- *prepararArregloEstanImagenes(int im)*

  inicializa un arreglo en false
  Costo $O(im)$

- *intervalosElementales(Lista(imagenes) img) − > Conj(puntos):res*

```
intervalosElementales(Lista(imagenes) img) -> Conj(puntos):res
{
    desde 0 a Tam(img) hacer    // O(1) / itera im veces
    {
        res <- agregar(x_0)     // O(log(2im))
        res <- agregar(x_1)     // O(log(2im))
    }
    return res                  // O(1)
}
```

  Como ingresamos dos puntos por imagen a lo sumo el conjunto tendra $2im$, por lo que pudimos acotar la insercion $log(2im)$.
  El costo total de este algoritmo es pues $O(im * log(im))$

- *insertar(IntervaloElemental ie)*

- *llenarIntervalosElementales(Lista(Imagen):imagenes, bool:X)*

```
llenarIntervalosElementales(Lista(Imagen):imagenes, bool:X)
{
    Conj(IntervaloElemental):puntos = intervalosElementales(imagenes, X);
                                        // O(im*log(im))

    por cada imagen de "imagenes"       // itera im veces
        insertar(elemento)              // O(log(n))
}
```

  Entonces la complejidad de insertar todos los intervalos elementales es
  $O(im * log(im)) + O(im * log(n))$

- *insertarImagen(int indiceImagen, inicio, final, min, max, Nodo actual)*

- *insertarImagenes(Lista(Imagen):imagenes, bool:X)*

```
insertarImagenes(Lista(Imagen):imagenes, bool:X)
{
    depeniendo de si es por X o por Y
        por cada imagnen de "imagenes"
            insertarImagen(i, inicio, fin, 0, ANCHO_ARBOL, raiz);
}
```

- *llenarArbol(Lista(Imagen):imagenes, bool:X) − > ArbolDeIntervalos*

```
llenarArbol( Lista(Imagen):imagenes, bool:X) -> ArbolDeIntervalos
{
    prepararArregloEstanImagenes(im);
    llenarIntervalosElementales(imagenes, X);
    insertarImagenes(imagenes, X);
}
```

- *buscarIndices(int punto, Nodo actual, conj(int))*

```
buscarIndices(int punto, Nodo actual, conj(int))
{
    si no llegue al final del arbol
        res <- agregar(los indices que aparecen en el nodo)
        {
        si  punto es <= actual_numero
            buscarIndices(punto, Izq(actual), conj(int))
        si  punto es >= actual_numero
            buscarIndices(punto, Der(actual), conj(int))
        }
}
```

Este es un algoritmo recursivo, que separa siempre el problema es dos partes iguales.
¿Entra siempre por las dos ramas?... No
Existe a lo sumo solo un nodo en el arbol que tiene el mismo valor que el punto buscado, por lo tanto el algoritmo, hace a lo sumo dos caminos.
Que recorra uno o dos caminos es despreciable, por lo que su complejidad se puede expresar de la siguinte manera.

$$\begin{cases} T(0) = d \\ T(n) = T(n/2) + c + k_i \end{cases}$$

desarrollando una vez

$$T(n) = T(n/4) + c + k_i + c + k_j$$

la formula general es

$$T(n) = T(n/2^h) + h * c + \sum_{i=1}^{h} k_i$$

con h es la altura del arbol
como es un *Árbol* ***Rojo*** *y* ***Negro*** la altura es cercana a $log(n)$

$$T(n) = T(n/2^{log(n)}) + log(n) * c + \sum_{i=1}^{h} k_i$$

como a cada imagen la incuentro una sola vez por camino (pues si la encontre en un nodo no puedo escontrarla en sus hijos), habré encontrado al final mis k imagenes buscadas.

$$T(n) = T(n/n) + log(n) * c + k$$

$$T(n) = T(1) + log(n) * c + k$$

Esto da un complejidad total de $O(log(n) + k)$.

- *busqueda(x, y, Lista(Imagen):Imagenes_levantadas, bool:X) − > Lista(Imagen)*

```
busqueda(x, y, Lista(Imagen):Imagenes_levantadas, bool:X) -> Lista(Imagen):res
{
    dependiendo si es por X o por Y                 // O(1)
        Lista(int):resIndices;                      // O(1)
        buscarIndices(x, raiz, resIndices);         // O(log(n) + k)

        desde i=0 a Tam(resIndices) hacer           // itera k veces
        {
            si esta entre los valores de y o x      // O(1)
                agregar imagen al resutado          // O(1)
            borrar del vector EstanImagenes         // O(1)
        }
}
```

Este algoritmo es simple, consigue los indices de las imagenes que debe agregar segun una de las coordendas (x o y) y vesifica luego que cumpla la otra coordenada tambien

Complejidad: $O(2 + log(n) + k + 3k) = O(log(n) + k)$

Índice