DATA STRUCTURE AND ALGORITHMS

LECTURE 7

Searching and Binary Search Tree

Reference links:

https://cs.nyu.edu/courses/fall17/CSCI-UA.0102-007/notes.php

https://www.comp.nus.edu.sg/~stevenha/cs2040.html

https://visualgo.net/en/bst

[M.Goodrich, chapter 11]

Lecture outline

- Searching Algorithms
  - Sequential Search
  - Binary Search
- Binary Search Tree (BST)
  - BST properity
  - BST operations
  - BST applications
- Balanced Search Tree
- Other Search Trees

Searching Algorithms

- Sequential Search
- Binary Search

Searching Problem

- Search problem: find an item or group of items with specific properties within a collection of items. The question can be:
  - Appear or not? How many times? At which positions?
- Collection of items
  - Store in a table: 1-dimention (list), 2-dimention (matrix)
  - Implement by array or linked list
- Search algorithm:
  - Iterate and compare items
  - But how the list be created? – affect to algorithm complexity

Sequential Search

- Sequential Search or Linear Search (in 1-d)
- Iterate begin from the first of the list until the item is found or the entire list has been searched
- Advantages
  - Algorithm easy to implement
  - List can be in any order
- Disadvantages
  - Inefficient (slow): O(n)

Binary Search

- Binary Search or Bisection Search (in 1-d)
- Using divide and conquer strategy
  - Compare the search value with the middle item of the list
  - Continue search in the half of the list where the value might be apprear.
- Advantages
  - Much more efficient than linear search O(log N)
- Disadvantages
  - List be required in order (be sorted)

Search Algorithm running time

Run time of list operations related to searching:

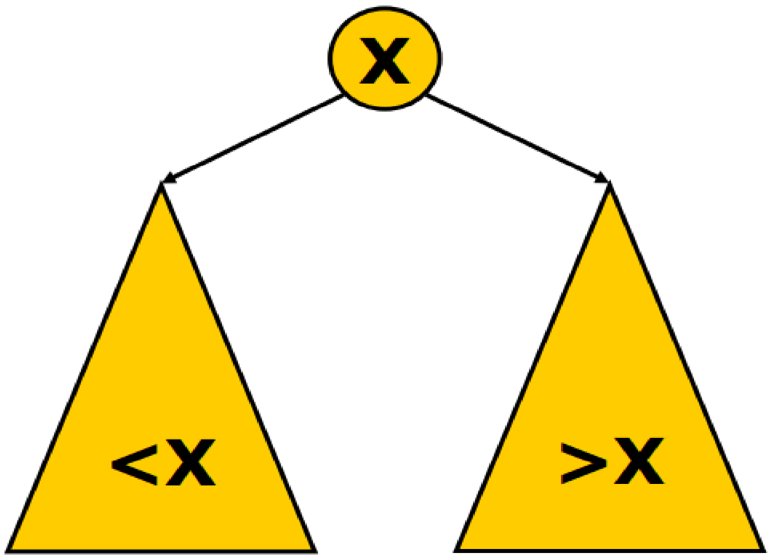
	Unsorted Array/List	Sorted Array	Unsorted Linked List
insert (add)	O(1)	O(n)	O(1)
delete (remove)	O(n)	O(n)	O(1)
search (isContain)	O(n)	O(log n)	O(n)

GOAL: O(log n) for all operations → Binary Search Tree

Binary Search Trees (BST)

[M.Goodrich, sec. 11.1, p. 460]

BST Property

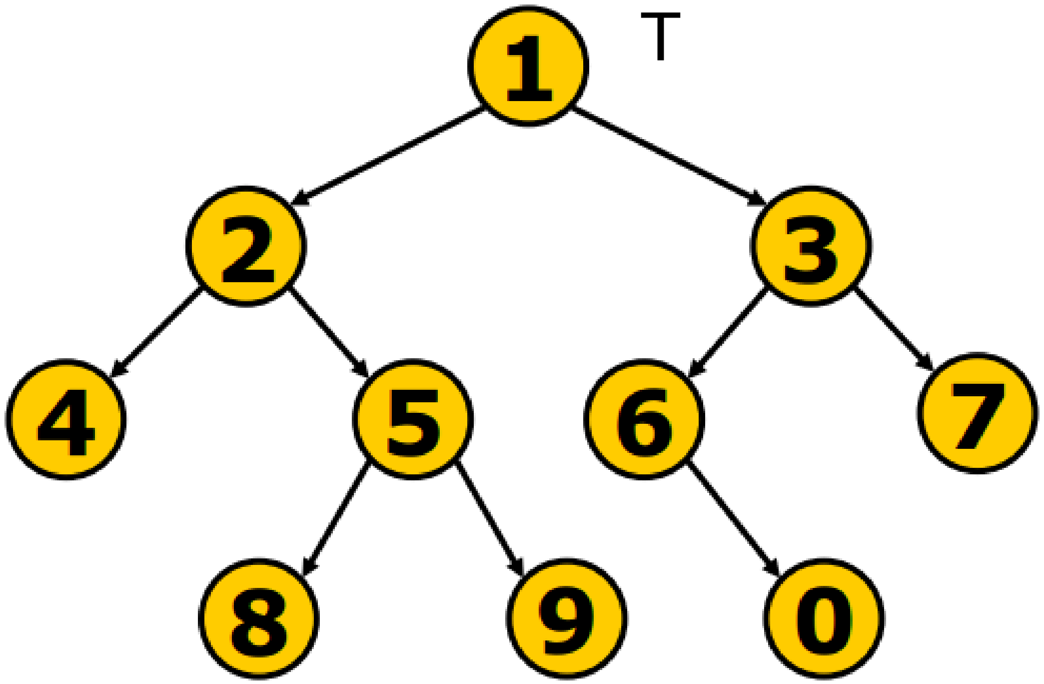


- BST organizes data in a binary tree such that
  - all keys **smaller** than the root are stored in the left subtree, and
  - all keys **larger** than the root are stored in the right subtree.

Q: Can we have the same key values in a BST?

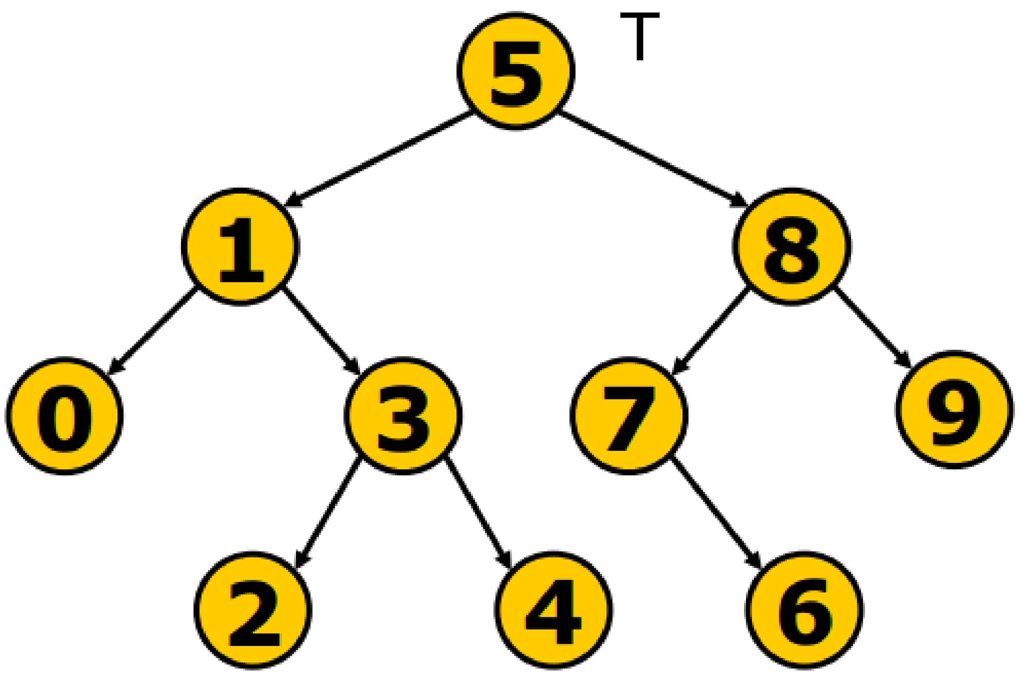
Or: How to handle duplicates in Binary Search Tree?

BST Example



T is a BST or not? Why?

