




The *Map* ADT

Model: A collection of values, each mapped by a key

Operations:

- $m = \text{Map}()$ Creates an empty map
- $m[k] = v$ adds value v with key k to m .
Insert  If k already exists in m , it replaces the value already associated with k , to be v
- $m[k]$ returns the value associated to k in m ,
Find  or raises *KeyError*, if k is not in m
- $\text{del } m[k]$ removes the value associated to k in m ,
Delete  or raises *KeyError*, if k is not in m
- $\text{len}(m)$ returns the number of entries in m
- $\text{iter}(m)$ returns an iterator that allows to iterate over the keys in m

Data Structures for *Map* ADT

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$

Data Structures for *Map* ADT

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap			

Binary Search Trees

Binary Search Trees

Definition:

Binary Search Trees

Definition:

Let T be a binary tree.

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

- 1.
- 2.

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the left subtree of n are less than the key stored in n
- 2.

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the left subtree of n are less than the key stored in n
2. All keys stored in the right subtree of n are greater than the key stored in n

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the **left** subtree of n are **less than** the key stored in n
2. All keys stored in the right subtree of n are greater than the key stored in n

Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the **left** subtree of n are **less than** the key stored in n
2. All keys stored in the **right** subtree of n are **greater than** the key stored in n

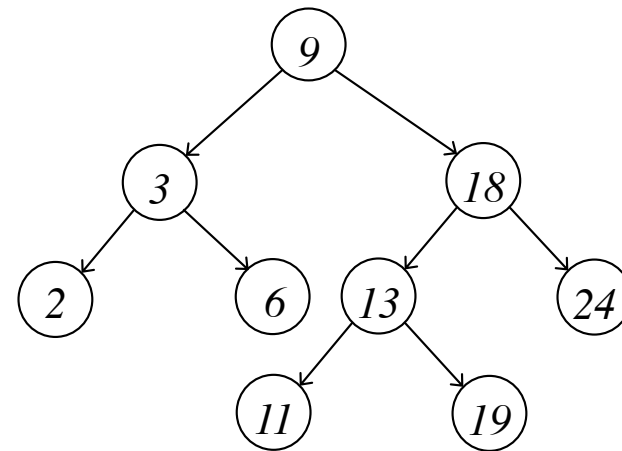
Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the **left** subtree of n are **less than** the key stored in n
2. All keys stored in the **right** subtree of n are **greater than** the key stored in n



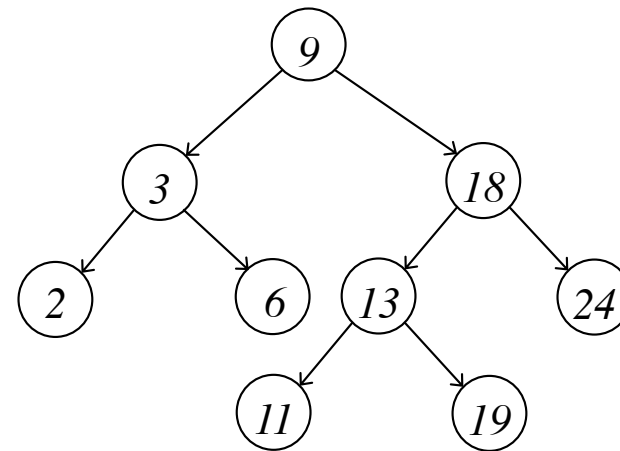
Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the **left** subtree of n are **less than** the key stored in n
2. All keys stored in the **right** subtree of n are **greater than** the key stored in n



