Binary Search Tree

Speaker 邱聖元

Outline

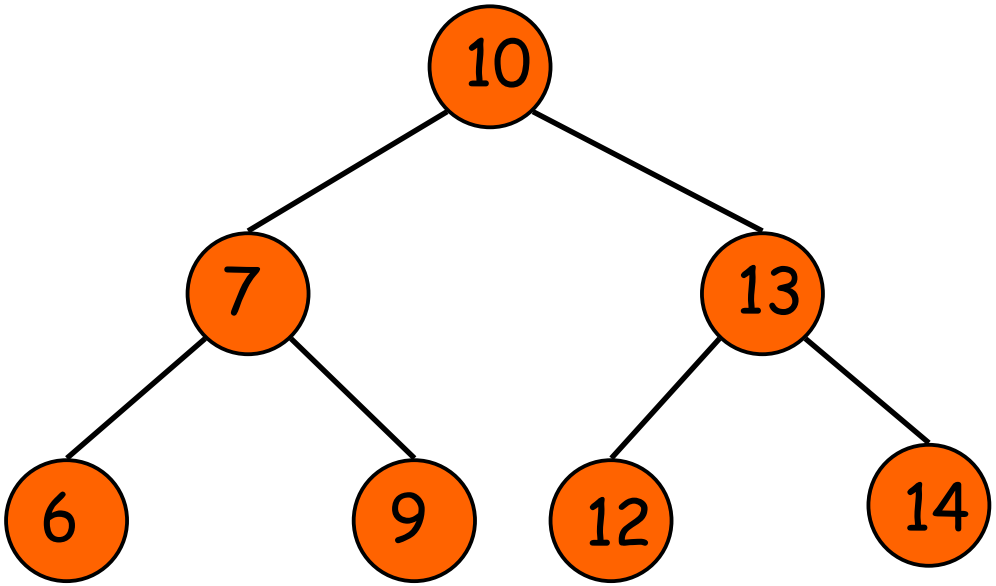
- Introduction
- Operations
- Conclusion

Introduction

- What's a binary search tree?
  - It's a binary tree !
  - For each node in a BST, the left subtree is smaller than it; and the right subtree is greater than it.

Introduction

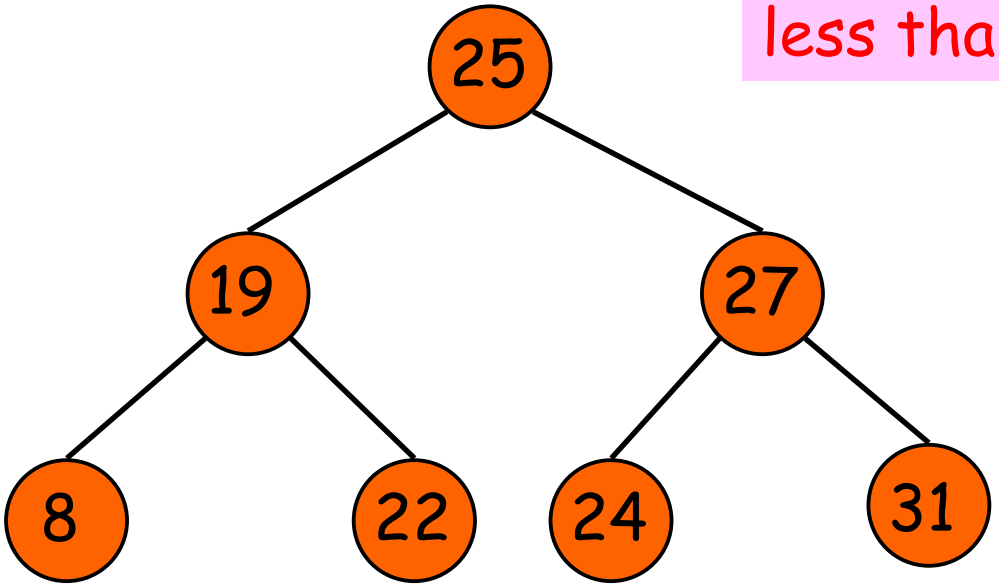
- This is a simple BST



Introduction

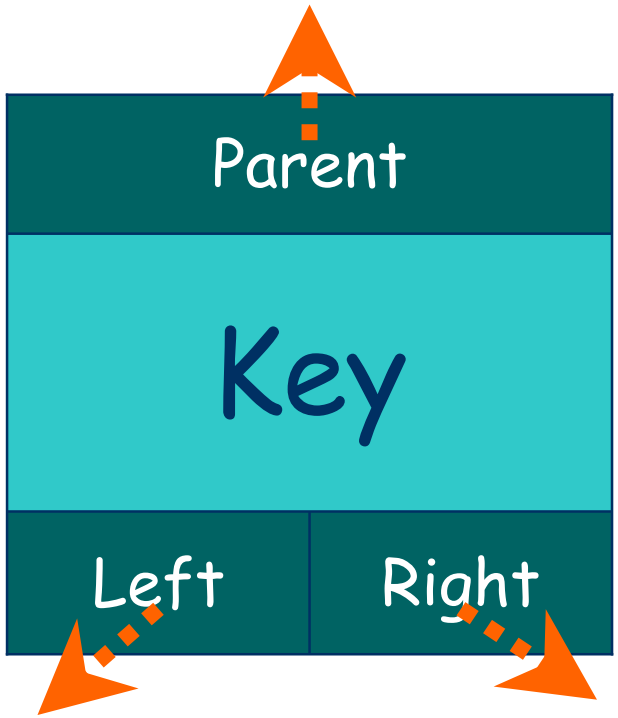
- Is this a BST ?