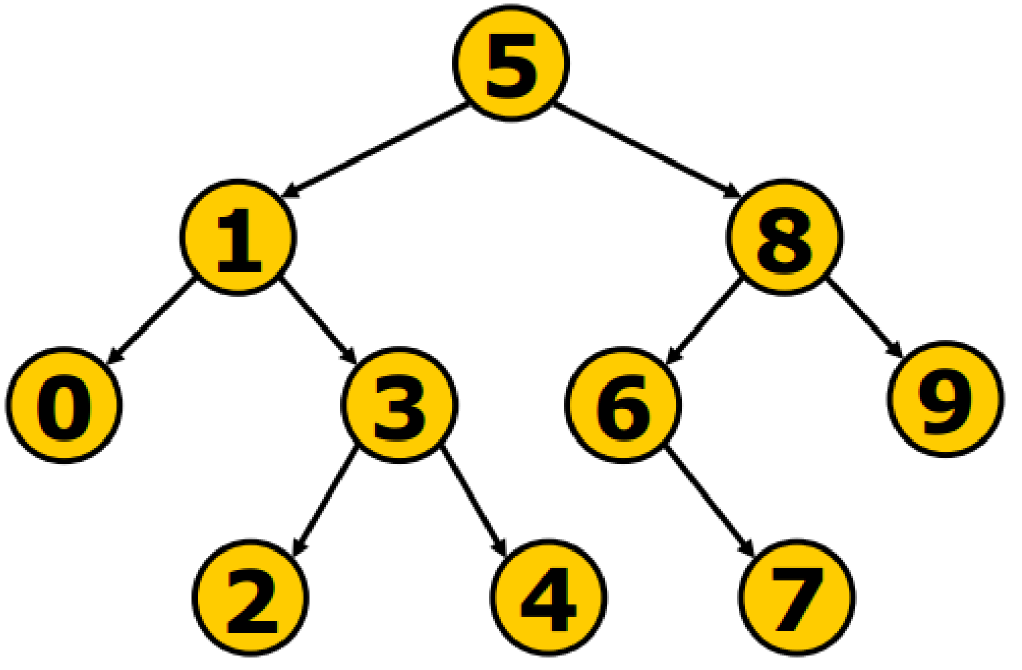
BST Example



And this tree T? No - Why?

How to rearrange it to BST?

BST Example



A BST

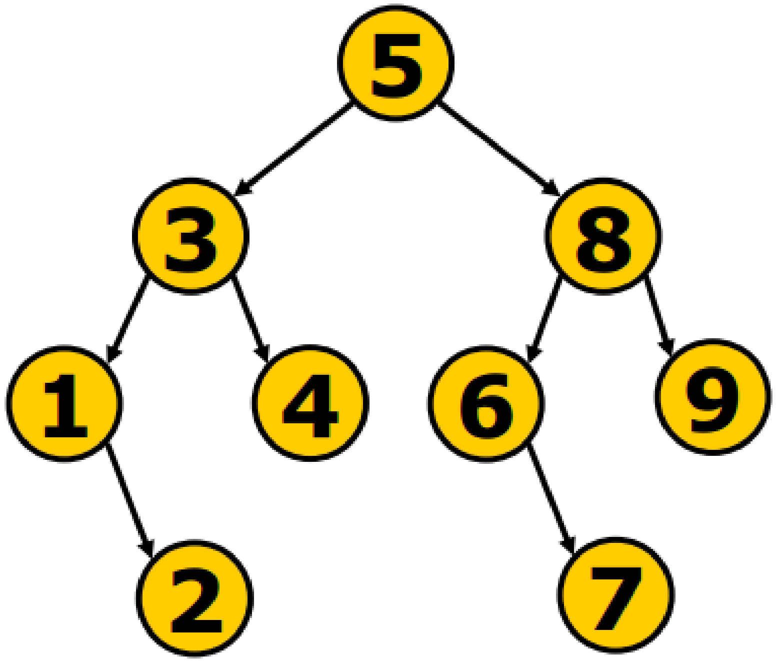
Q: What do you get when traverse a BST in **In-order**?

A: We get list 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

BST Operations

Finding Minimum Element

```
findMin(T) ≡
    while (T.left is not empty)
        T = T.left;
    return T.item;
end.
```



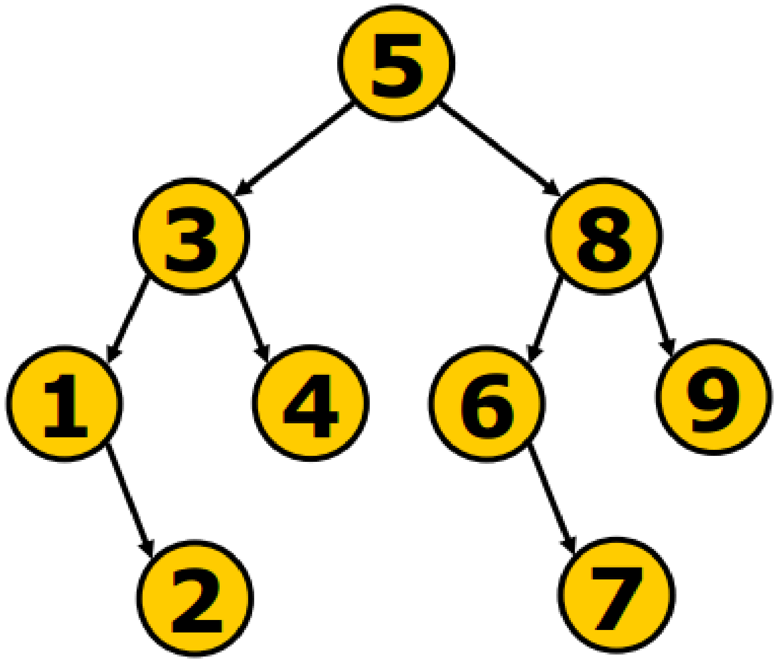
Running time: O(h)

Q: How to find maximum element?

Q: How to find top-k (or bottom-k) element?

Searching x in T (iterative solution)

```
search (x,T) ≡
    while (T is not empty)
        if (x==T.item)
            return T
        else if (x<T.item)
            T = T.left
        else
            T = T.right
    return null // T is empty, so x is not in T
end.
```

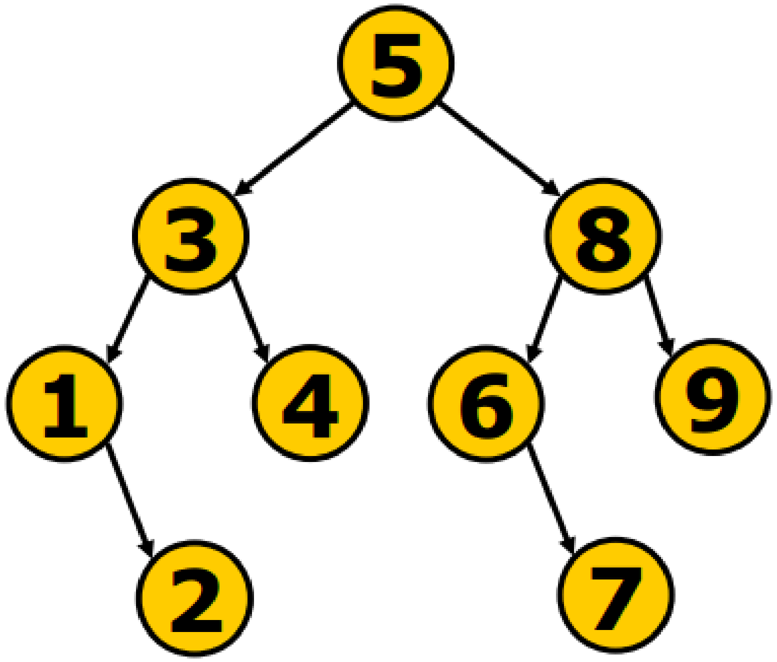


Ex: search(6,T)

Running time: O(h)

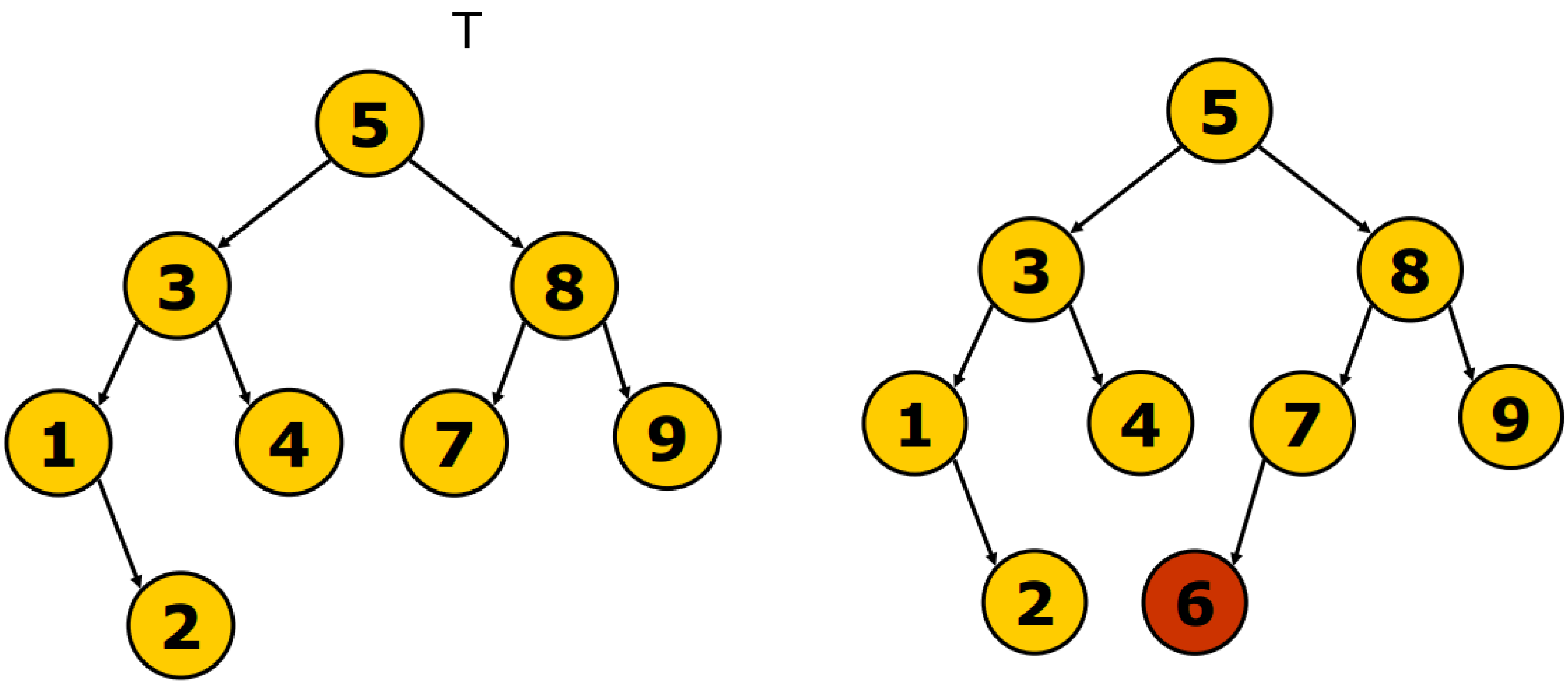
Searching x in T (recursive solution)

```
search (x,T) ≡
    if (T is empty)
        return null
    if (x==T.item)
        return T
    else if (x<T.item)
        return search(x,T.left)
    else
        return search(x,T.right)
end.
```



Running time is O(h), isn't it?

