RapidLight

El problema

La empresa *RapidLight* se dedica al transporte de mercancías por carretera. La empresa dispone de vehículos con distintas capacidades de depósito, por lo que está interesada en disponer de un programa que, dado un *plan de transporte*, les permita determinar la capacidad mínima del vehículo necesaria para llevarlo a cabo. Más concretamente, un *plan de transporte* consta de:

- Una serie de etapas que deben completarse. Cada etapa requiere, para su realización, un determinado volumen de combustible: el depósito deberá contener, como mínimo, dicha cantidad de combustible, cantidad que se consumirá al realizar la etapa. Asimismo, al final de cada etapa será posible, si fuera necesario, volver a llenar el depósito del vehículo.
- Un número máximo de repostajes permitido, ya que cada repostaje supone invertir tiempo, y uno de los objetivos de *RapidLight* es transportar las mercancías en el menor tiempo posible.

Se pide desarrollar un algoritmo de *divide y vencerás* que determine la capacidad de depósito mínima del vehículo necesaria para ejecutar un plan de transporte, dados: (i) la lista de consumos requeridos por las etapas que componen el viaje; y (ii) el número máximo de repostajes permitidos en el camino. Aparte de desarrollar el algoritmo, deberás determinar razonadamente su complejidad.

Ten en cuenta que el camión comenzará su andadura con el depósito lleno, con cada repostaje llena de nuevo completamente el depósito, y le basta con llegar con la gasolina justa a su destino (es decir, con un volumen de gasolina mayor o igual que `0`).

Trabajo a realizar

Para realizar el control se proporciona un archivo `rapidlight.cpp` que contiene un programa que lee por la entrada estándar planes de transporte, y, para cada plan leído, invoca al algoritmo (llamando antes a la función `min_capacidad`), y escribe la capacidad mínima necesaria del vehículo para llevar a cabo dicho plan.
Cada plan de viaje consta de dos líneas: En la primera aparece el número de etapas del viaje (como mucho `100000`) y el número máximo de repostajes permitido. En la segunda aparecen, en orden, el consumo de combustible requerido por cada etapa. La entrada termina con una línea que contiene `-1`.
A continuación, se muestra un ejemplo de entrada / salida.

Entrada	Salida
2 1 6 4 3 0 3 6 5 3 1 3 6 5 -1	6 14 9

Importante:

Deben indicarse los nombres y apellidos de los autores del control en el comentario que aparece al principio del archivo `rapidlight.cpp` (**de lo contrario, el control no puntuará**). En caso de que alguno de los miembros del grupo no haya realizado el control, se deberá indicar en dicho comentario.

Únicamente puntuarán las soluciones que superen satisfactoriamente los casos de prueba del juez, en el tiempo máximo asignado para ello (veredicto de `CORRECT`).

Importante: aparte de la implementación, el algoritmo debe especificarse, diseñarse y analizarse correctamente, utilizando el método explicado en clase, utilizando los comentarios habilitados al respecto. Llevar a cabo una especificación, un diseño (generalización, casos, inmersión, terminación…) y un análisis de la complejidad correctos (recurrencias …) será esencial para superar esta prueba.

Únicamente se tendrá en cuenta el último envío `CORRECT` enviado dentro de plazo.

No modificar el código proporcionado. Únicamente deben responderse a los distintos apartados, en el interior de los comentarios, implementar la función `min_capacidad` y todas las funciones auxiliares que se consideren necesarias.