NO! Because 24 is less than 25



Introduction

- Node structure



Operations

- There are 3 common operations
  - QUERY
  - INSERT
  - DELETE

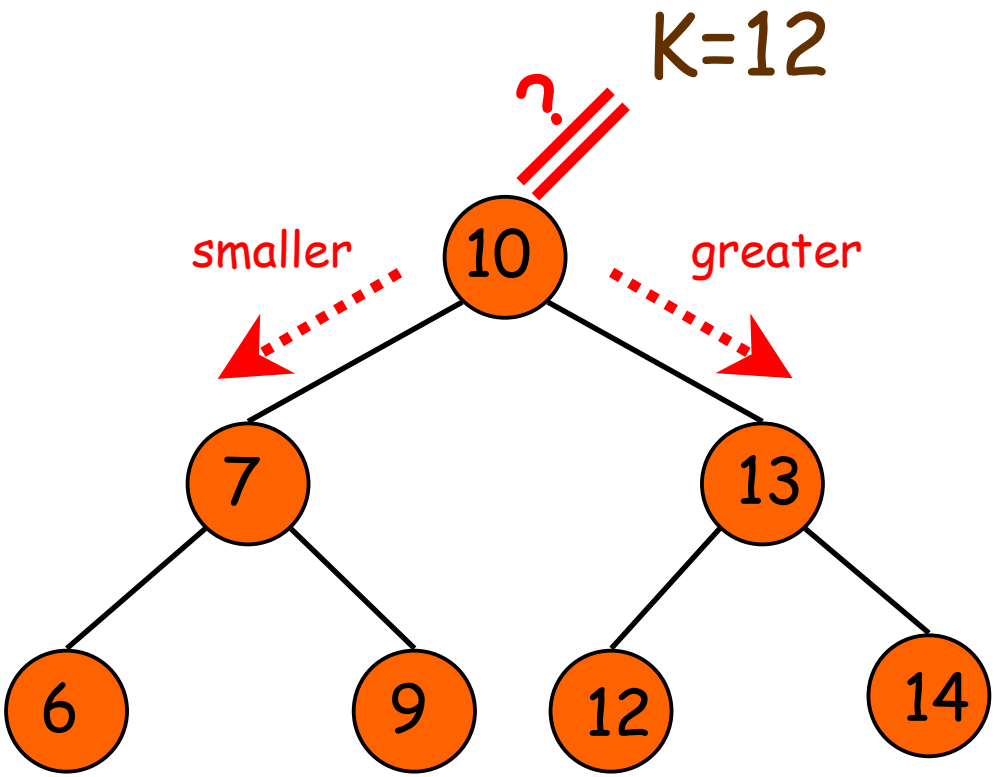
Operation - Query

- The QUERY operation can be further spit into
  - Search
  - Max/Min
  - Successor/Predecessor