Insert x to T



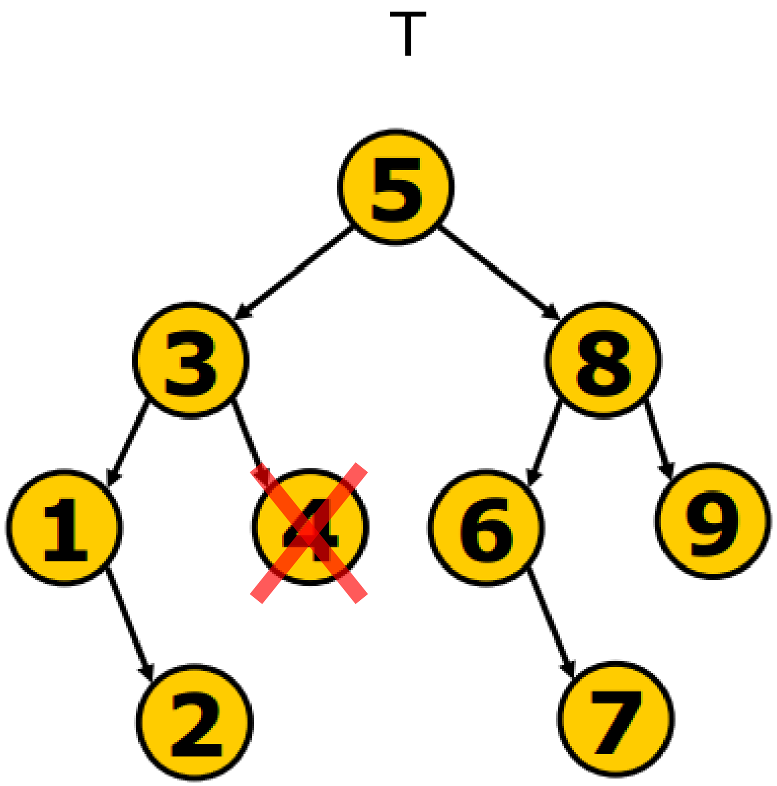
How to insert element with value key **6** to T?

Insert x to T

```
insert (x,T) ≡
    if (T is empty)
        return new TreeNode(x) //a tree with only node x
    else if (x<T.item)
        T.left=insert(x,T.left)
    else if (x>T.item)
        T.right=insert(x,T.right)
    else
        ERROR!  //x already in T
    return T;       //return the new tree T
end.
```

Running time is $O(h)$

Delete x from T



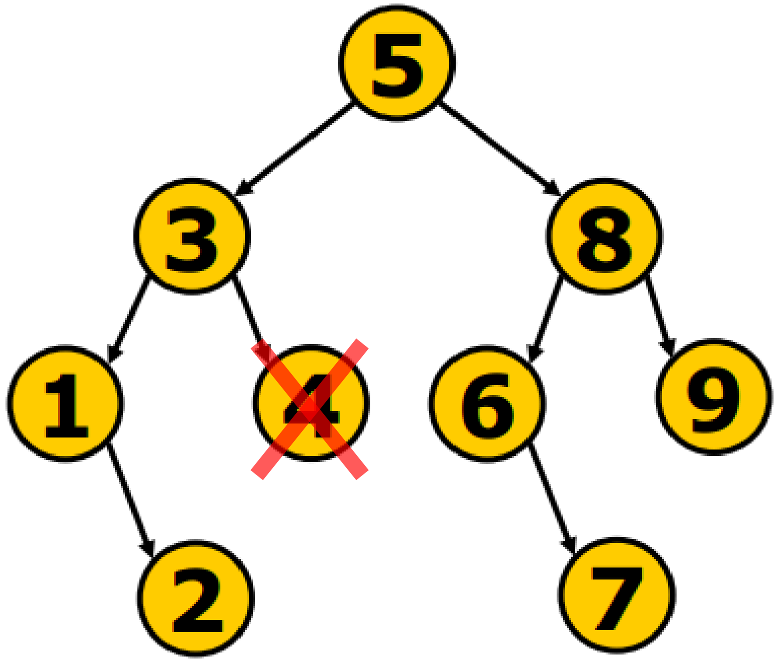
How to delete the element with value key x from T?

Some difference cases from T has children or not...

Delete x from T: case 1

- Node to be deleted has no children. Ex: Delete 4 from T

```
delete (x,T) ≡
    if (T has no children)
        if (T.item == x)
            return empty tree
        else
            NOT FOUND
end.
```

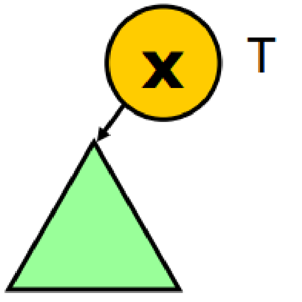


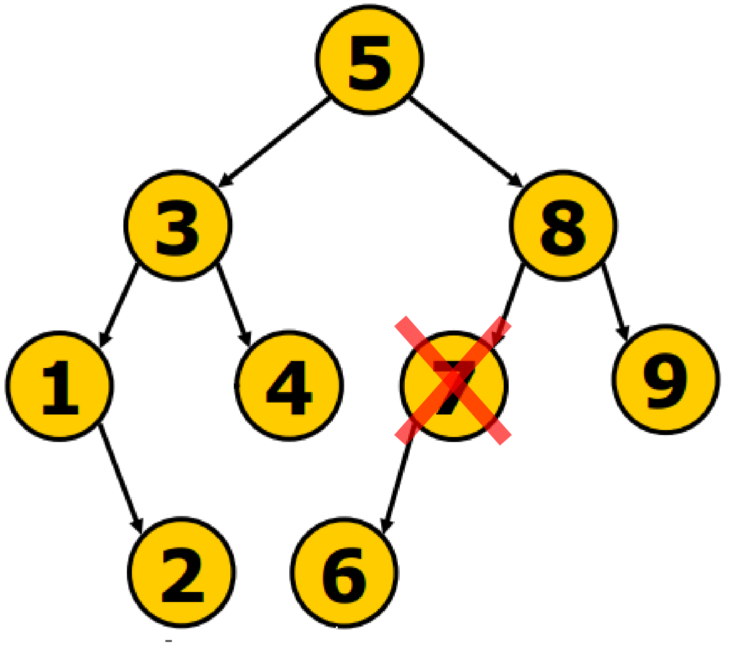
Ex: Delete 4 in tree T

Delete x from T: case 2(a)

- Node to be deleted has only left child

```
delete (x,T) ≡
    if (T has only 1 child (left))
        if (x==T.item)
            return T.left
        else
            T.left=delete(x,T.left)
    return T
end.
```

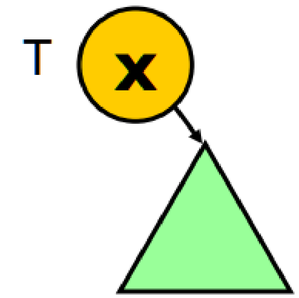




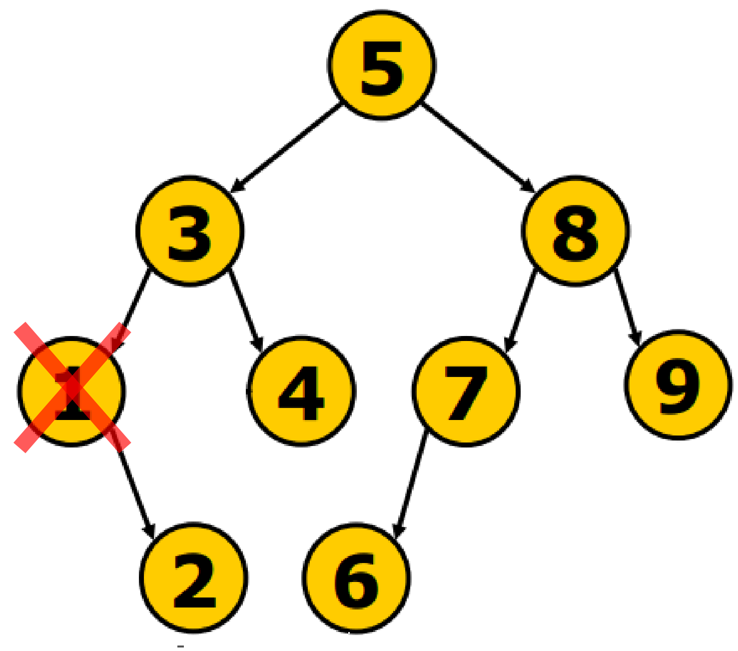
Ex: Delete 7 from tree T

Delete x from T: case 2(b)

