Gasolinera

```
-- Estrategia 1: búsqueda lineal sobre la cantidad de gasolina

    1. límite inferior: máxima gasolina necesaria para ir de una ciudad a su contigua
    2. aplicar búsqueda lineal empezando por el límite inferior. Es decir:
          2.0. cantidad actual de gasolina es el límite inferior
          2.1. si puedo hacer el recorrido ---> retornar cantidad actual
          2.2. si no puedo ---> incrementar en 1 la cantidad de gasolina y volver a 2.1

    Coste: O(n*k) en donde:
        n: número de ciudades
        k: mínima cantidad de gasolina para hacer todo el trayecto sin paradas


-- Estrategia 2: búsqueda dicotómica sobre la cantidad de gasolina

    1. límite inferior: máxima gasolina necesaria para ir de una ciudad a su contigua
    2. límite superior: mínima cantidad de gasolina para no hacer paradas
    3. aplicar búsqueda dicotómica con esos límites inicialmente. Es decir:
          3.0. mientras los dos límites sean diferentes:
          3.1.     calcular valor intermedio entre límite inferior y superior actual
          3.2.     si puedo hacer el recorrido ---> actualizar límite superior
          3.3.     si no puedo                 ---> actualizar límite inferior

    Coste: O(n log k)
```

Inversiones

```
-- Estrategia 1: comprobar todos con todos

    1. por cada posición i del vector
    2.      por cada posición j del vector tal que i < j
    3.            comprobar si esas posiciones son inversión y actualizar el contador

    Coste: O(n*n)

-- Estrategia 2: modificación del algoritmo de merge sort

    1. dividir en dos partes a, b el vector:

    2. solve:
          * llamada recursiva sobre a: retorna inversiones en a y a ordenado
          * llamada recursiva sobre b: retorna inversiones en b y b ordenado

    3. combinar:
          * a medida que se hace merge se puede contar las inversiones entre a y b
              con coste lineal

          * cuándo hay inversiones?:
              + si a[i] > b[j] entonces a[i], ..., a[size_a - 1] son inversiones con b[j]

          * por qué?:
              + Por el invariante del merge:
                             a[0], ..., a[i - 1] <= b[j]   (no son inversiones sobre b[j])
              + Como a está ordenado:
                             a[i] <= a[i + 1], ..., a[size_a - 1]
              + Si b[j] < a[i] entonces:
                             b[j] < a[i], ..., a[size_a - 1]  (son inversiones sobre b[j])

    Coste: O(n log n)
```