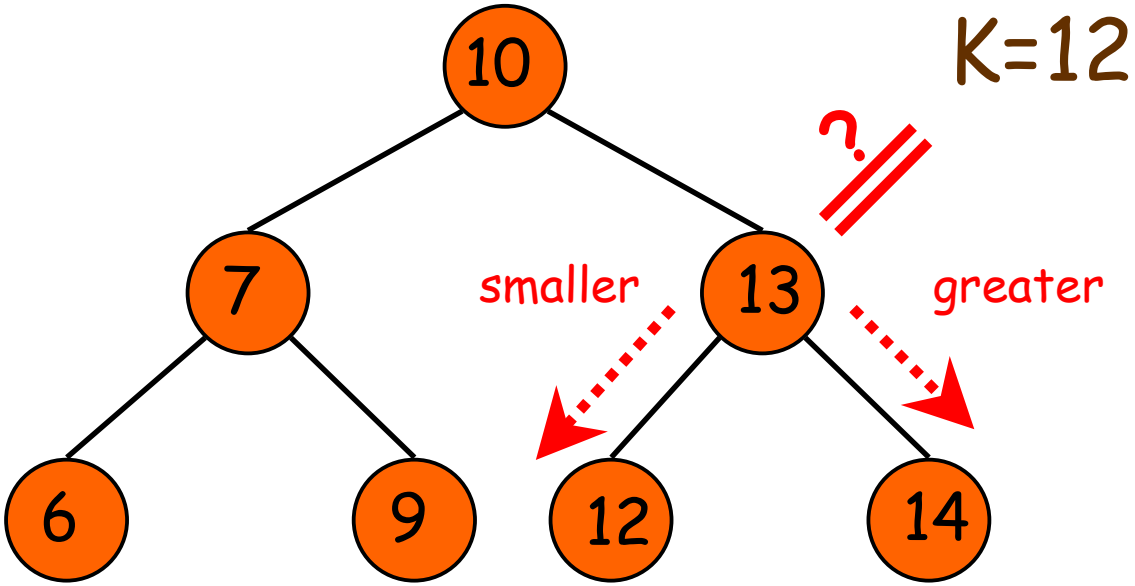
Operation - Search

- Search(T,k)
  - search the BST T for a value k

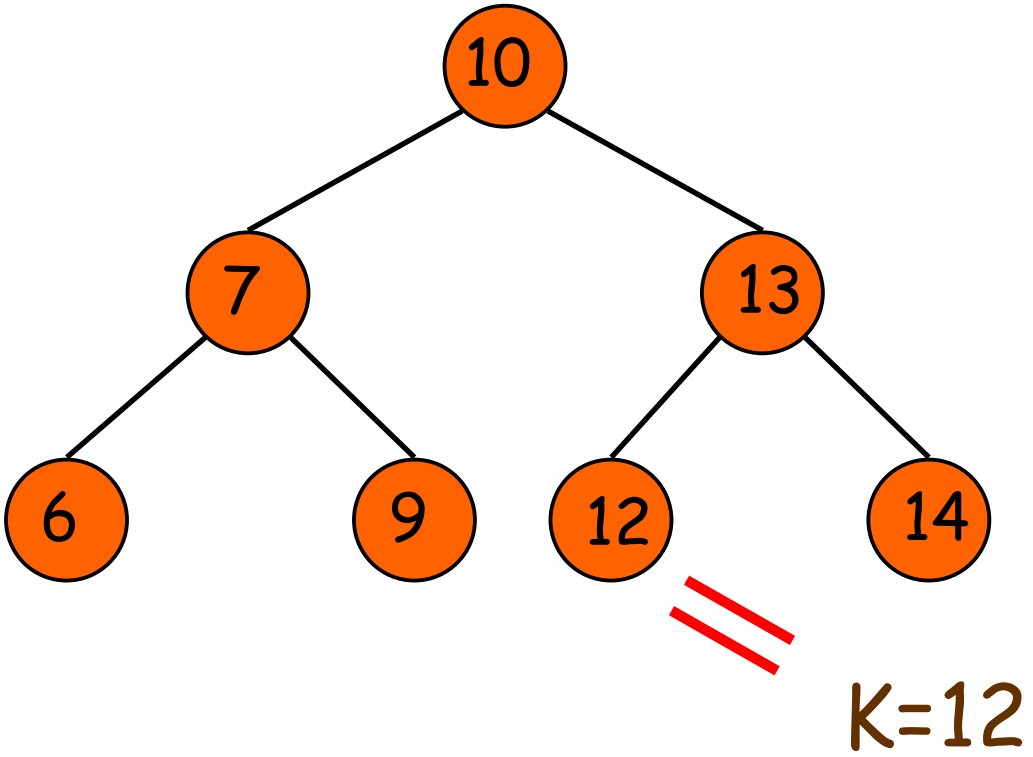
Operation - Search



Operation - Search



Operation - Search



Operation - Search

- Search operation takes time O(h), where h is the height of a BST

Operation – Min/Max

- For Min, we simply follow the left pointer until we find a null node
- Why? Because if it's not the minimum node, then the real min node must reside at some node's right subtree. By the property of BST, it's a contradiction
- Similar for Max
- Time complexity: O(h)

Operation Predecessor/Successor

- Successor(x)
  - If we sort all elements in a BST to a sequence, return the element just after x
  - Time complexity: O(h)

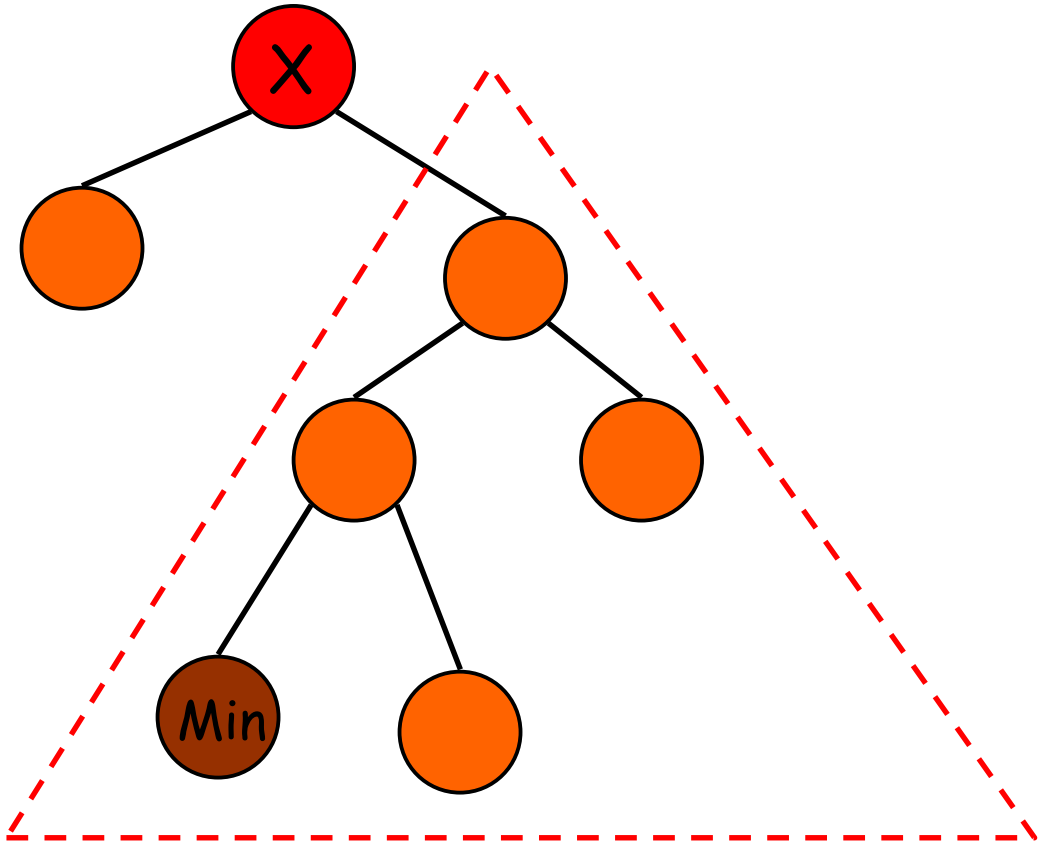
Operation Predecessor/Successor

- Find Successor

  if Right(x) exists, // Step 1
  then return Min(Right(x)) ;
  else // Step 2
  Find the first ancestor of x whose left subtree contains x ;