- Node to be deleted has only right child



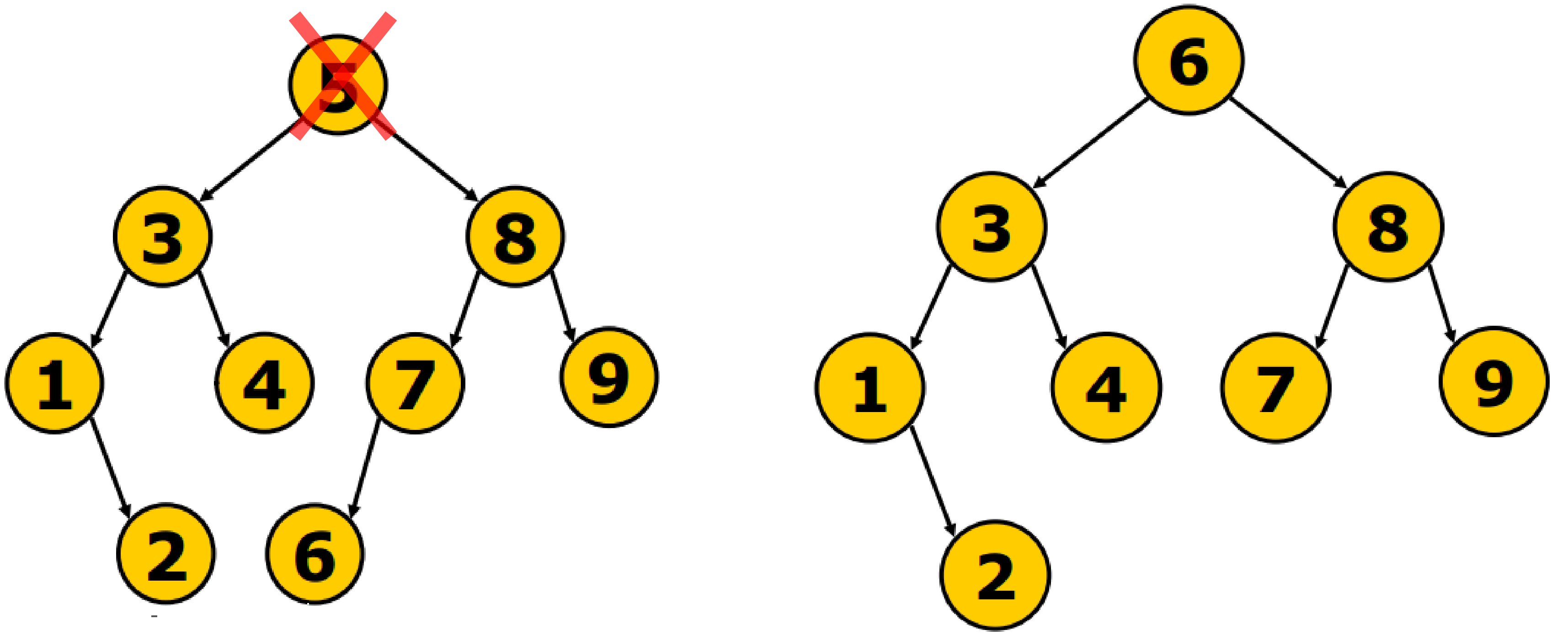

```
delete (x,T) ≡
    if (T has only 1 child (right))
        if (x==T.item)
            return T.right
        else
            T.right=delete(x,T.right)
    return T
end.
```



Ex: Delete 1 from tree T

Delete x from T: case 3

- Node to be deleted has 2 children. Ex: Delete 5 from T



5 deleted from T!

Delete x from T: case 3

```
delete (x,T) ≡
      if (T has two children)
            if (T.item == x)
                  T.item=findMin(T.right) //replace T.item by min of right
                  T.right = delete(T.item, T.right)
            else  if (x<T.item)
                  T.left=delete(x,T.left)
            else
                  T.right=delete(x,T.right)
      return T
end.
```

Running time is $O(h)$

BST algorithms running time

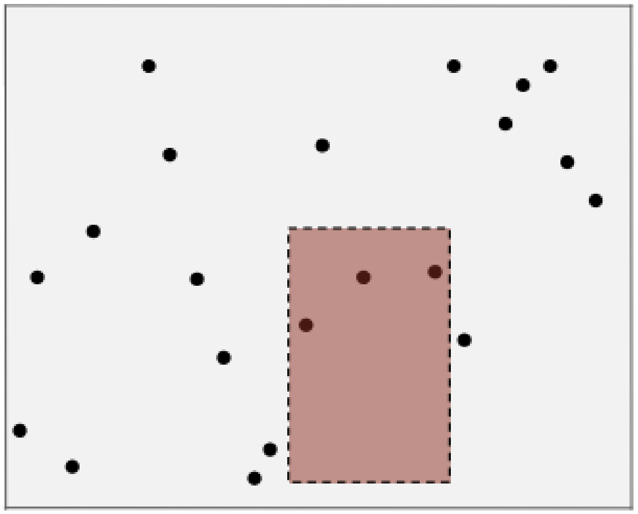
- Running time of BST operations:

	Unsorted Array/List
findMin()	O(h)
search(x,T)	O(h)
insert(x,T)	O(h)
delete(x,T)	O(h)

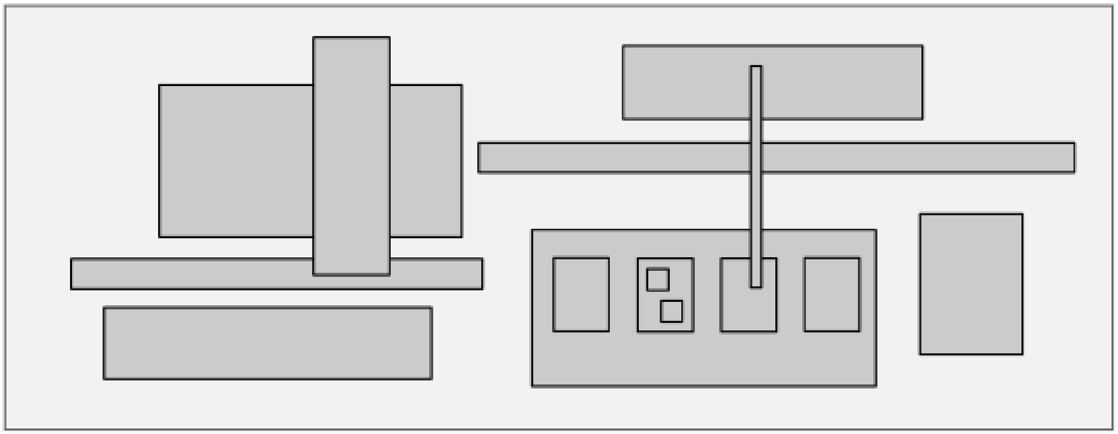
GOAL: O(log n) for all operations – REACHED!

BST Applications

- Intersections among geometric object



2d orthogonal range search



orthogonal rectangle intersection

- Applications: CAD, games, movies, virtual reality, databases, GIS...
- Efficient solutions: Binary search trees (and extensions).

