Examen de Algorítmica y Complejidad (Plan 2014) — **11 de enero de 2023**

Nº matrícula:_____ **Nombre:** _____

Apellidos: _____

Problema 2 (5 puntos). El famoso streamer Ibai Llanos necesita ayuda para organizar su próximo Ibainéfico donde los streamers participantes, cada uno con un nivel de habilidad diferente (>0), se dividirán en equipos con un nivel total de habilidad lo más similar posible. Todos los streamers deben pertenecer a un equipo y todos los equipos deben tener al menos un streamer. Ibai les pide a los alumnos de Algorítmica que le echen una mano diseñando un algoritmo que, dada la lista con los niveles de habilidad de cada streamer, los dividan en *k* equipos con un grado de *desigualdad* lo menor posible. La desigualdad se mide como la diferencia entre la habilidad total del equipo más habilidoso menos la habilidad total del equipo menos habilidoso. Ejemplo: dado el siguiente vector de habilidades de los streamers para formar k = 4 equipos:

0	1	2	3	4	5	6	7	8	9
12	5	8	9	3	10	15	4	20	7

la formación del equipo 0 con los streamers de las posiciones {0,1,9} da un nivel total de 24, el equipo 1 con {2,6} da un nivel de 23, el equipo 2 con {3,5,7} da un nivel de 23 y el equipo 3 con {4,8} da un nivel de 23. Por lo tanto, el grado de desigualdad sería 24-23=1. Sin embargo, para k=11 equipos y el vector habilidades anterior, no hay solución pues algún equipo se quedaría vacío.

SE PIDE: Implementar un algoritmo en Java, basado en el ***esquema*** de ***Selección Óptima***[1], que ofrezca esta funcionalidad[2]. El algoritmo tendrá la siguiente cabecera:

```
int[] equipos(int[] habilidad, int k)
```

donde *habilidad* es un vector con el nivel de habilidad de cada streamer y *k* es el número de equipos a formar. El método devolverá un vector de valores *int*, con tantos elementos como tenga el vector *habilidad*, indicando el índice del equipo (empezando en 0) seleccionado para cada streamer. Si el algoritmo no encuentra solución devolverá *null*. Se podrán implementar todos los métodos que se consideren necesarios. No es necesario programar las clases Java Booleano y Entero.

```
int[] equipos (int[] habilidad, int k){
    if (habilidad.length >= k){
        int[] distribucion = new int[habilidad.length];
        int[] mejorDistribucion = new int[habilidad.length];
        int[] nivelEquipos = new int[k];
        Entero menorDesigualdad= new Entero(Integer.MAX_VALUE);
        for (int i=0; i<distribucion.length;i++){
            distribucion[i]=-1; mejorDistribucion[i]=-1;
        }
        for (int i=0; i<k; i++) nivelEquipos[i] = 0;
        equiposAux(habilidad, k, 0, 0, distribucion, nivelEquipos,
                   mejorDistribucion, menorDesigualdad);
        return mejorDistribucion;
    }
    else return null;
}
```

1 Desarrollar un algoritmo que no esté basado en la estrategia de Selección Óptima conllevará una puntuación de 0 en el ejercicio.
2 Desarrollar un algoritmo que genere un árbol de estados inválido para dar solución al problema conllevará una puntuación de 0 en el ejercicio.

```
void equiposAux(int[] habilidad, int k, int nivel, int proximo, int[] distribucion,
                int[] nivelEquipos, int[] mejorDistribucion, Entero menorDesigualdad){
    if (nivel==habilidad.length){
        boolean noEquiposVacios = true;
        int menorNivel=Integer.MAX_VALUE;
        int mayorNivel = 0;
        int i = 0;
        while (noEquiposVacios && i < k) {
            noEquiposVacios = nivelEquipos[i] != 0;
            menorNivel = Math.min(menorNivel, nivelEquipos[i]);
            mayorNivel = Math.max(mayorNivel, nivelEquipos[i]);
            i++;
        }
        if (noEquiposVacios && mayorNivel-menorNivel<menorDesigualdad.getValor()) {
            menorDesigualdad.setValor(mayorNivel-menorNivel);
            for (i = 0; i < distribucion.length; i++)
                mejorDistribucion[i] = distribucion[i];
        }
    } else{
        for (int c=0; c<=proximo; c++){
            // se intenta asignar habilidad[nivel] al equipo c
            distribucion[nivel] = c;
            nivelEquipos[c] = nivelEquipos[c] + habilidad[nivel];
            int ultimo = proximo; // para evitar estados simétricos
            proximo = Math.min(k-1,Math.max(c+1, proximo));
            nivel++;
            equiposAux(habilidad, k, nivel, proximo, distribucion, nivelEquipos,
                       mejorDistribucion, menorDesigualdad);
            nivel--;
            proximo = ultimo;
            distribucion[nivel] = -1;
            nivelEquipos[c] = nivelEquipos[c] - habilidad[nivel];
        }
    }
}
```