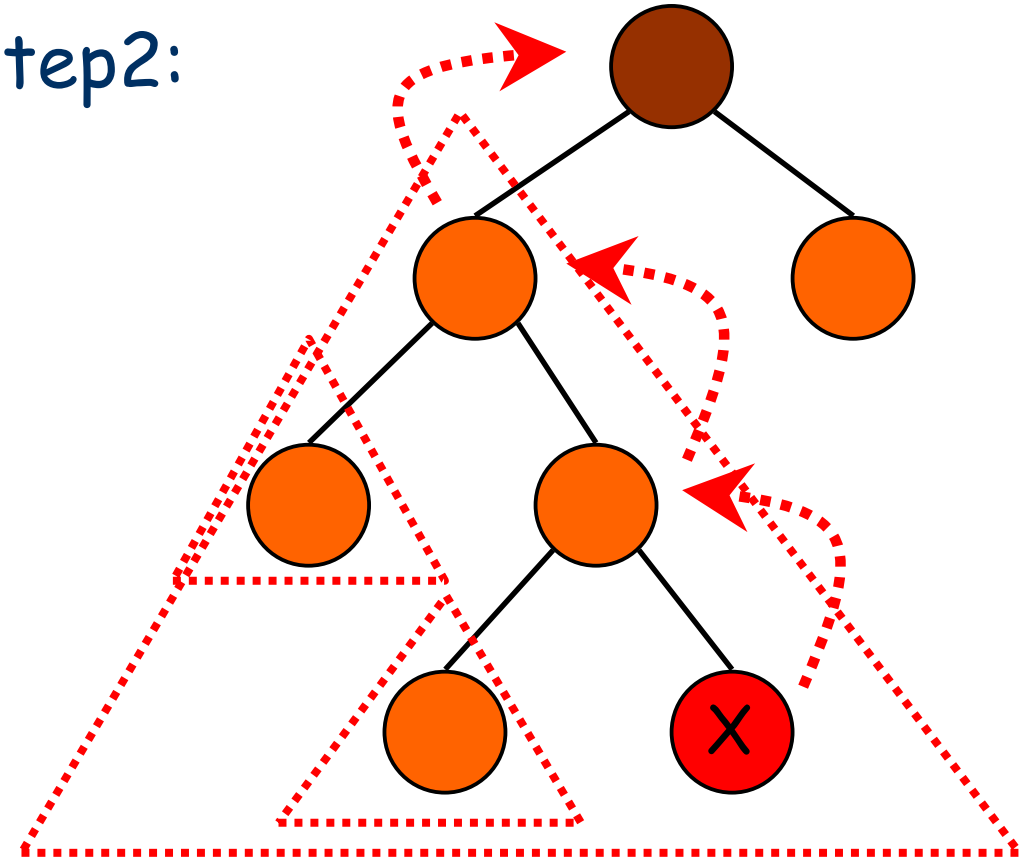
Operation Predecessor/Successor

Step1:



Operation Predecessor/Successor

Step2:



Finding the ancestor whose left subtree contains X

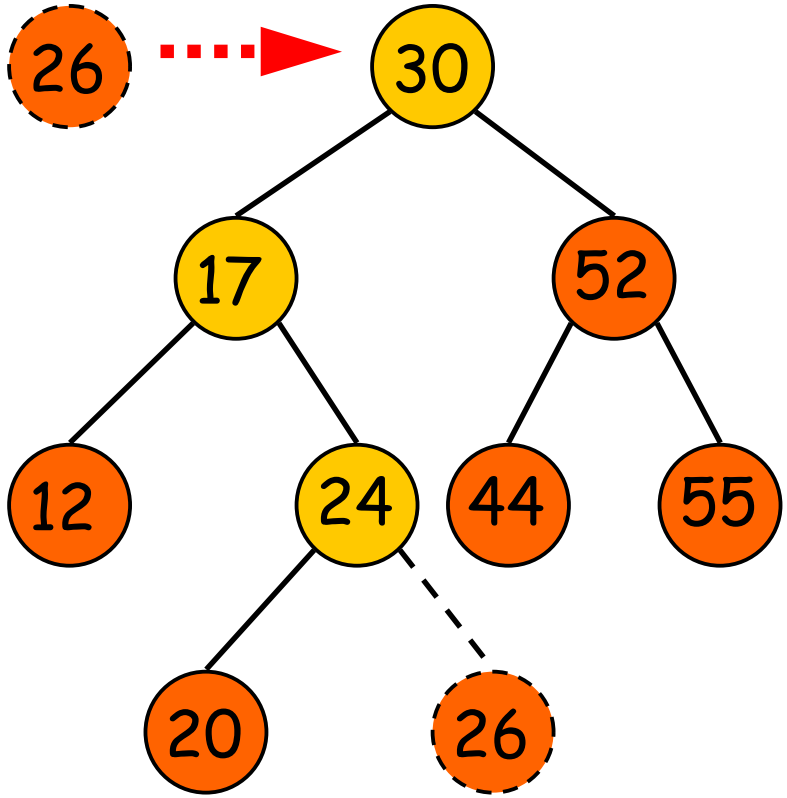
Operation - Insert

- Insert(T,z)
  - Insert a node with KEY=z into BST T
  - Time complexity: O(h)

Operation - Insert

- Step1: if the tree is empty,
  then Root(T)=z
- Step2: Pretending we are searching for z
  in BST T, until we meet a null node
- Step3: Insert z