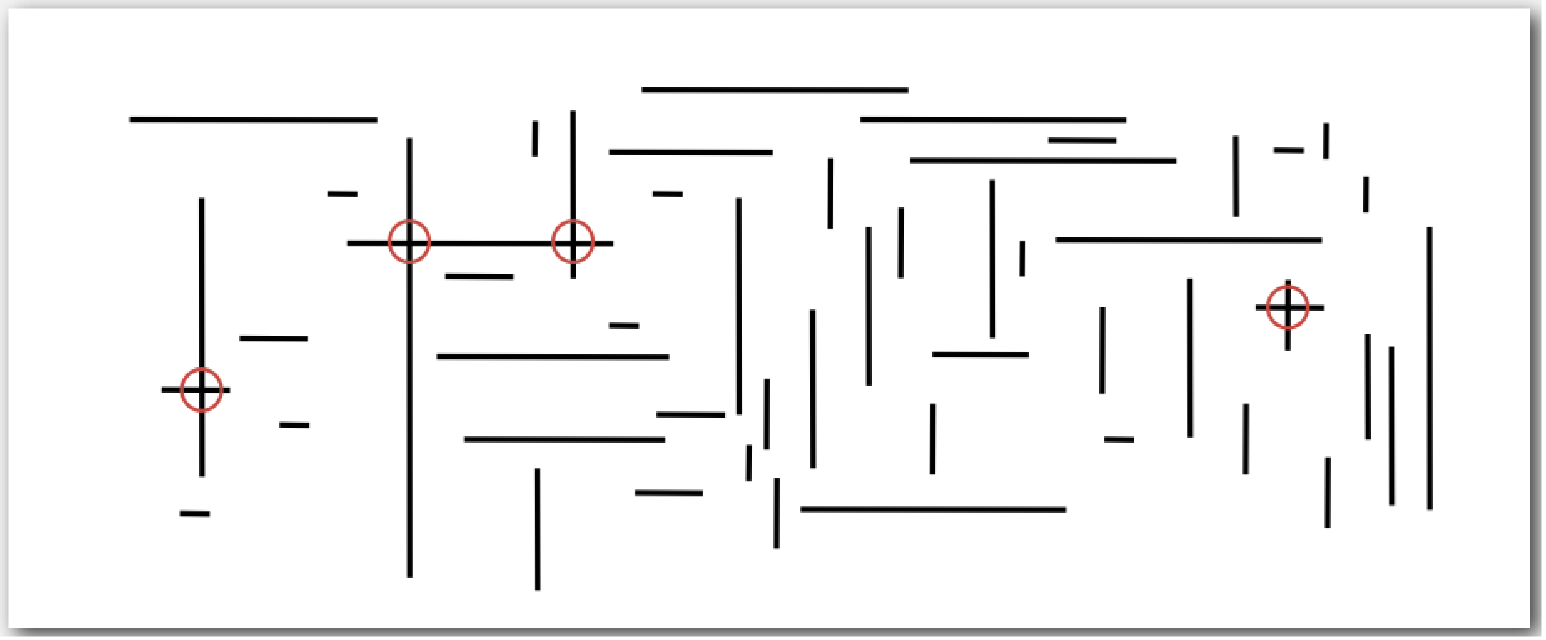
BST Applications

- Geometric Application of BSTs
  - 1-d range search (tìm kiếm 1 chiều)
  - Line segment intersection (giao đoạn thẳng)
  - k-d trees (cây k chiều)
  - Interval search trees (cây tìm kiếm khoảng)
  - Rectangle intersection (giao các hình chữ nhật)