Not a Binary Search Tree

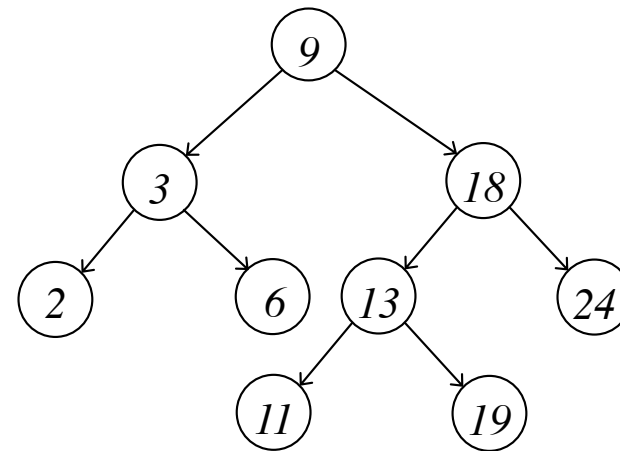
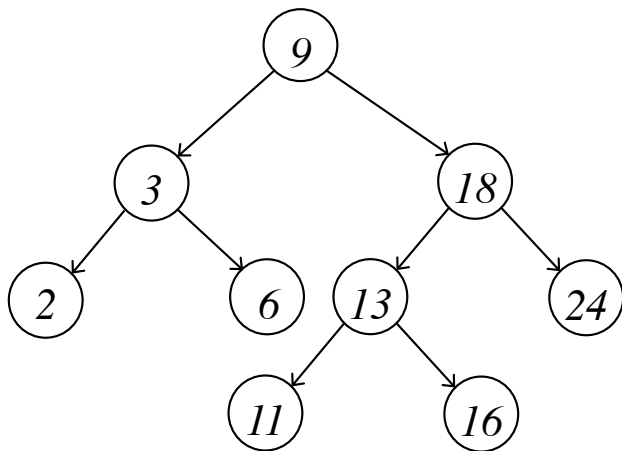
Binary Search Trees

Definition:

Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

1. All keys stored in the **left** subtree of n are **less than** the key stored in n
2. All keys stored in the **right** subtree of n are **greater than** the key stored in n



Not a Binary Search Tree

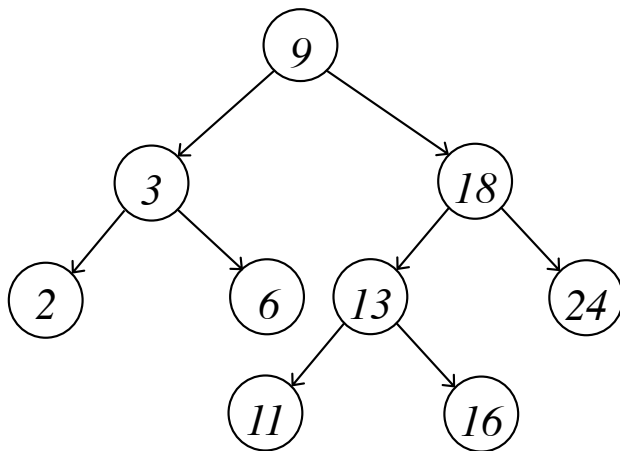
Binary Search Trees

Definition:

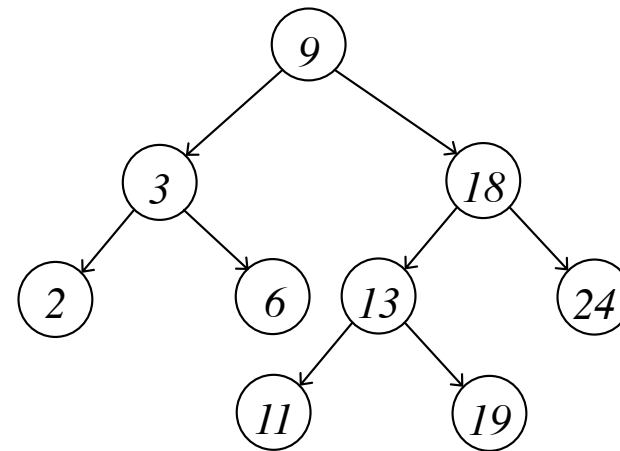
Let T be a binary tree.

We say that T is a *Binary Search Tree*, if **for each** node n in T :

