Control 8

Torneos

El objetivo de este control es profundizar en el diseño y el análisis de la eficiencia de algoritmos recursivos basados en el esquema "divide y vencerás".

El problema

El caballero Sir Closfit se encuentra ante una cola de enemigos. Debe batirse con todos ellos en sangriento torneo, según el orden que ocupan en la cola. En cada torneo puede seleccionar batirse con tantos enemigos como desee, pero siempre respetando el orden (es decir, si, en el siguiente torneo desea batirse con 10 enemigos, deberá ser con los 10 primeros en la cola). Así mismo, únicamente se le está permitido que realice un número máximo de torneos. Por último, debe elegir un conjunto de armas con poder suficiente como para poder vencer todos los torneos. Para ello, cada enemigo posee un poder de ataque, de tal forma que el poder de las armas elegidas por Sir Closfit debe superar la suma de los poderes de ataque de los contrincantes en cada torneo. Dado que el precio de las armas es directamente proporcional a su poder de ataque, Sir Closfit, que no se encuentra en su mejor momento económico, desea encontrar el mínimo poder de ataque con el que vencer exitosamente a sus contrincantes.

Debes diseñar un algoritmo "divide y vencerás" que ayude a Sir Closfit en su empeño. Las entradas a dicho algoritmo serán: (i) un vector de enteros indicando el poder de los enemigos (el valor de la posición 0, el poder del primer enemigo en la cola, el de la posición 1, el poder del segundo enemigo en la cola, etc.); y (ii) el máximo número de torneos que puede llevar a cabo. La salida será el poder mínimo de las armas que permitirán a Sir Closfit salir victorioso de la contienda.

Trabajo a realizar

Debe diseñarse el algoritmo DV pedido, completando los apartados indicados entre comentarios en el archivo `control8.cpp` que se proporciona como apoyo. Debe implementarse, además, el algoritmo. El punto de entrada al mismo será la función `min_poder`. Si se considera necesario, deberá definirse e implementarse una generalización adecuada, y definir el algoritmo pedido como una inmersión de dicha generalización. Aparte de llevar a cabo el diseño del algoritmo, e implementar el mismo, deberás determinar justificadamente su complejidad, definiendo y resolviendo las recurrencias necesarias (la resolución se llevará a cabo utilizando los patrones de recurrencias genéricas discutidos en clase).

Programa de prueba

Se proporcionan dos versiones alternativas (puede elegirse una u otra comentando o descomentando la definición `#define DOM_JUDGE` que se encuentra al comienzo del archivo):

- Si DOM_JUDGE está definido, el programa lee por la entrada estándar casos de prueba, los resuelve invocando a `min_poder`, e imprime los resultados. Cada caso de prueba consta de 2 líneas:
  - La primera contiene el número de enemigos, y el número máximo de torneos permitidos.
  - La segunda contiene, en orden, el poder de los enemigos que esperan para batirse con Sir Closfit.

  La lista de casos de prueba termina con un -1. A continuación se muestra un ejemplo de E/S:

Entrada	Salida
10 4 1 3 2 4 12 1 1 8 11 1 8 3 5 5 1 6 6 5 15 8 -1	12 23

- Si DOM_JUDGE no está definido, el programa genera aleatoriamente 1000 casos de prueba y compara el resultado del algoritmo con una implementación *naif* del mismo. Este modo de ejecución es útil para someter al algoritmo a una prueba de estrés. Si se genera un caso para el cuál la implementación DV y la implementación *naif* discrepan, el programa termina y muestra el valor para el que el algoritmo falla. Para activar este modo basta comentar la línea `#define DOM_JUDGE`

El archivo completo debe entregarse a través del juez en línea de la asignatura.

Importante:

- Únicamente se evaluarán aquellas entregas que superen satisfactoriamente los casos de prueba del juez.
- No modificar el código proporcionado.
- **No se corregirá ninguna entrega en la que no se hayan incluido los nombres de los miembros del grupo que han realizado el trabajo en el comentario que se incluye al comienzo del archivo.**