For more detail: Geometric Application of BST.pdf

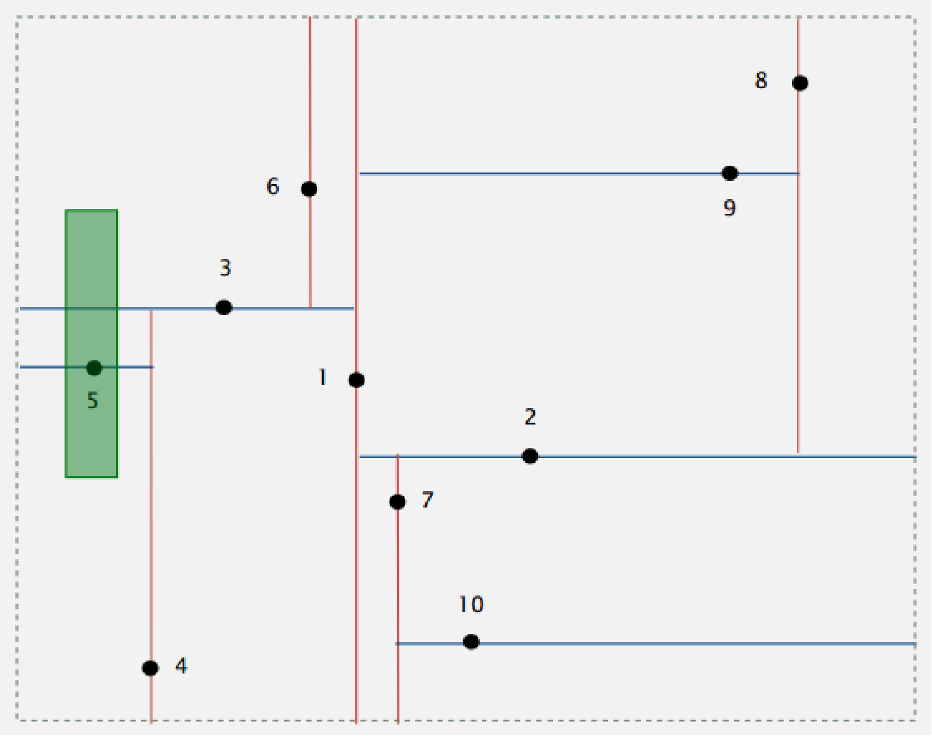
BST Applications

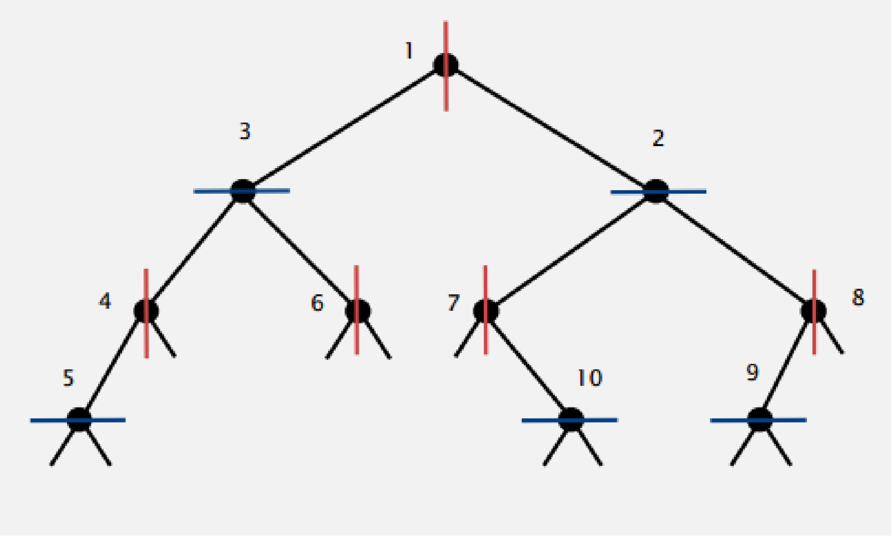
- Line segment intersection



BST Applications

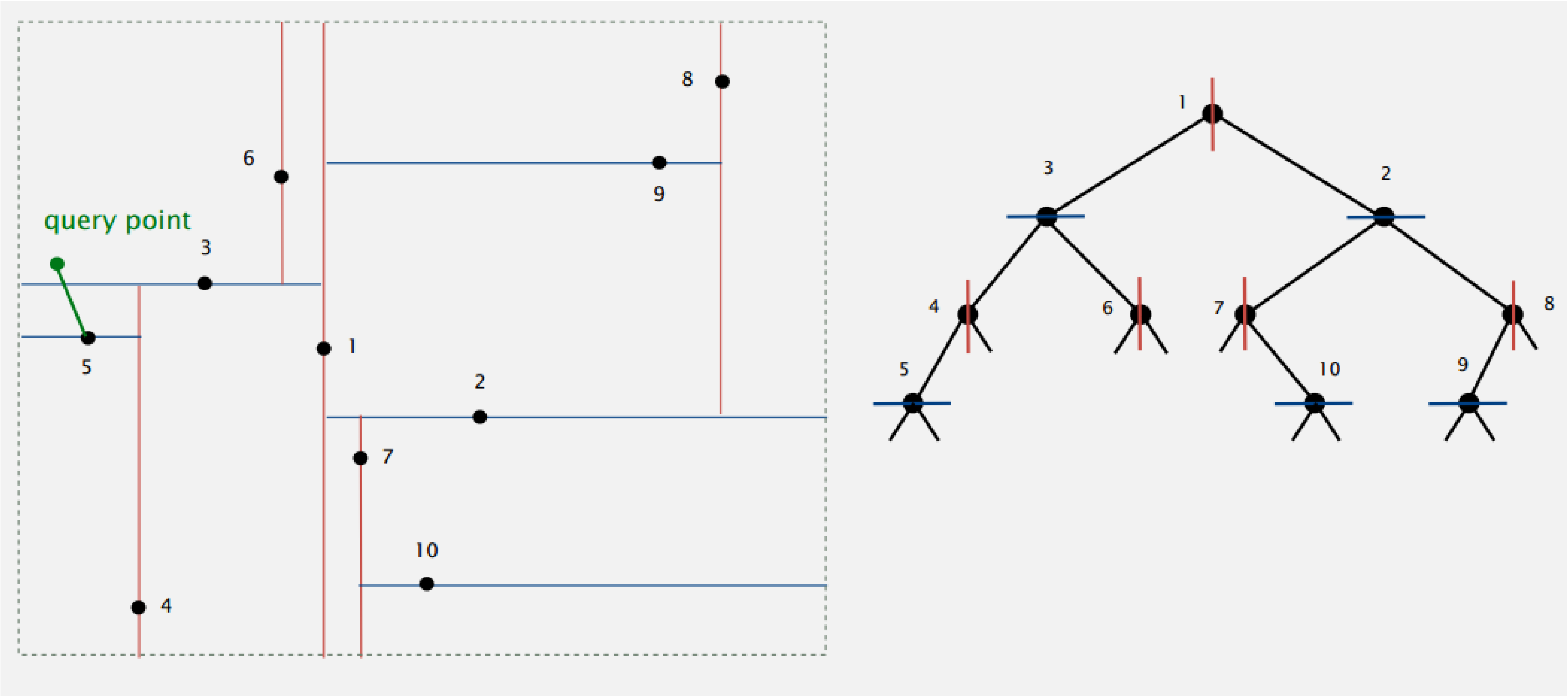
- Range search in a 2-d tree





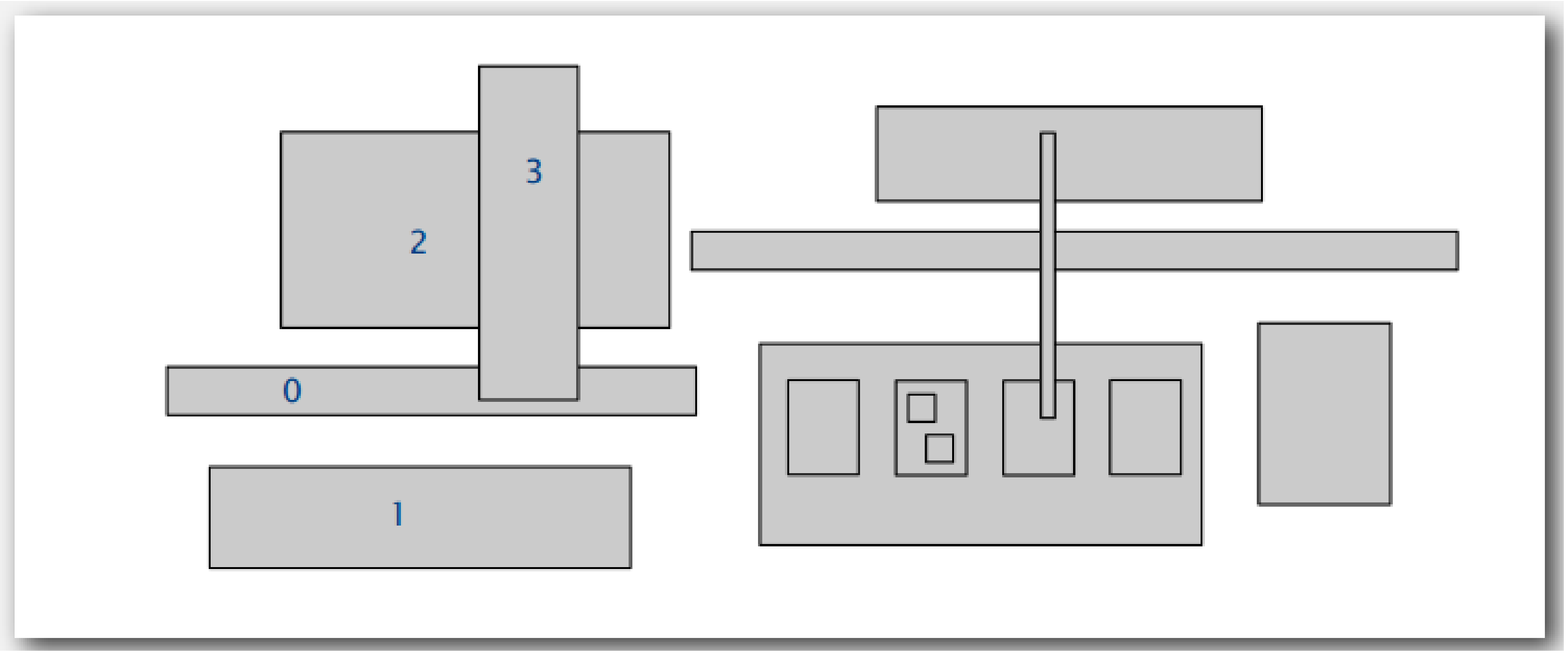
BST Applications

- Nearest neighbor search in a 2-d tree (Láng giềng gần nhất)



BST Applications

- Rectangle intersection



BST Applications

- Interesting natural phenomenon and math algorithm:

Flocking birds and [Boids Algorithm]

https://www.youtube.com/watch?v=4LWmRuB-uNU

BST Applications

For more details: Geometric Application of BST.pdf

Binary Search Tree

Running time of BST operations:

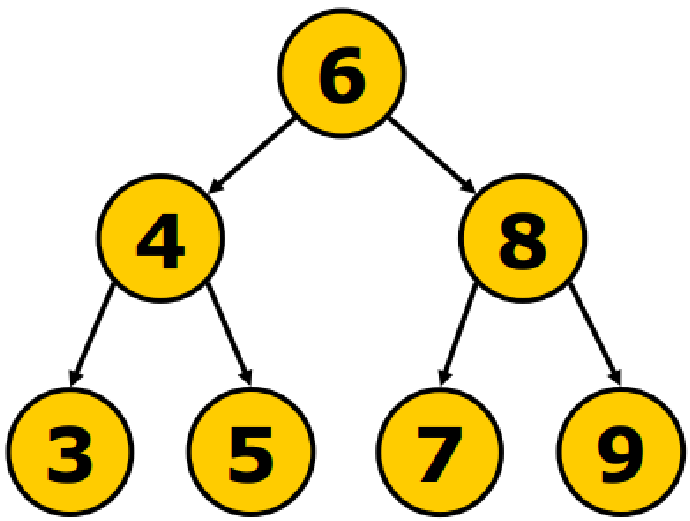
	Unsorted Array/List
findMin()	O(h)
search(x,T)	O(h)
insert(x,T)	O(h)
delete(x,T)	O(h)

GOAL: O(log n) for all opeartions – REACHED!

BUT...!

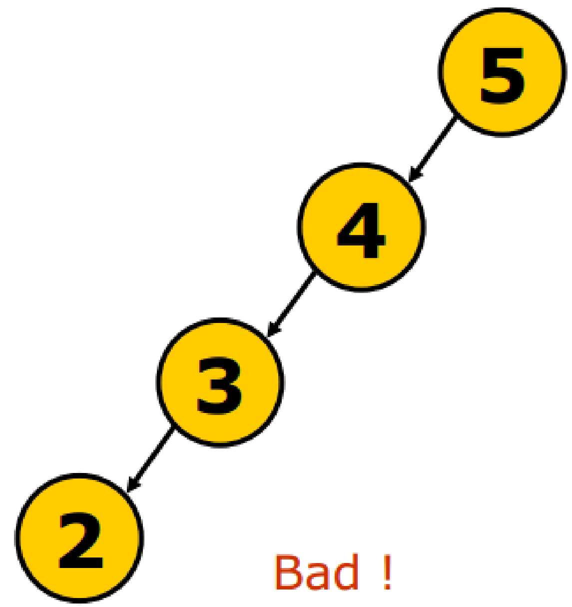
BST algorithms running time

But... h is not always O(log n) with BST size n!!!



Good !

$h = O(\log n)$



Bad !

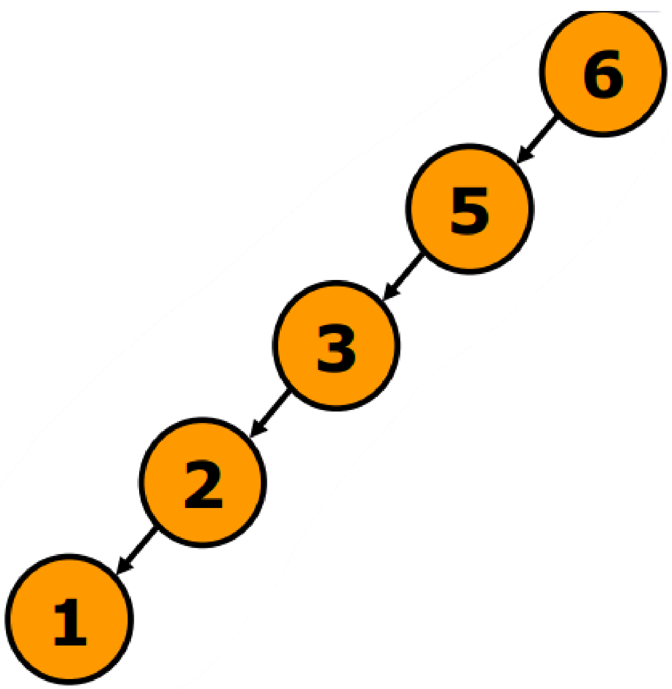
$h = O(n)$

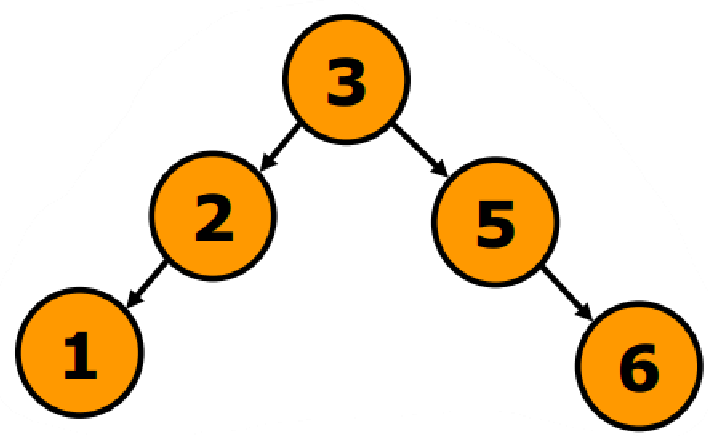
We get **skewed** tree when nodes are inserted in increasing or decreasing order.