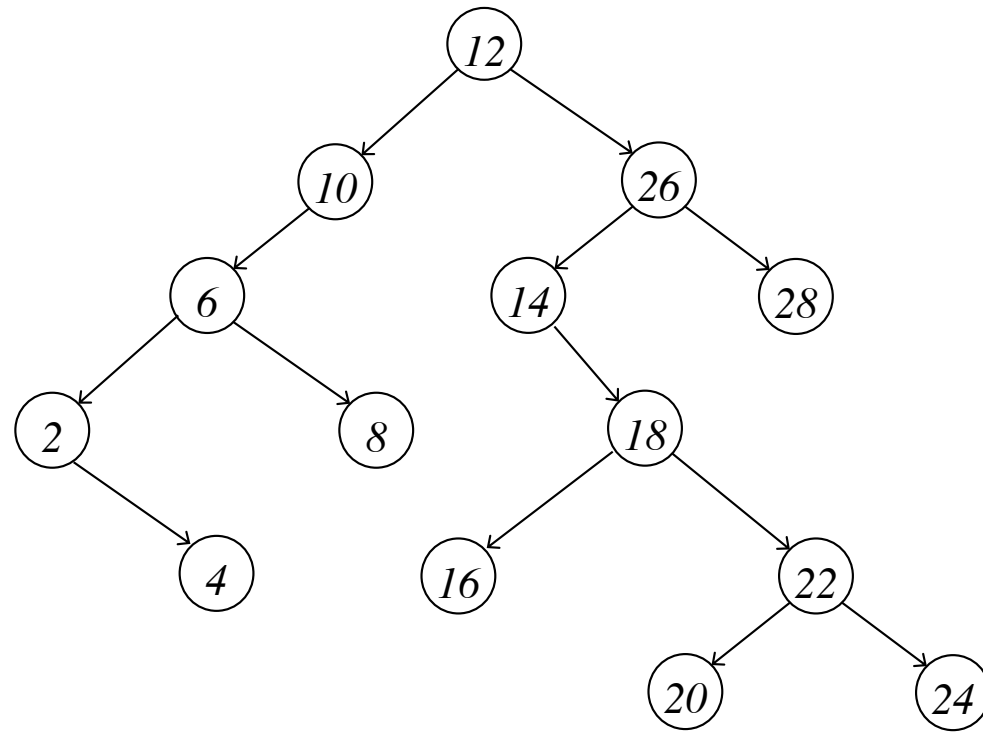
1. All keys stored in the **left** subtree of n are **less than** the key stored in n
2. All keys stored in the **right** subtree of n are **greater than** the key stored in n