Operation - Insert



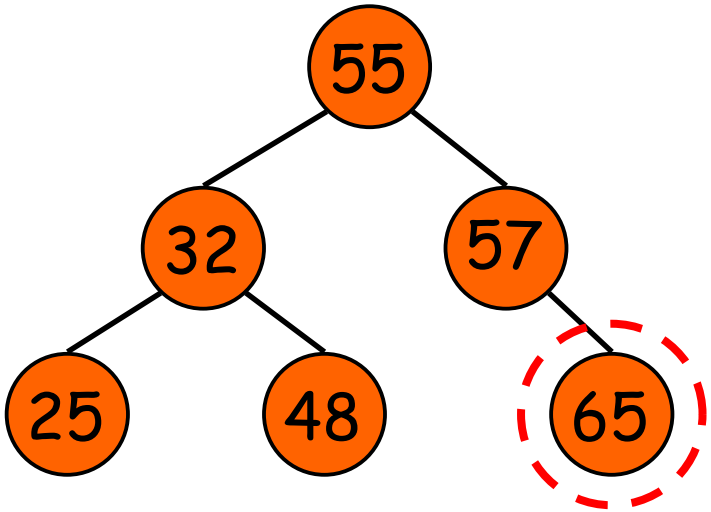
The light nodes are compared with k

Operation - Delete

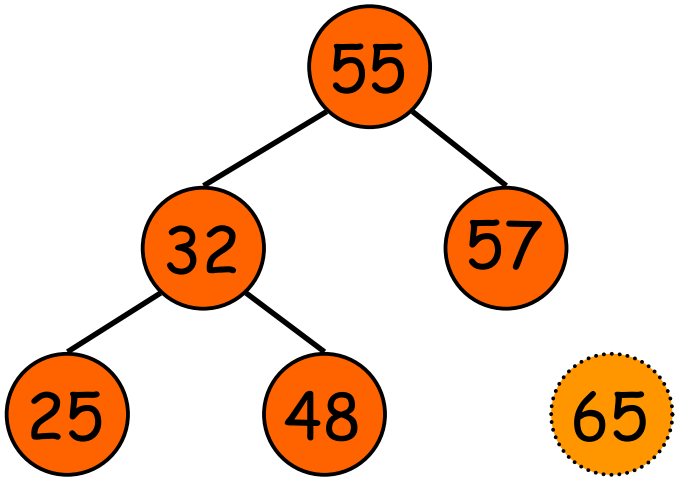
- Delete(T,z)
  - Delete a node with key=z from BST T
  - Time complexity: O(h)