BST Solution for worst case

Skewed trees

Balanced trees

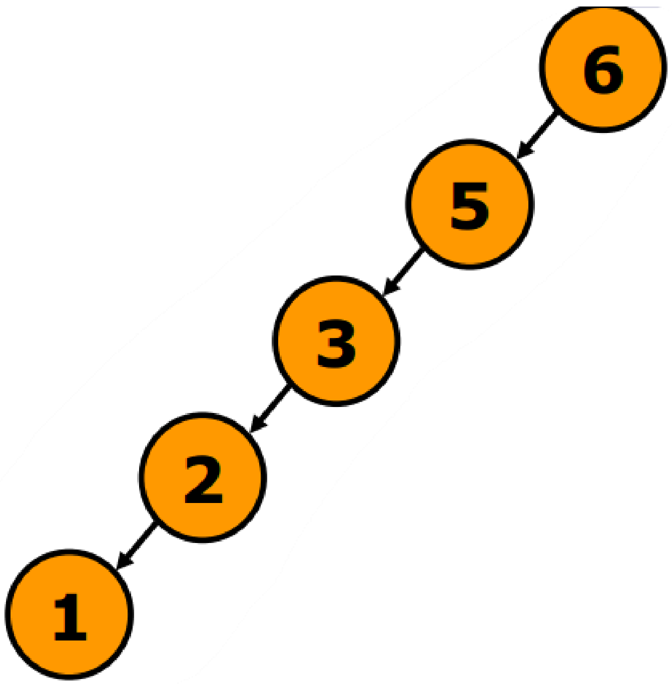


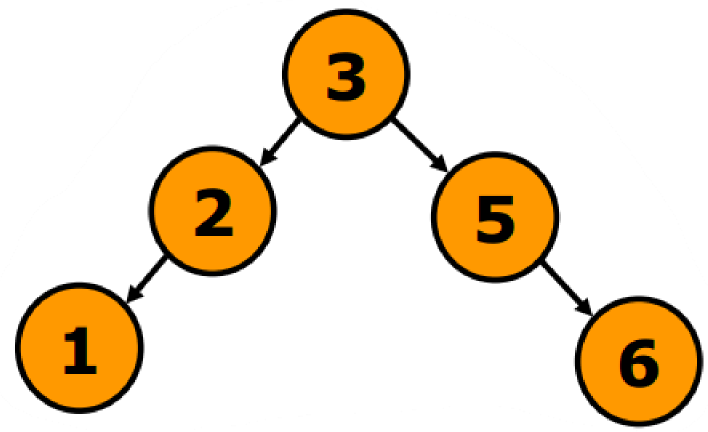


Balanced Search Tree

[M.Goodrich, sec. 11.2, p. 472]