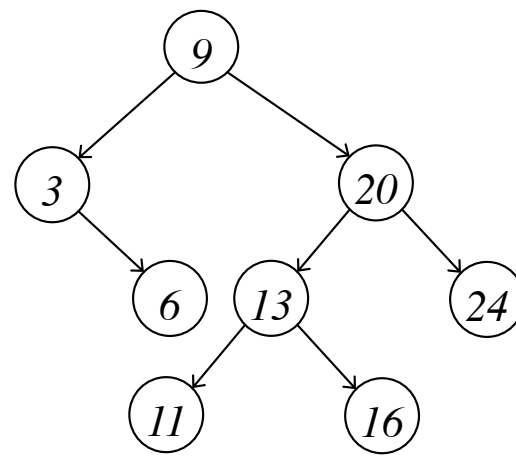


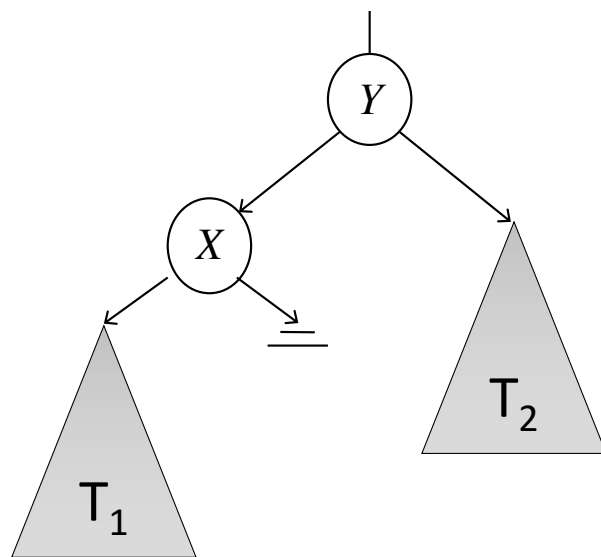
A Binary Search Tree

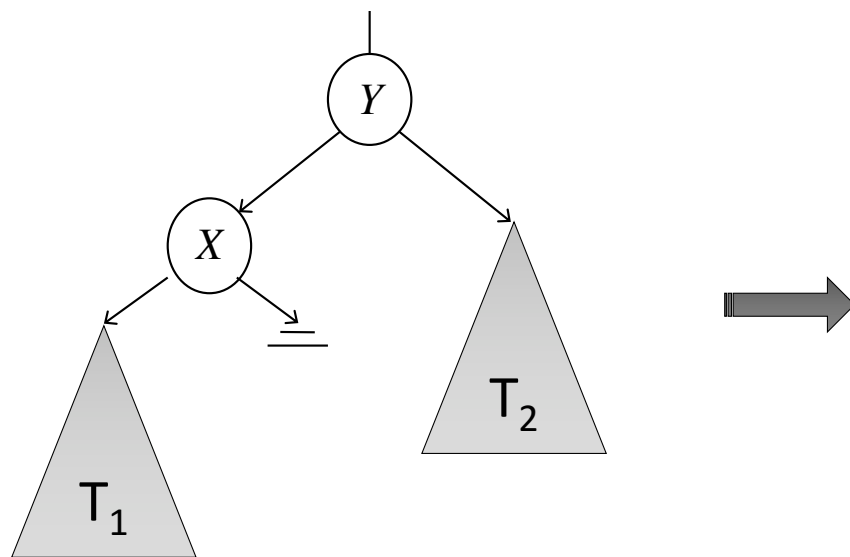


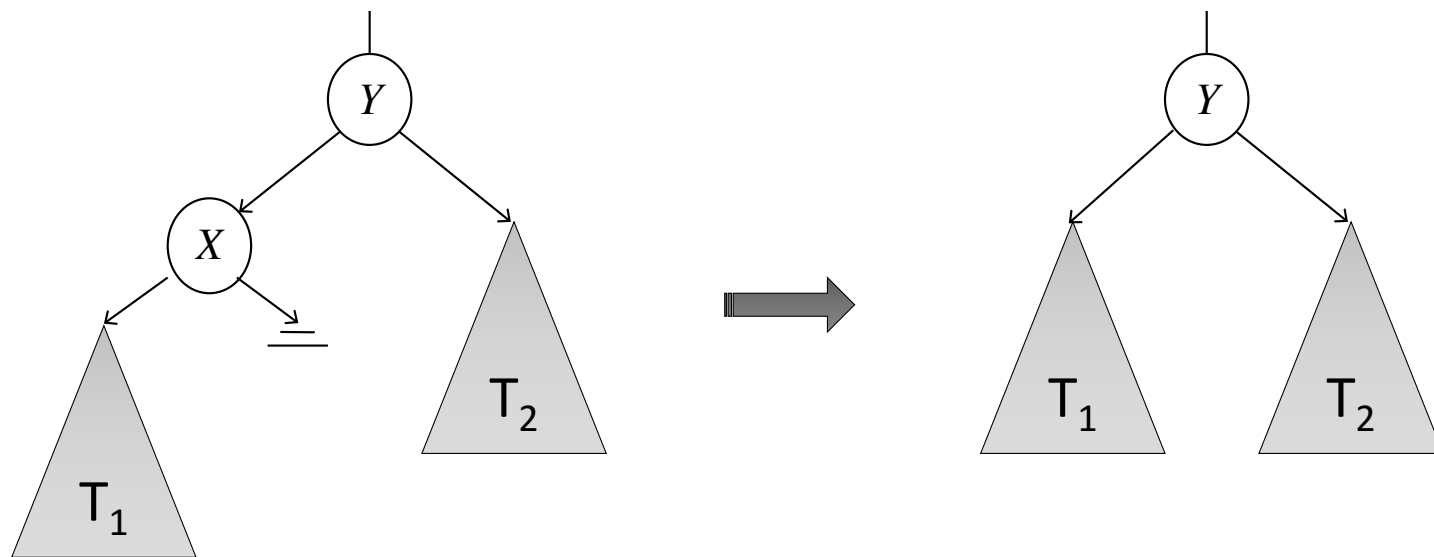
Not a Binary Search Tree



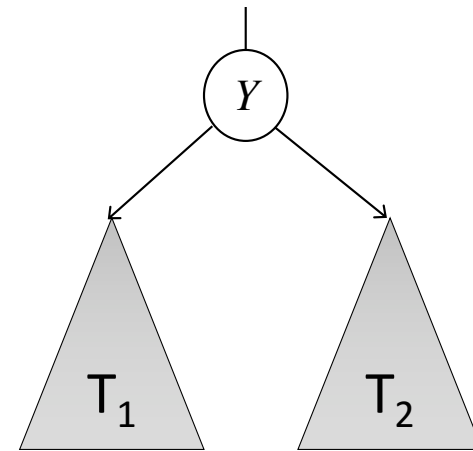
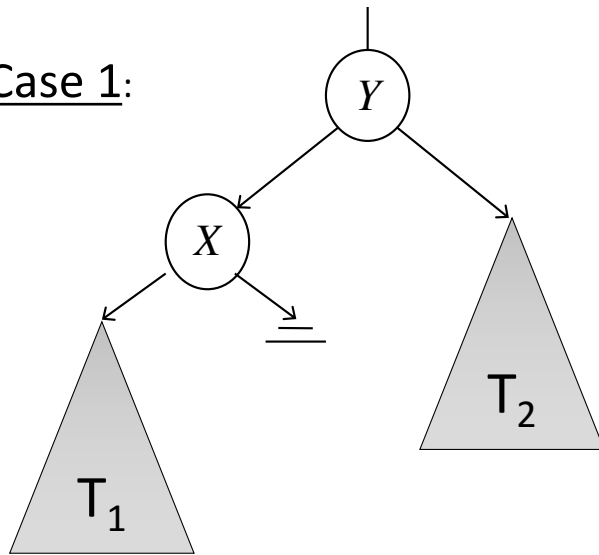