Operation - Delete

- Case 1: z has no child



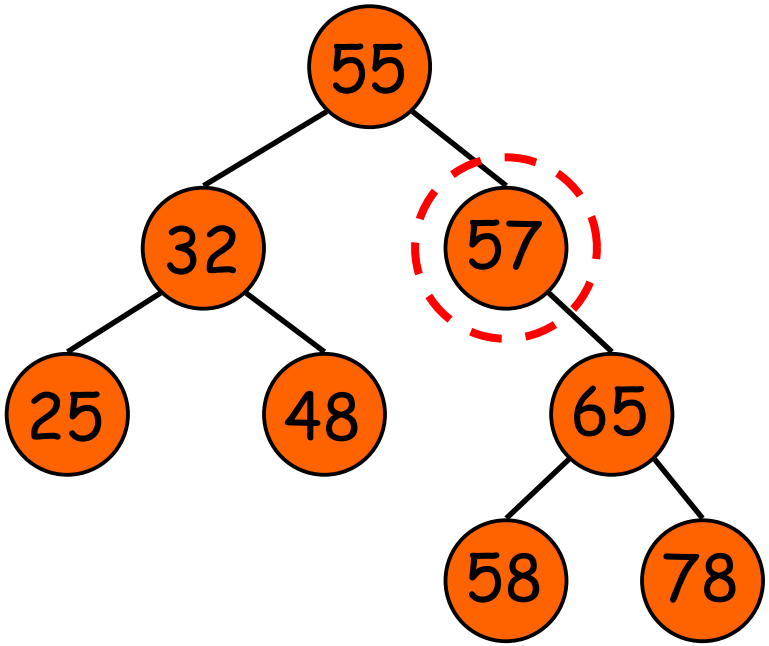
Operation - Delete



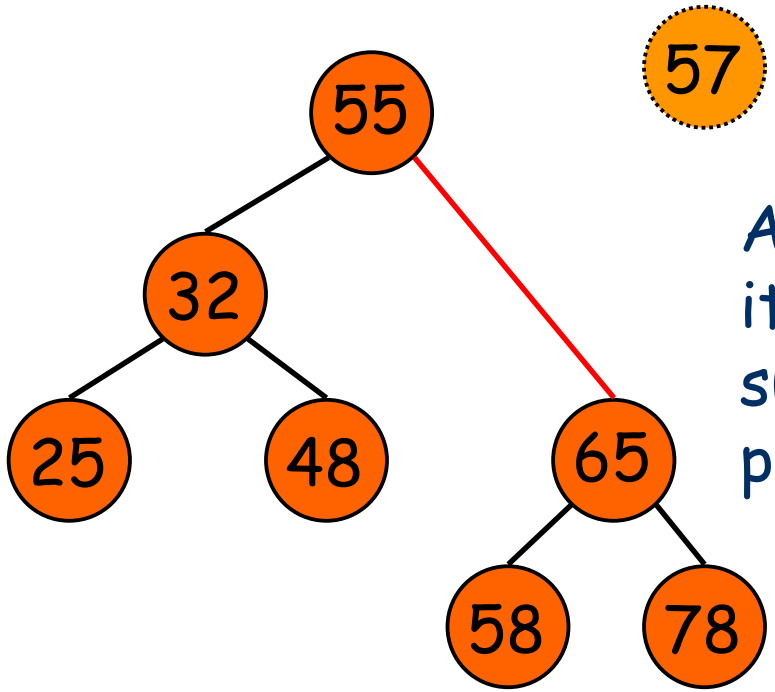
We can simply remove it from the tree

Operation - Delete

- Case 2: z has one child



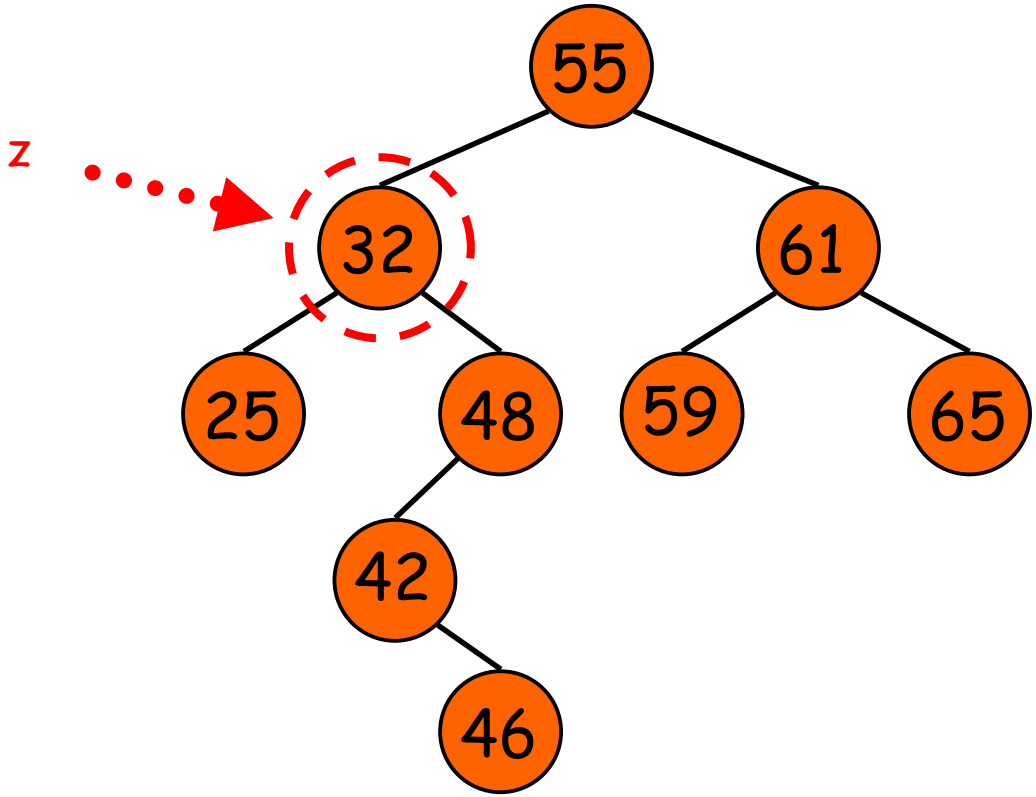
Operation - Delete



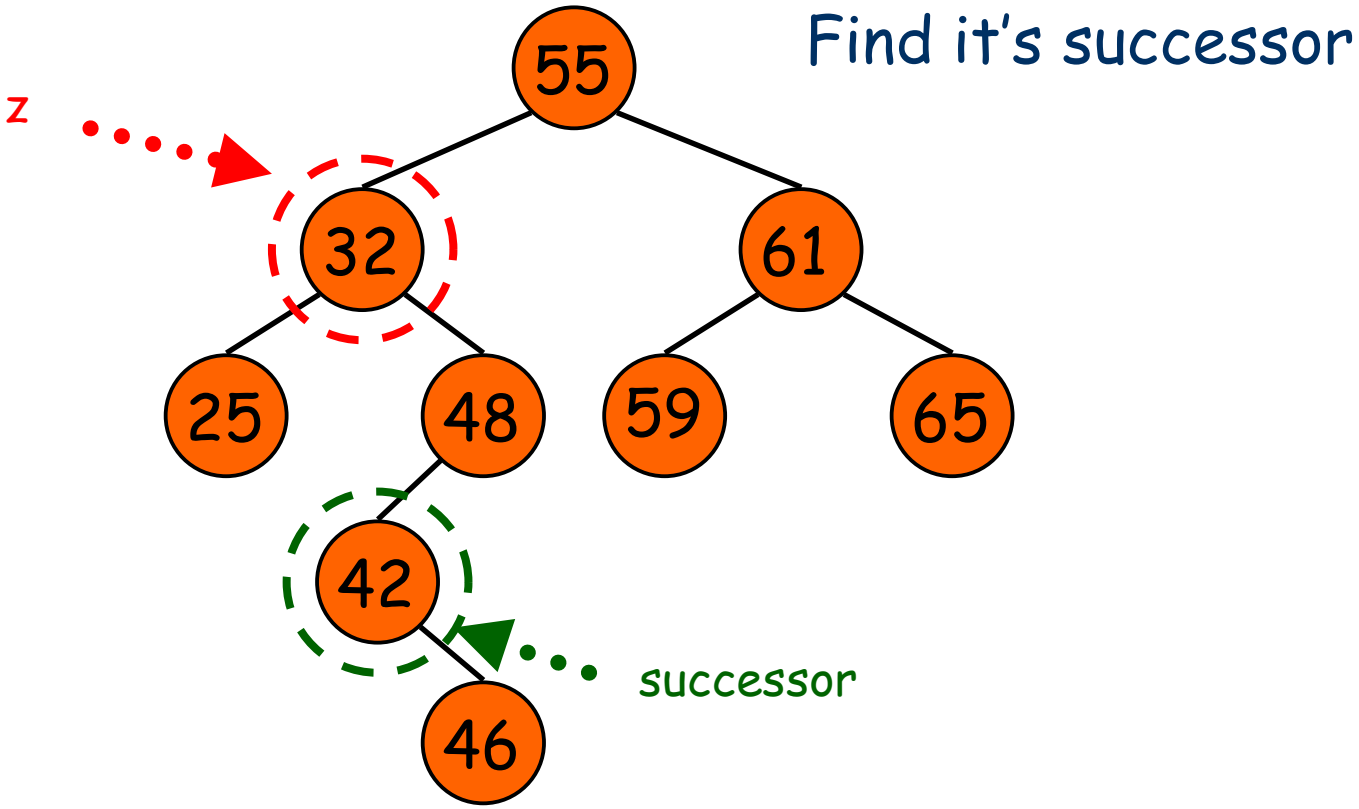
After removing it, connect it's subtree to it's parent node

