3. Queremos encontrar la suma de los elementos de un multiconjunto de números naturales. Cada suma se realiza exactamente entre dos números x e y y tiene costo $x + y$.

   Por ejemplo, si queremos encontrar la suma de $\{1, 2, 5\}$ tenemos 3 opciones:

   - $1 + 2$ (con costo 3) y luego $3 + 5$ (con costo 8), resultando en un costo total de 11;
   - $1 + 5$ (con costo 6) y luego $6 + 2$ (con costo 8), resultando en un costo total de 14;
   - $2 + 5$ (con costo 7) y luego $7 + 1$ (con costo 8), resultando en un costo total de 15.

Queremos encontrar la forma de sumar que tenga costo mínimo, por lo que en nuestro ejemplo la mejor forma sería la primera.

$a)$ Explicitar una estrategia golosa para resolver el problema.

S es nuestro multiconjunto

```
SumaGolosa(S):
  S' = min-heap(S)
  total = 0

  while(|S'| > 1):
    x = S'.extract()
    y = S'.extract()

    S'.insert(x+y)
    total+= x+y

  return total
```

Invariante: En el k-ésimo paso, nuestra solución G toma los elementos minimos $x, y \in S$, hacemos:

$S - \{x, y\} \wedge_L S \cup (x + y)$ y $G_k = x + y + G_{k-1}$

$b)$ Demostrar que la estrategia propuesta resuelve el problema.

Precondiciones:

$S = \{s_1, ..., s_n\}$ el multiconjunto de entrada, $s_i \in \mathbb{N}$

$S_k = S_{k-1} - \{x, y\} \cup \{(x + y)\}$, con x, y los minimos $\in S_{k-1}$ (o sea, los elementos que podemos usar en la k-ésima iteración).

G es nuestra solución greedy

El costo greedy C hasta el paso k: $C_k = x + y + C_{k-1}$, con $x, y \in S_k$, con $C_0 = C_1 = 0$

O una solución óptima cualquiera

Usaremos inducción fuerte sobre k:

Caso base $k = 2$:
Luego $S_2 = \{x, y\} \Rightarrow C_2 = x + y + C_1 = x + y$ hay una sola suma válida y trivialmente es igual a la óptima O.

Paso recursivo $k > 2$:
H.I.: $\forall j < k :: C_j$ es el costo mínimo hasta el j-ésimo paso.

G_k toma los mínimos $x, y \in S_{k-1}$ tal que $C_k = x + y + C_{k-1}$

Caso 1: Si O_k toma x, y y por HI C_{k-1} es óptimo, luego C_k es igual de óptimo que O_k

Caso 2: Si O_k toma $v, w \in S_k$ con $v + w \neq x + y$, dado que x, y son mínimos en S_k, entonces sabemos que $x + y \leq v + w$, luego:

Si en lugar de tomar $v + w$ tomase $x + y$, entonces el costo C'_k de O_k sería

$$C'_k = x + y + C'_{k-1} \leq v + w + C'_{k-1}$$

Por lo que el costo de C'_k es óptimo y $v \vee w$ quedan disponibles para elegirse posteriormente.

---

c) Implementar esta estrategia en un algoritmo iterativo. **Nota:** el mejor algoritmo simple que conocemos tiene complejidad $\mathcal{O}(n \log n)$ y utiliza una estructura de datos que implementa una secuencia ordenada.

```
public static int sg(List<Integer> S) {
    PriorityQueue<Integer> heap = new PriorityQueue<>(S);
    int x,y;
    int total = 0;

    while(heap.size() > 1) {
        x = heap.poll();
        y = heap.poll();

        heap.offer(x+y);
        total += x+y;
    }

    return total;
}
```