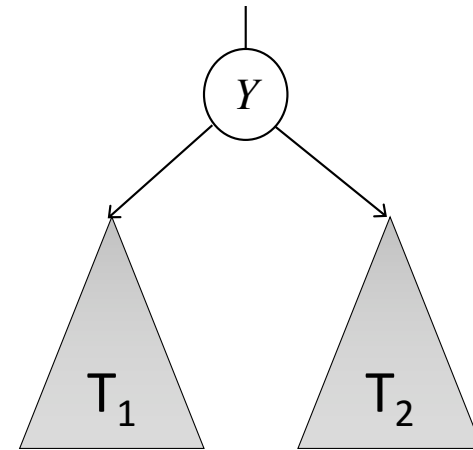
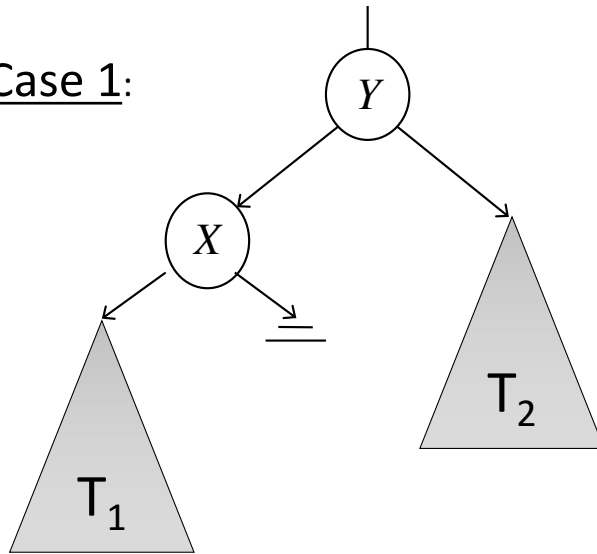




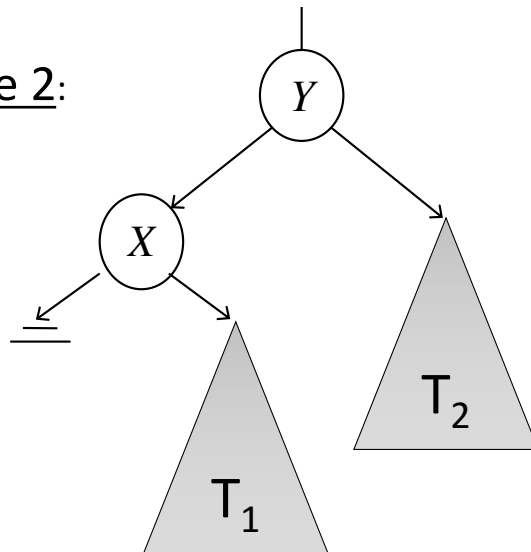
Case 1:



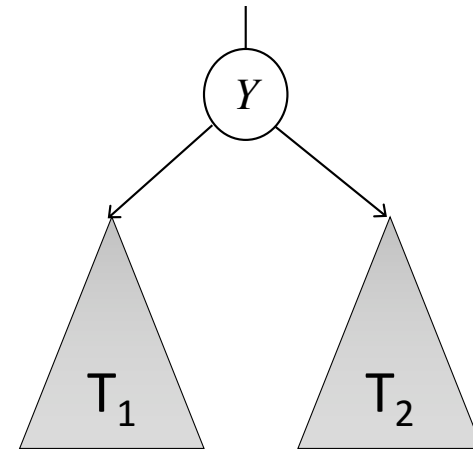
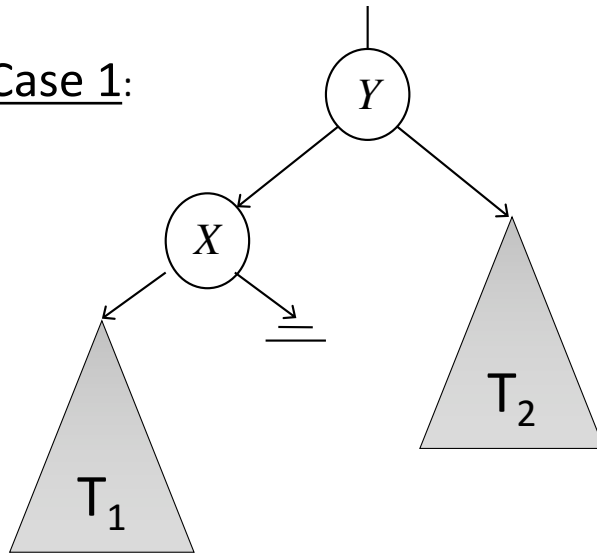
Case 1:



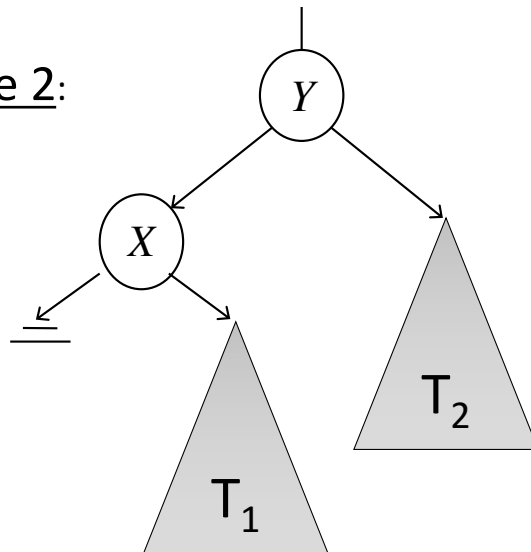
Case 2:



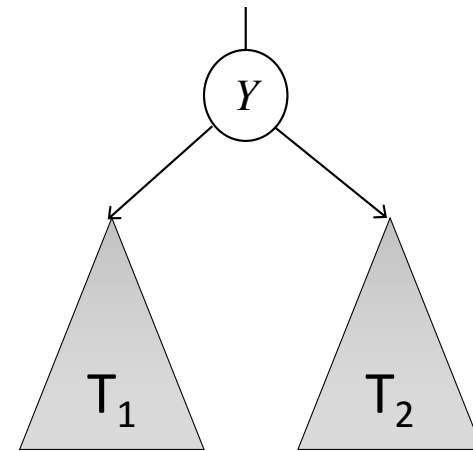
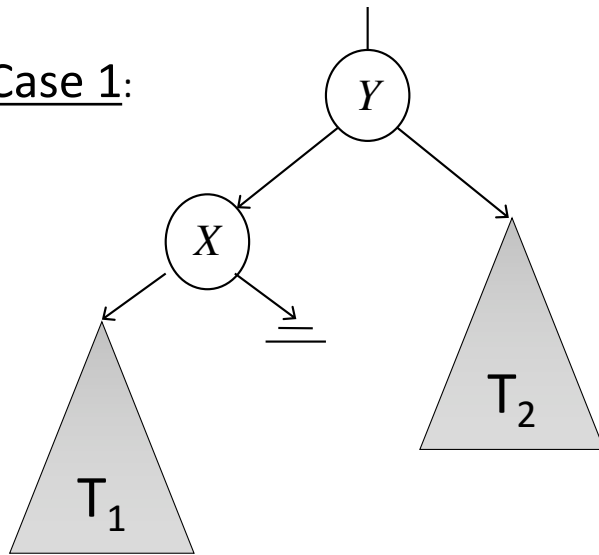
Case 1:



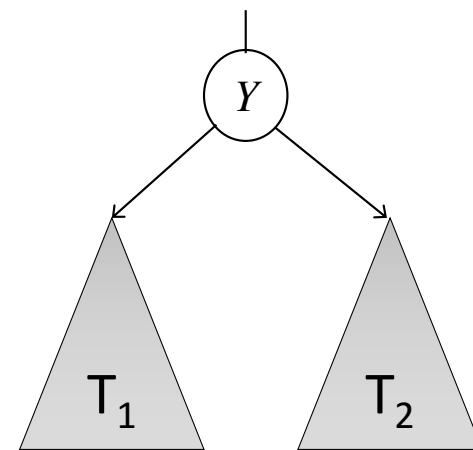
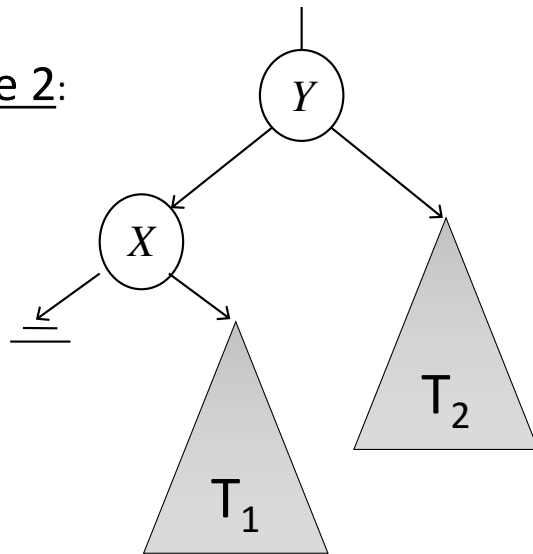
Case 2:



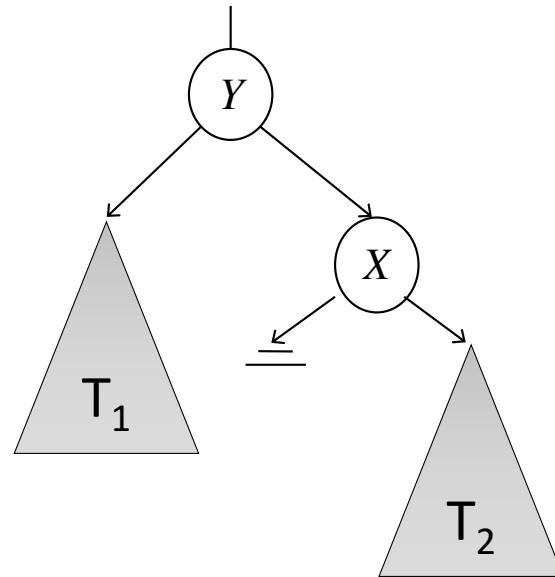
Case 1:



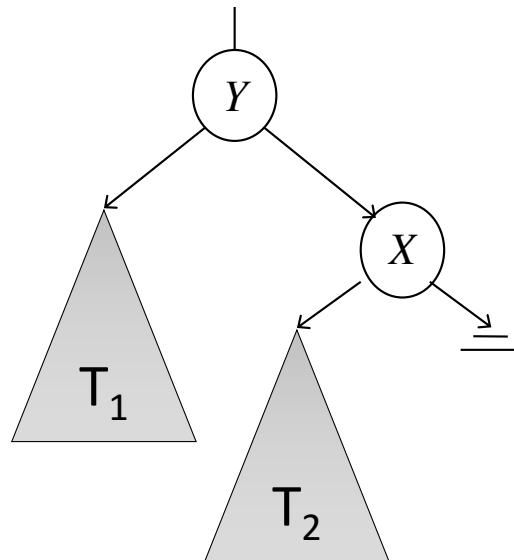
Case 2:



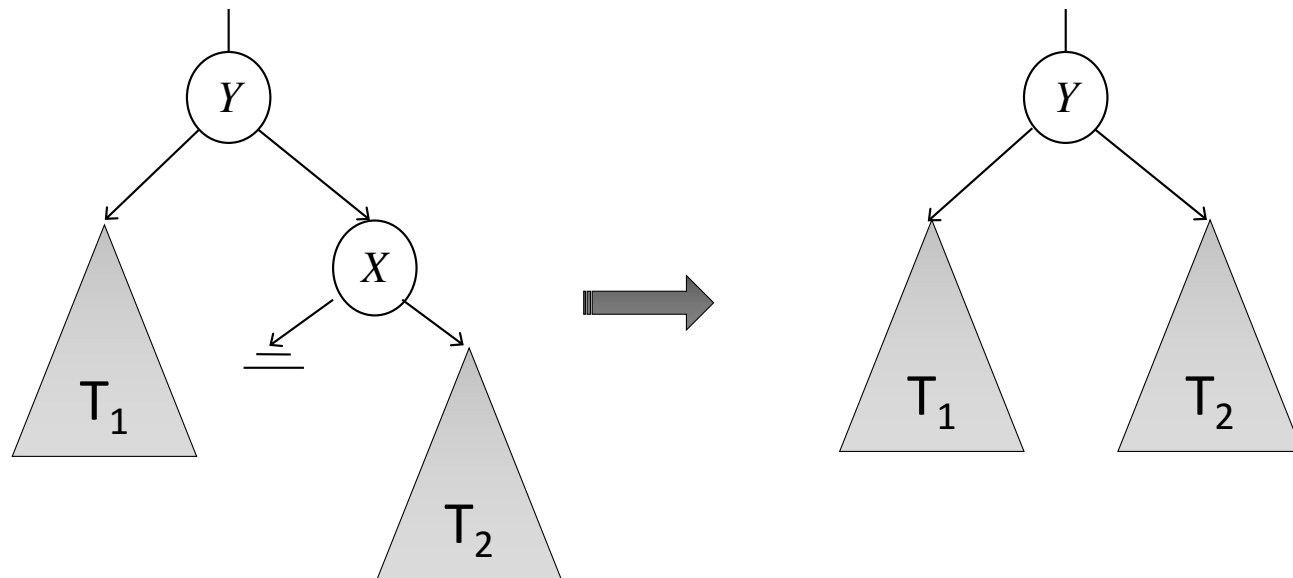
Case 3:



