- 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.

6. Para obtener las mejores facturas y medialunas, es fundamental abrir el horno el menor número de veces posible. Por supuesto que no siempre es fácil ya que no hay que sacar nada del horno demasiado temprano, porque queda cruda la masa, ni demasiado tarde, porque se quema.

   En el horno se encuentran n piezas de panadería (facturas, medialunas, etc). Cada pieza i que se encuentra en el horno tiene un tiempo mínimo necesario de cocción t_i y un tiempo máximo admisible de cocción T_i. Si se la extrae del horno antes de t_i quedará cruda y si se la extrae después de T_i se quemará.

   Asumiendo que abrir el horno y extraer piezas de él no insume tiempo, y que $t_i \leq T_i$ para todo $i \in \{1, \ldots, n\}$, ¿qué criterio utilizaría un algoritmo voraz para extraer todas las piezas del horno en perfecto estado (ni crudas ni quemadas), abriendo el horno el menor número de veces posible? Implementarlo.

   - **Criterio de selección**
     Extrae la pieza en buen estado que más pronta esté a quemarse junto todas las demás que ya se hayan cocinado, es decir que se encuentren entre el tiempo mínimo t_i y el tiempo máximo T_i.

   - **Estructuras de datos**
     Planteo a todas las piezas como una tupla de tres elementos, indicando el tiempo mínimo de cocción para evitar que no esté cruda, el tiempo máximo de cocción para evitar que se queme y un identificador para ubicar tal pieza.

```
type Piezas = tuple
                id: nat
                tmin_coccion: nat
                tmax_coccion: nat
              end tuple

{- PRE: t_i ≤ T_i -}
fun abrir_horno(P: Set of Piezas) ret res: nat
```

     Presento a las piezas de panadería que están dentro del horno como un conjunto, y el algoritmo devolverá la menor cantidad de veces en las que se debe abrir el horno.

   - **Descripción de cómo se soluciona el problema**
     El algoritmo elige la pieza más pronta a quemarse para sacar del horno y abre el horno antes de que eso suceda. Además retira el resto de piezas que estén perfectamente cocinadas y se las retira del conjunto, cerrando nuevamente el horno. Repito eso sucesivamente hasta que no queden más piezas dentro del horno.

   - **Definición del algoritmo**

```
type Piezas = tuple
                id: nat
                tmin_coccion: nat
                tmax_coccion: nat
              end tuple

{- PRE: t_i ≤ T_i -}
fun abrir_horno(P: Set of Piezas) ret res: nat
    var P_aux: Set of Piezas
    var p: Piezas
```

```
    var tiempo: nat

    P_aux := copy_set(P)
    res := 0
    tiempo := 0

    while not is_empty_set(P_aux) do
        p := elegir_pieza(P_aux,tiempo)
        elim(P_aux,p)
        tiempo := tiempo + p.tmax_coccion
        descartar_piezas_cocinadas(P_aux,tiempo)
        res := res + 1
    od

    destroy_set(P_aux)
end fun

fun elegir_pieza(P: Set of Piezas, t: nat) ret res: Piezas
    var P_aux: Set of Piezas
    var p: Piezas
    var casi_quemada: nat

    P_aux := copy_set(P)
    casi_quemada := ∞

    while not is_empty_set(P_aux) do
        p := get(P_aux)
        if p.tmax_coccion < casi_quemada then
            casi_quemada := p.tmax_coccion
            res := p
        fi
    od

    destroy_set(P_aux)
end fun

proc descartar_piezas_cocinadas(in/out P: Set of Piezas, in t: nat)
    var P_aux: Set of Piezas
    var p: Piezas

    P_aux := copy_set(P)

    while not is_empty_set(P_aux) do
        p := get(P_aux)
        if p.tmin_coccion ≤ t ∧ p.tmax_coccion ≥ t then
            elim(P,p)
        fi
        elim(P_aux,p)
    od

    destroy_set(P_aux)
end proc
```