Operation - Delete

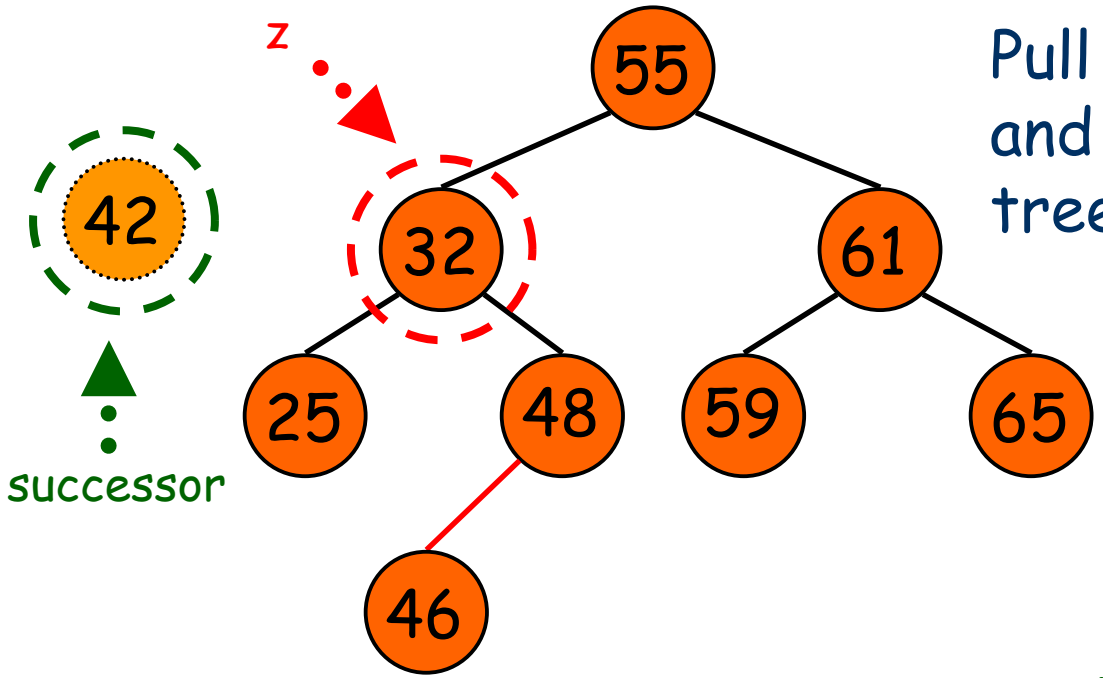
- Case 3: z has two child



Operation - Delete



Operation - Delete



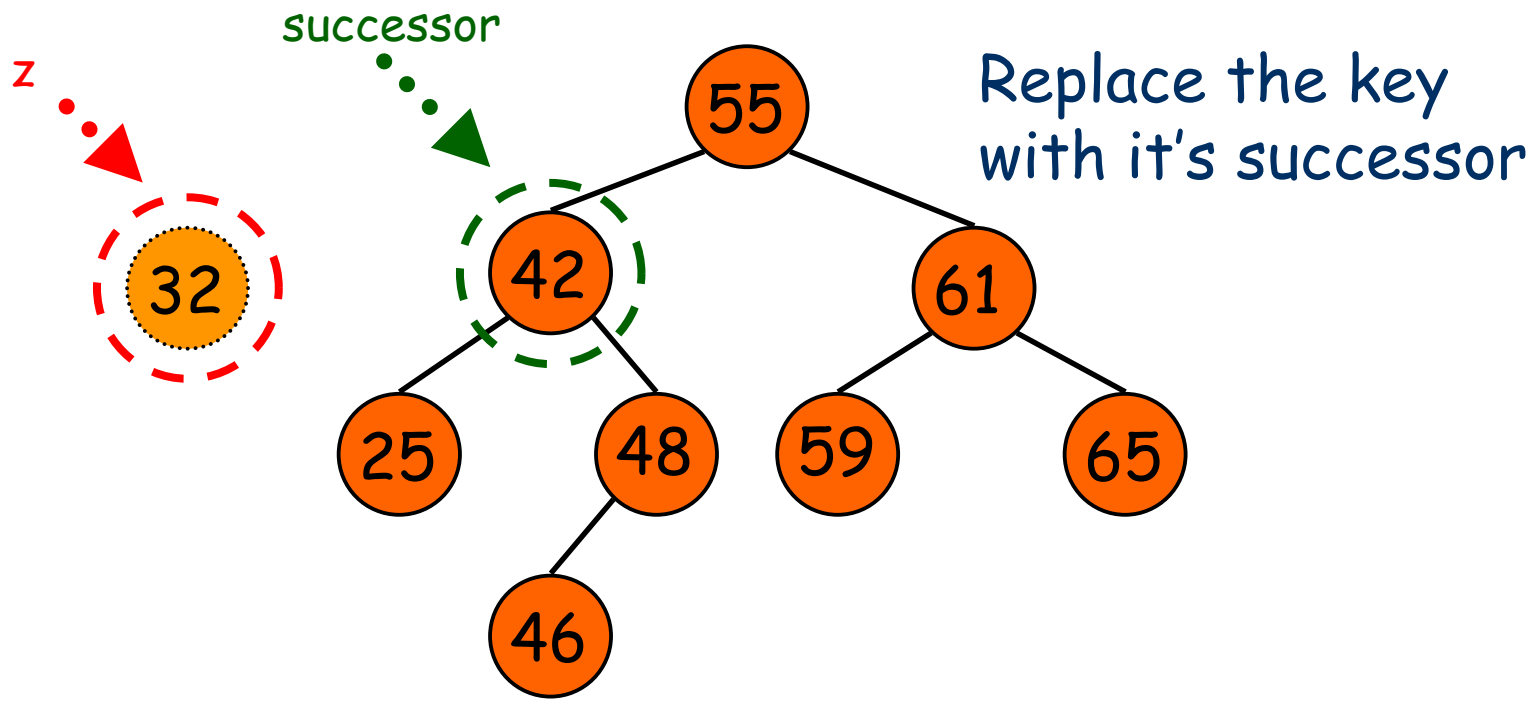
Pull out successor, and connect the tree with it's child

What if successor has two children?

