```
type Edificio = (Int,Int,Int)

type Coordenada = (Int,Int)

type Skyline = [Coordenada]
```

`resuelveSkyline`: se trata de la función principal que implementa el algoritmo del Skyline mediante un esquema Divide y Vencerás. Debe recibir un único parámetro consistente en una lista de edificios y deberá devolver un valor de tipo `Skyline` conteniendo la línea de horizonte que dibuja la lista de edificios de entrada.

```
-- Función principal que resuelve un Skyline a partir de una lista de edificios

-- Implementa el algoritmo mediante un Divide y Vencerás

resuelveSkyline :: [Edificio] -> Skyline
```

- De la lista vacia, nos devuelve la lista vacia
- `*Skyline> resuelveSkyline[]`
- `[]`
  Cuando la lista tiene más de dos elementos, la tiene que dividir la lista en dos, volver a aplicar resulveSkyline a cada una de las partes. La función combina tiene que aplicarse a dos elementos resuelveSkyline “primera parte”, resuelveSkyline “segunda parte”
- De la lista con un solo elemento nos aplica la función edificioAskyline

```
*Skyline> resuelveSkyline[(1,4,5),(2,4,7),(3,7,22)]
[(1,5),(2,7),(3,22),(7,0)]
*Skyline> divide [(1,4,5),(2,4,7),(3,7,22)]
([(3,7,22),(1,4,5)],[(2,4,7)])
*Skyline> combina (combina (resuelveSkyline[(3,7,22)]) (resuelveSkyline
[(1,4,5)])) (resuelveSkyline[(2,4,7)])
[(1,5),(2,7),(3,22),(7,0)]
```

`edificioAskyline`: es la función que será llamada por `resuelveSkyline` cuando el esquema Divide y Vencerás no realiza nuevas subdivisiones del problema. Es decir, cuando se encuentra con el caso trivial. Recibe un único edificio y devuelve su línea de horizonte.

```
-- Función que transforma un Edificio en su Skyline correspondiente
-- Se utiliza en el caso base del algoritmo
edificioAskyline :: Edificio -> Skyline


*Skyline> edificioAskyline(2,7,30)
[(2,30),(7,0)]
*Skyline> :t edificioAskyline
edificioAskyline :: Edificio -> Skyline
```

`divide`: es la función que será llamada por `resuelveSkyline` para realizar la división de la lista de edificios en dos mitades de igual tamaño (o, si la longitud es impar, una lista contendrá un edificio más que la otra). Recibe una lista de edificios y devuelve una tupla con las dos listas de edificios en las que se va a dividir el problema.

-- Función que dado un problema (en forma de lista de Edificios) lo divide en
-- dos subproblemas cuyo tamaño es la mitad del problema original

divide :: [a] -> ([a],[a])

*Skyline> divide [(1,4,5),(2,4,7),(3,7,22),(8,22,55)]
([(3,7,22),(1,4,5)],[(8,22,55),(2,4,7)])
*Skyline> divide [(1,4,5),(2,4,7),(3,7,22)]
([(3,7,22),(1,4,5)],[(2,4,7)])
*Skyline>
*Skyline> divide [(1,4,5)]
([(1,4,5)],[])
*Skyline> divide []
([],[])
*Skyline> divide [(1,4,5),(2,4,7)]
([(1,4,5)],[(2,4,7)])
*Skyline> :t divide
divide :: [a] -> ([a], [a])
*Skyline>

`combina`: es la función que será llamada por `resuelveSkyline` para combinar las soluciones parciales de los dos subproblemas. Recibirá dos líneas de horizonte y las combinará en una única línea utilizando el proceso descrito en la sección 2.1.1.

-- Función de combinación que dados dos Skylines los combina en uno único
-- Realiza dos tareas al mismo tiempo:

```
*Skyline> resuelveSkyline[(1,4,5),(2,4,7),(3,7,22)]
[(1,5),(2,7),(3,22),(7,0)]
```

```
*Skyline> putStrLn(dibujaSkyline
(resuelveSkyline[(1,4,5),(2,4,7),(3,7,22)]))
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
   ****
  *****
  *****
 ******
 ******
 ******
 ******
 ******
--------
```

1- Eficiencia ejecución por ejemplo Java es más potente, ya que tiene asignación de memoria a variables y no es necesario recorrer una lista entera para llegar a un elemento. Array-Vector indexado.
2- Eficiencia respecto a la programación, Haskell, Prolog, Java.
3- Corte en Prolog ! y Java.

http://www.swi-prolog.org/pack/list?p=list_util

See `list_util.pl` documentation below for details about each exported predicate.

```
?- pack_install(list_util).
```

use_module(library(list_util)).

even(X) :- 0 is X mod 2.

edificioASkyline([e(X,Y,Z)],D):- D=[c(X,Z),c(Y,0)].

?- edificoASkyline([e(1,7,20)],D).

Correct to: "edificioASkyline([e(1,7,20)],D)"? yes

D = [c(1, 20), c(7, 0)].

1 ?- use_module(library(list_util)).

true.

2 ?- L0 =[1,2,3,4,5,6,7,8],length(L0,X), Y is div(X,3),split_at(Y,L0,Take,Rest).

L0 = [1, 2, 3, 4, 5, 6, 7, 8],

X = 8,

Y = 2,

Take = [1, 2],

Rest = [3, 4, 5, 6, 7, 8].

6 ?- L0 =[1,2,3,4,5,6,7,8,9],length(L0,X), Y is div(X,2),split_at(Y,L0,Take,Rest),length(Take,X1),length(Rest,X2),Y2 is div(X1,2),Y3 is div(X2,2),split_at(Y2,Take,Take1,Rest1),split_at(Y3,Rest,Take2,Rest2).

L0 = [1, 2, 3, 4, 5, 6, 7, 8, 9],

X = 9,

Y = X1, X1 = 4,

Take = [1, 2, 3, 4],

Rest = [5, 6, 7, 8, 9],

X2 = 5,

Y2 = Y3, Y3 = 2,

Take1 = [1, 2],

Rest1 = [3, 4],

Take2 = [5, 6],

Rest2 = [7, 8, 9].

8 ?- L0 =[1,2,3,4],length(L0,X), Y is div(X,2),split_at(Y,L0,Take,Rest),length(Take,X1),length(Rest,X2),Y2 is div(X1,2),Y3 is div(X2,2),split_at(Y2,Take,Take1,Rest1),split_at(Y3,Rest,Take2,Rest2).

L0 = [1, 2, 3, 4],

X = 4,

Y = X1, X1 = X2, X2 = 2,

Take = [1, 2],

Rest = [3, 4],

Y2 = Y3, Y3 = 1,

Take1 = [1],

Rest1 = [2],

Take2 = [3],

Rest2 = [4].