nullptr) {
            s[pos]= s2[f];
            pos++;
        }
    }
    return s;
}
```

O(1)
O(1)
O(n^2*k)
O(1)
O(n^2*k)

```
int ** comparamitad( int **v, int i, int n, int k,int &tam){
    int cont,j,l;
    tam = 2 ;
    bool nodominado = false;
    for (l = i; l < n  ; l++) {
        for (j = 0, cont = 0; j < k && nodominado == false; j++) {
            if (v[l][j] < v[n][j]) {
                cont++;
            }else {
                nodominado = true;
            }
        }
        if (cont == k) {
            tam --;
            v[i] = nullptr;
        } else if (cont == 0) {
            tam --;
            v[n] = nullptr;
        }
    }
    int ** s = new int * [tam];
    int pos;
    for (l=i, pos= 0; (l <= n) && pos < tam  ; l++ ) {
        if (v[l] != nullptr){
            s[pos]= v [l];
            pos ++;
        }
    }
    return s;
}
```

O(1)
O(1)
O(n*k)
O(1)
O(n)

```
int ** divideyvenceras( int **v, int i, int n, int k, int &tam) {
    int ** s;
    if ((n-i) == 1) {
        s = comparamitad( v, i, n, k, tam);
    }else if (i<n){
        int ** s1, ** s2;
        int tam1, tam2;
        int q =  (n+i) / 2;
        s1 =divideyvenceras( v, i, q, k, tam1);
        s2 = divideyvenceras(v, q + 1, n, k, tam2);
        s =  fusion(s1,s2, tam1 , tam2, k, tam);
    }else if (n==i){
        s = new int * [1];
        s[0] = new int  [k];
        for (int f = 0; f < k; f++) {
            s[0][f] = v[i][f];
        }
        tam = 1;
    }
    return s;
}
```

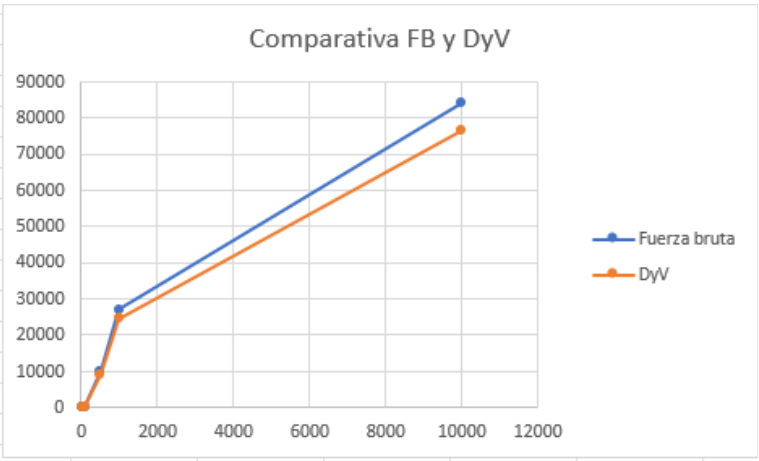
O(1)
O(n^2)
O(1)
O(log(n))
O(log(n))
O(n^2*k)
O(1)
O(1)
O(n)
O(n^2*k)

3. Implemente los algoritmos “básico” 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 asuma siempre un valor de K=10.

Tanto el algoritmo básico como el algoritmo Divide y Vencerás se han adjuntado como códigos fuentes a la práctica. La gráfica ha sido realizada con un algoritmo básico distinto al nuestro, ya que con nuestro algoritmo básico obteníamos una eficiencia muy difícil de superar.

## RESOLUCIÓN DEL PROBLEMA DEL UMBRAL DE FORMA EXPERIMENTAL

| | | |
|---|---|---|
| 10 | 6 | 5 |
| 100 | 290 | 237 |
| 500 | 9962 | 9041 |
| 1000 | 26985 | 24567 |
| 10000 | 83977 | 76339 |




Vemos al trazar la gráfica con la cual comparamos experimentalmente la eficiencia de ambos códigos como dependiendo de la cantidad de valores que añadamos a cada algoritmo la eficiencia es diferente.
En el caso de tener un bajo número de valores la se comprueba que dicha eficiencia es prácticamente la misma o incluso menos para el caso de Fuerza Bruta.
Si seguimos observando la gráfica conforme va a adoptando valores cada vez mayores se va incrementando la diferencia entre un algoritmo y otro, eficientemente hablando.
Cuando más altos son los valores mayor es la diferencia, reflejada en que DyV es más eficiente.
El umbral determinado de forma experimental es 10, puesto que a partir de esta cantidad de valores, el algoritmo DyV pasa a ser más eficiente que el algoritmo conocido como Fuerza Bruta.