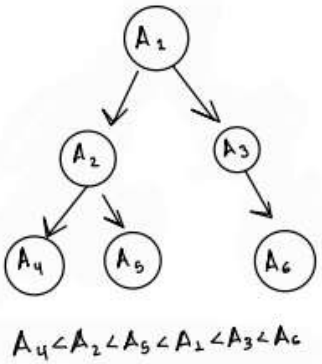
TAD BST Tree

BST TREE = {a1, a2, a3 ... aN}

- A1 es el principal elemnto, A2 y A3 son sub arboles de A1, cualquier elemnto menor a A1 va hacia la izquierda, y si es mayor a A1 va a la derecha



Inv: (a1> a2, a1 <a3) para cualquier árbol y subárbol BST, a la izquierda del elemento, los elementos son menores que y a la derecha del elemento, los elementos son mayores que

Primitive operations:

-createBST:	->BST
-Insert: Element x BST	->BST
-delete: Element x BST	->BST
-search: BST	->Elemento
-searchElement: BST	->Elemento

Insert(K key,E newItem) : Modifier

"Inserte una nueva clave dentro del árbol binario, si la clave ya existe, inserte una nueva posición"

{ pre: Binary Tree initializated }
{ post: Increments the depth of the branch with +1 in this specific sub-tree }

Delete(K key): Modifier

"Eliminar un elemento o clave específicos del árbol binario"

{ pre: árbol binario inicializado}
{ post: disminuye la profundidad de la rama con -1 en este subárbol específico}

Search(K key): Analayzer

"Busca un valor de clave específico dentro del árbol binario y lo devuelve

{pre: Binary Tree initializated}
{post: Devuelve la ArrayList de elementos o devuelve "False" si la clave no existe}

SearchElement(K key): Analyzer

"Busca un elemento específico con un valor clave único y lo devuelve"

{ pre: Árbol binario inicializado }
{ post: Elemento: El elemento con el valor clave específico, si el elemento no existe, devuelve Falso}

CreateBST( ) : Constructor

"Crear (inicializar) un nuevo árbol binario vacío para agregar nuevos elementos"

{ pre: TRUE}
{ post: NewTree: El nuevo árbol binario creado listo para agregar nuevos elementos }