- Describa cuál es el criterio de selección.
- ¿En qué estructuras de datos representará la información del problema?
- Explique el algoritmo, es decir, los pasos a seguir para obtener el resultado. No se pide que "lea" el algoritmo ("se define una variable x", "se declara un for"), si no que lo explique ("se recorre la lista/el arreglo/" o "se elije de tal conjunto el que satisface...").
- Escriba el algoritmo en el lenguaje de programación de la materia.

7. Un submarino averiado descansa en el fondo del océano con n sobrevivientes en su interior. Se conocen las cantidades $c_1, \ldots, c_n$ de oxígeno que cada uno de ellos consume por minuto. El rescate de sobrevivientes se puede realizar de a uno por vez, y cada operación de rescate lleva t minutos.

   (a) Escribir un algoritmo que determine el orden en que deben rescatarse los sobrevivientes para salvar al mayor número posible de ellos antes de que se agote el total C de oxígeno.

- **Criterio de selección**
  Elige al sobreviviente que más oxígeno consume por minuto.

- **Estructuras de datos**
  Planteo a los sobrevivientes como una tupla de dos elementos, conformada por un identificador con el nombre de la persona y la cantidad de oxígeno que consume por minuto.

```
type Sobrevivientes = tuple
                        id: string
                        oxigeno: nat
                      end tuple

fun rescate(S: Set of Sobrevivientes, t: nat, C: nat) ret res: List of Sobrevivientes
```

Los navegantes del submarino en espera de ser rescatados se organizan en un conjunto. Aparte de ello se proporciona el tiempo de cada operación de rescate y el oxígeno total que hay dentro del submarino. Luego los sobrevivientes que pudieron ser salvados son derivados a una lista.

- **Descripción de cómo se soluciona el problema**
  El algoritmo elige al navegante que más oxígeno consume por minuto, priorizando así un menor consumo de oxígeno en la cabina para favorecer a la mayoría de los sobrevivientes. A medida que una persona es rescatada, se la descarta y se vuelve a repetir el algoritmo hasta que el oxígeno del interior del submarino se agote completamente.

- **Definición del algoritmo**

```
type Sobrevivientes = tuple
                        id: string
                        oxigeno: nat
                      end tuple

fun rescate(S: Set of Sobrevivientes, t: nat, C: nat) ret res: List of Sobrevivientes
    var S_aux: Set of Sobrevivientes
    var s: Sobrevivientes
    var oxigeno_restante: int

    S_aux := copy_set(S)
    res := empty_list()
    oxigeno_restante := C

    while not is_empty_set(S_aux) v oxigeno_restante > 0 do
        s := elegir_sobreviviente(S_aux)
        addr(res,s)
```

```
        elim(S_aux,s)
        oxigeno_restante := actualizar_oxigeno(S_aux,oxigeno_restante,t)
    od

    destroy_set(S_aux)
end fun

fun elegir_sobreviviente(S: Set of Sobrevivientes) ret res: Sobrevivientes
    var S_aux: Set of Sobrevivientes
    var s: Sobrevivientes
    var mayor_consumo_oxigeno: nat

    S_aux := copy_set(S)
    mayor_consumo_oxigeno := -∞

    while not is_empty_set(s) do
        s := get(S_aux)
        if mayor_consumo_oxigeno < s.oxigeno then
            mayor_consumo_oxigeno := s.oxigeno
            res := s
        fi
        elim(S_aux,s)
    od

    destroy_set(S_aux)
end fun

fun actualizar_oxigeno(S: Set of Sobrevivientes, o: int, t: nat) ret res: int
    var S_aux: Set of Sobrevivientes
    var s: Sobrevivientes

    S_aux := copy_set(S)
    res := o

    while not is_empty_set(S_aux) do
        s := get(S_aux)
        res := res - (t * s.oxigeno)
        elim(S_aux,s)
    od

    set_destroy(S_aux)
end fun
```

(b) Modificar la solución anterior suponiendo que por cada operación de rescate se puede llevar a la superficie a m sobrevivientes (con $m \leq n$).

```
type Sobrevivientes = tuple
                        id: string
                        oxigeno: nat
                      end tuple

fun rescate(S: Set of Sobrevivientes, t: nat, m: nat, C: int) ret res: List of
Sobrevivientes
    var S_aux: Set of Sobrevivientes
    var s: Sobrevivientes
    var oxigeno_restante: int
```

```
    S_aux := copy_set(S)
    res := empty_list()
    oxigeno_restante := C

    while not is_empty_set(S_aux) v oxigeno_restante > 0 do
        for i := 1 to m do
            if not is_empty_set(S_aux) then
                s := elegir_sobreviviente(S_aux)
                addr(res,s)
                elim(S_aux,s)
            fi
        od

        oxigeno_restante := actualizar_oxigeno(S_aux,oxigeno_restante,t)
    od

    destroy_set(S_aux)
end fun

fun elegir_sobreviviente(S: Set of Sobrevivientes) ret res: Sobrevivientes
    var S_aux: Set of Sobrevivientes
    var s: Sobrevivientes
    var mayor_consumo_oxigeno: nat

    S_aux := copy_set(S)
    mayor_consumo_oxigeno := -∞

    while not is_empty_set(s) do
        s := get(S_aux)
        if mayor_consumo_oxigeno < s.oxigeno then
            mayor_consumo_oxigeno := s.oxigeno
            res := s
        fi
        elim(S_aux,s)
    od

    destroy_set(S_aux)
end fun

fun actualizar_oxigeno(S: Set of Sobrevivientes, o: int, t: nat) ret res: int
    var S_aux: Set of Sobrevivientes
    var s: Sobrevivientes

    S_aux := copy_set(S)
    res := o

    while not is_empty_set(S_aux) do
        s := get(S_aux)
        res := res - (t * s.oxigeno)
        elim(S_aux,s)
    od

    set_destroy(S_aux)
end fun
```