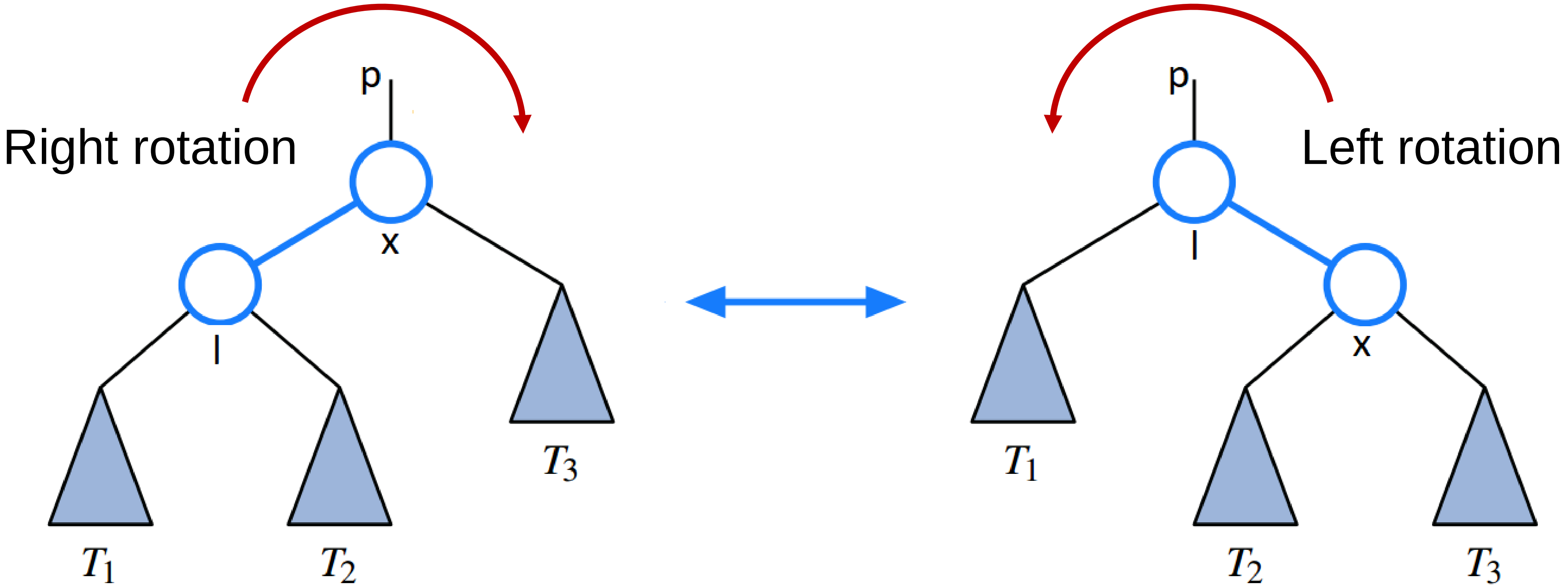
BST Solution for worst case





Skewed tree — convert into through a series of rotations → Balanced tree

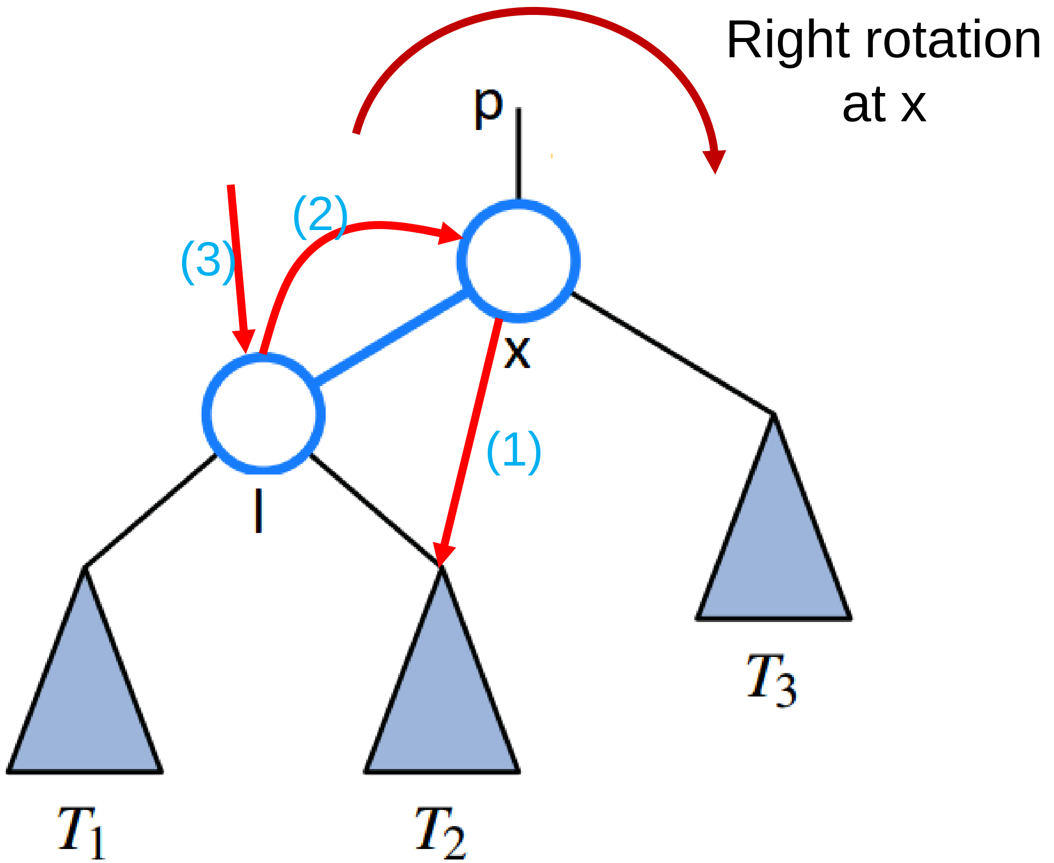
Balanced Tree: Rotation



Rotate: A child to be above its parent

Balanced Tree: Rotation algorithm

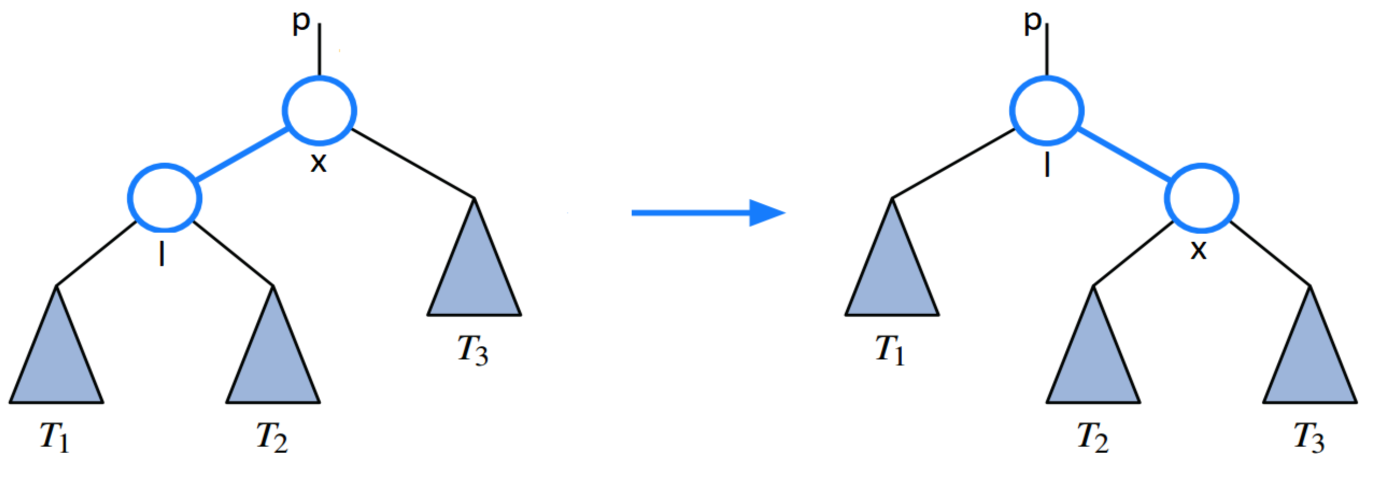
```
rotateRight (x) ≡
    l = x.left
    if (l is empty)
        return
    x.left = l.right //(1)
    l.right = x //(2)
    p = x.parent
    if (x is a left child)
        p.left = l //(3)
    else
        p.right = l //(3)
end.
```



Run time: O(1)

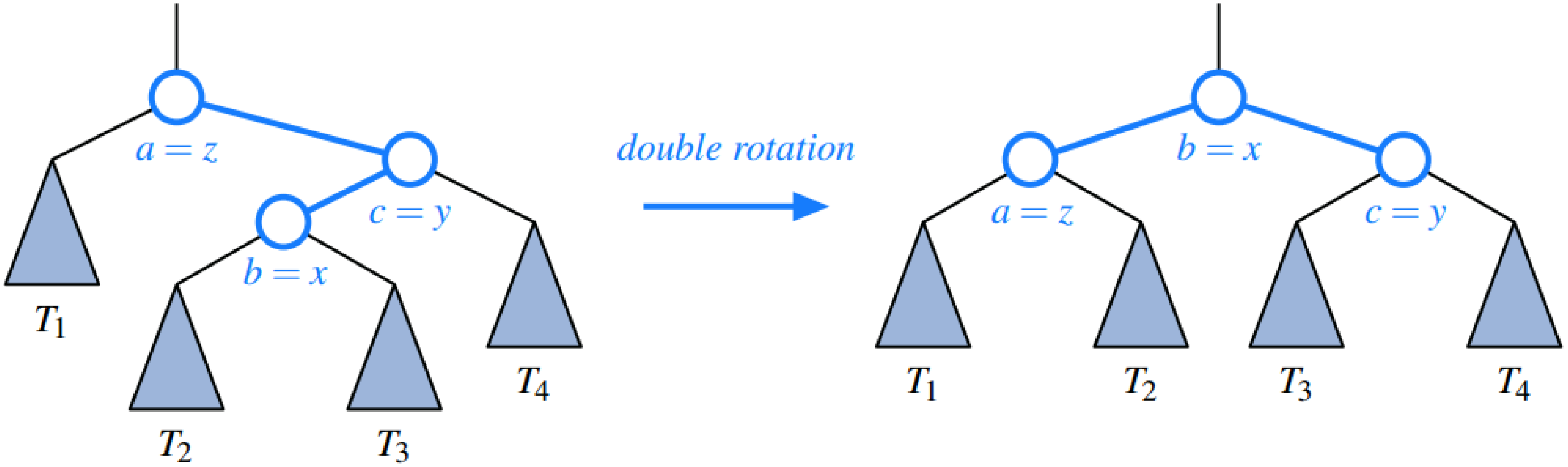
Balanced Tree: Rotation algorithm

- Effect of Rotate Right at x
  - l which is x's left child, and l's left subtree (T_1), move up 1 level
  - x and x's right subtree move down 1 level
  - l's right subtree becomes x's left subtree and remains at the same level
  - x's parent becomes l's parent, and x becomes the right child of l



Balanced Tree: Rotation algorithm

- Rotate Left: same Rotate Right
- Multi Rotate: One or more rotations can be combined to provide broader rebalancing within a tree
  - Trinote restructuring algorithm



Let's study it more by yourself! [M.Goodrich, p. 473]

Other Search Trees

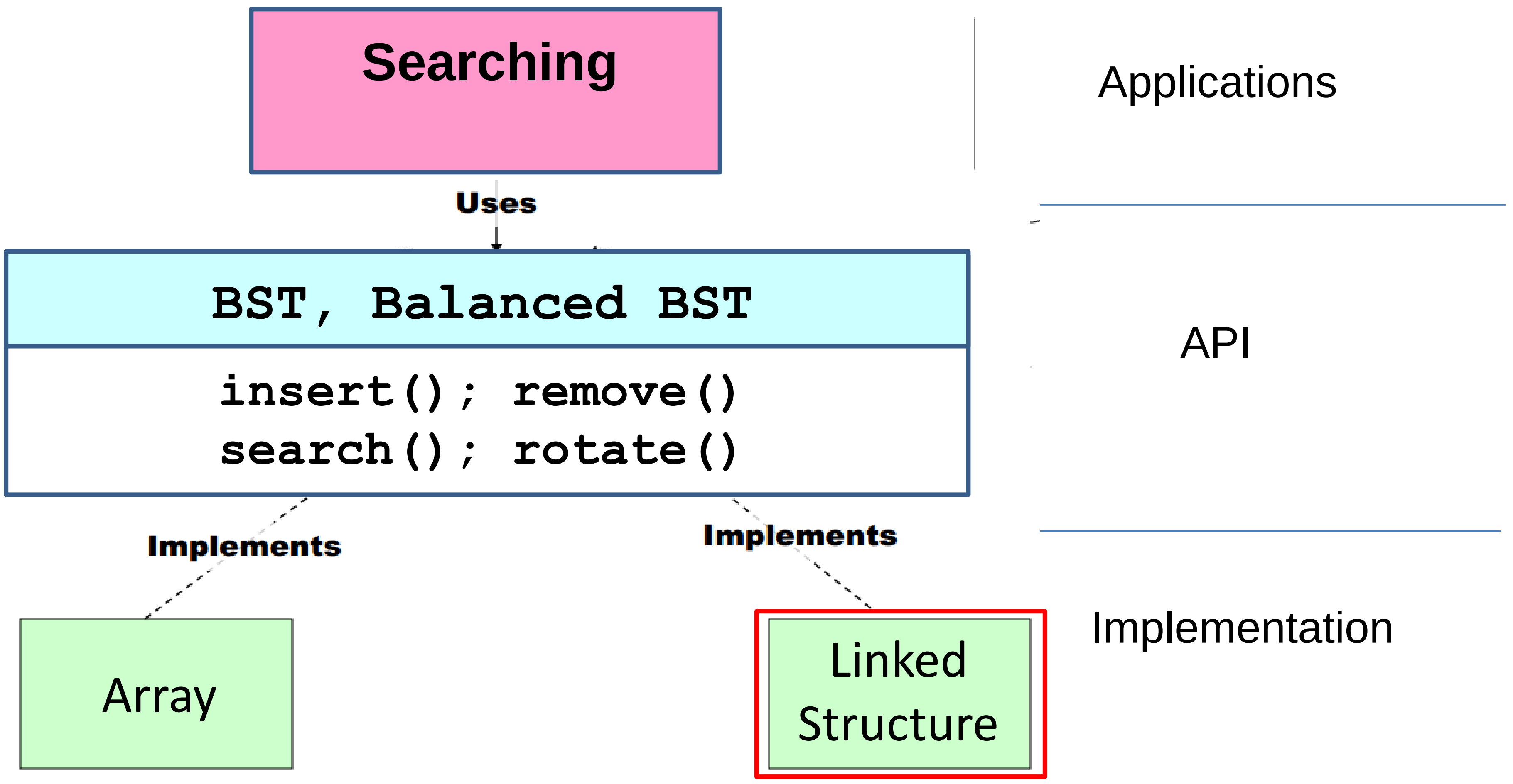
[M.Goodrich, sec. 11.6, p. 510]

Other Search Trees

- AVL Trees – Cây AVL [M.Goodrich, sec. 11.3]
- Red-Black Trees – Cây đỏ đen [M.Goodrich, sec. 11.6]
- ...

Self study to find out interesting things!

Summary


Searching
Applications
Uses
BST, Balanced BST
insert(); remove()
search(); rotate()
API
Implements
Implements
Array
Linked Structure
Implementation

Other ADTs

- Graph (đồ thị)
- Maps (ánh xạ, từ điển)
- Hash table (bảng băm)
- Set, Multisets and Multimaps (tập hợp)
- Text Processing and Prefix/Suffix Tree

Study about a new data structure

- The need for a new data structure (ADT)
- Definition
- Specification
- Implementation
  - Storing data
  - Algorithm
- Application