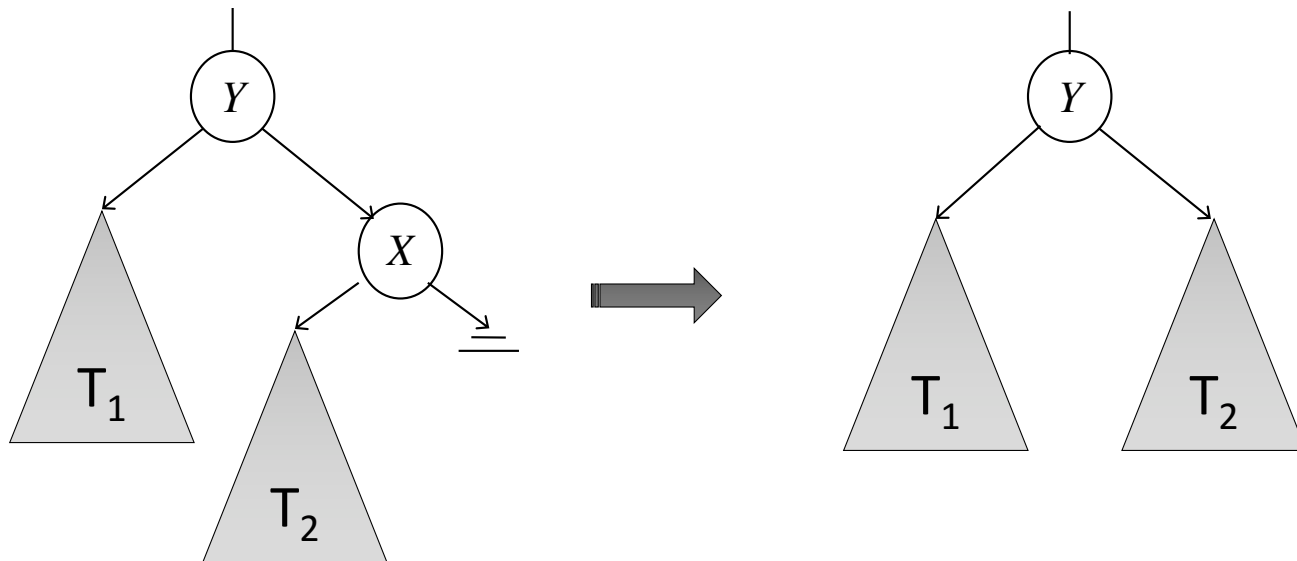
Case 4:



Case 3:



Case 4:



Data Structures for *Map* ADT

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap			

Data Structures for *Map* ADT

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap	$\theta(n)$ Also, $\theta(h)$		

Data Structures for *Map* ADT

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$	

Data Structures for *Map* ADT

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$

Data Structures for *Map ADT*

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$
AVLTreeMap			

Data Structures for *Map ADT*

	Find	Insert	Delete
UnsortedArrayMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
UnsortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
SortedArrayMap	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$
SortedLinkedListMap	$\theta(n)$	$\theta(n)$	$\theta(n)$
BinarySearchTreeMap	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$	$\theta(n)$ Also, $\theta(h)$
AVLTreeMap	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$