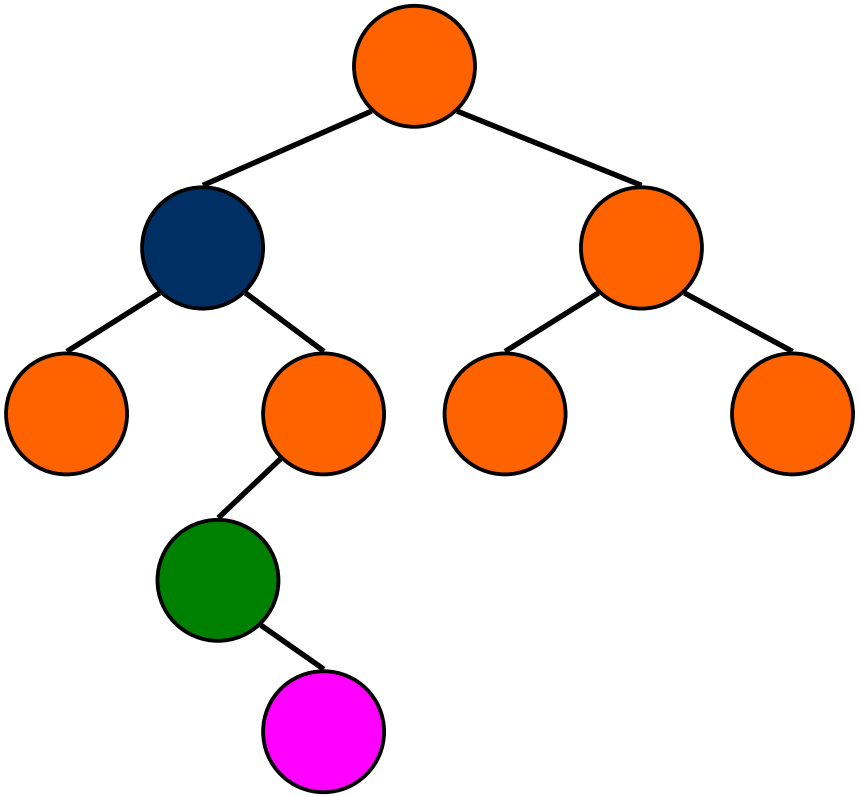
Operation - Delete

- What if the successor has two nodes?
- Not possible ! Because if it has two nodes, at least one of them is less than it, then in the process of finding successor, we won't pick it !

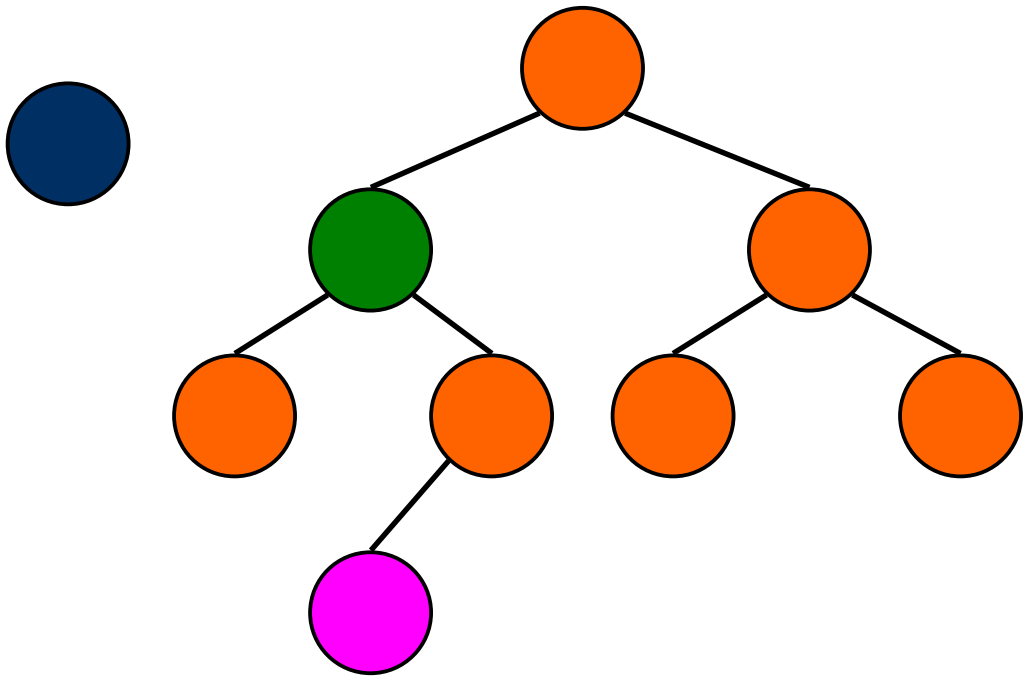
Operation - Delete



Operation - Delete



Operation - Delete



Operation - Delete

- The original sequence is:

  25 32 42 46 48 55 59 61 65

- After deleting 32

  25 42 46 48 55 59 61 65

Conclusion

- If we have a sorted sequence, and we want to design a data structure for it
- Array? Or BST?

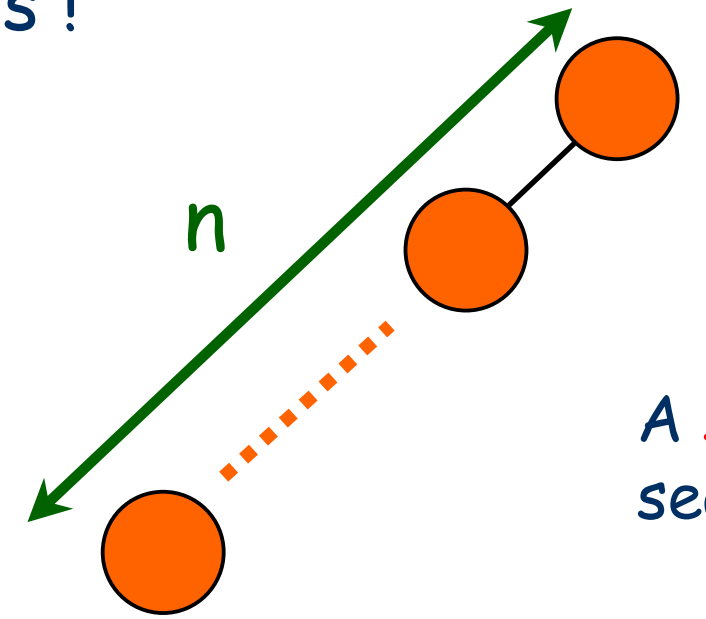
Conclusion

- The Search time:

BST	O(h)
Array	O(log n)

Conclusion

- We already know that n is fixed, but h differs from how we insert those elements !



A skew binary search tree

Conclusion

- So why we still need BST?

- Easier insertion/deletion
- And with some optimization, we can avoid the worst case !