Lecture 14
Dictionaries
Ordered Maps

What is the difference between a map and a dictionary?

Map Interface

- search(k)
- insert(k,v)
- delete(k)
- delete(p)

Additional Dictionary Operation findAll(k)

Implement Dictionary Operations

insert(k,v)

findAll()

What will be the order of returned elements?

Time Complexity?

delete(k)

How do you delete specific element?

A MAP can be easily extended to a Dictionary!

MAP does not need the keys to have total order!

In many cases, keys do have total order!

Order related functions

1. min/max
2. predecessor
3. successor

More Ordered Functions

- ceilingEntry(k)
- floorEntry(k)
- lowerEntry(k)
- higherEntry(k)

Complexity with Hash Table?

1. min/max
2. predecessor
3. successor

Additional limitations of Hash Table:

-large array
-worst case
-hash cost

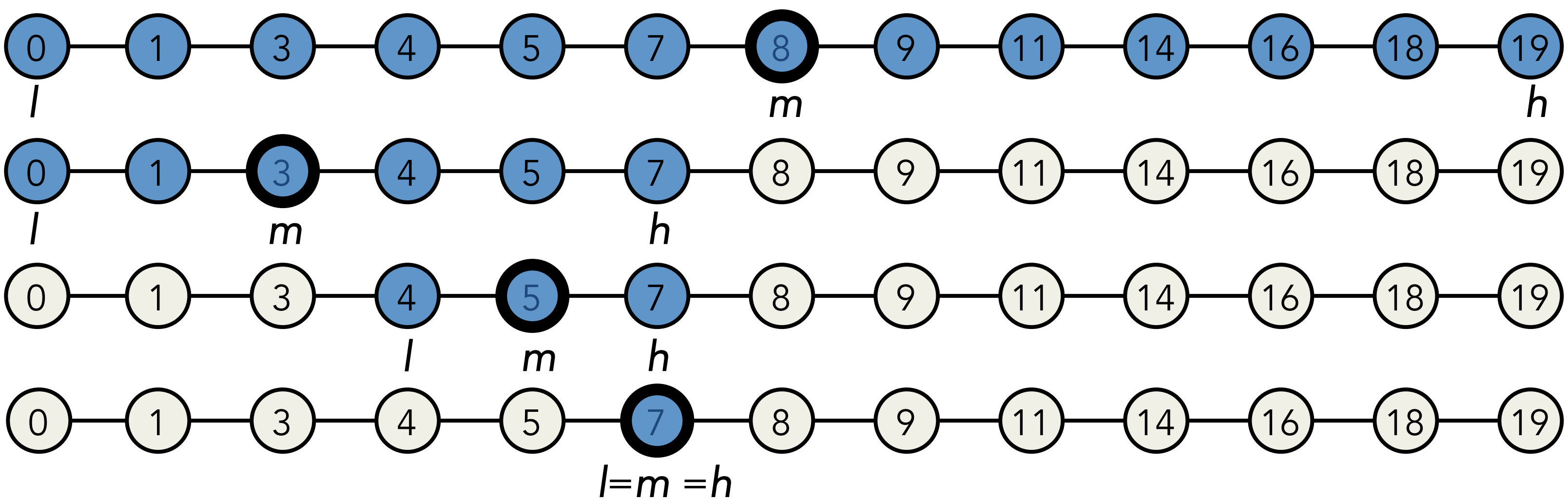
Keep the elements in some order!

Ordered Maps

Idea: keep keys in a sorted array!

Binary Search
O(log n)

-find(7)



Time complexity

insert – $O(n)$
remove – $O(n)$
search – $O(\log n)$

How do we improve insert/remove time?

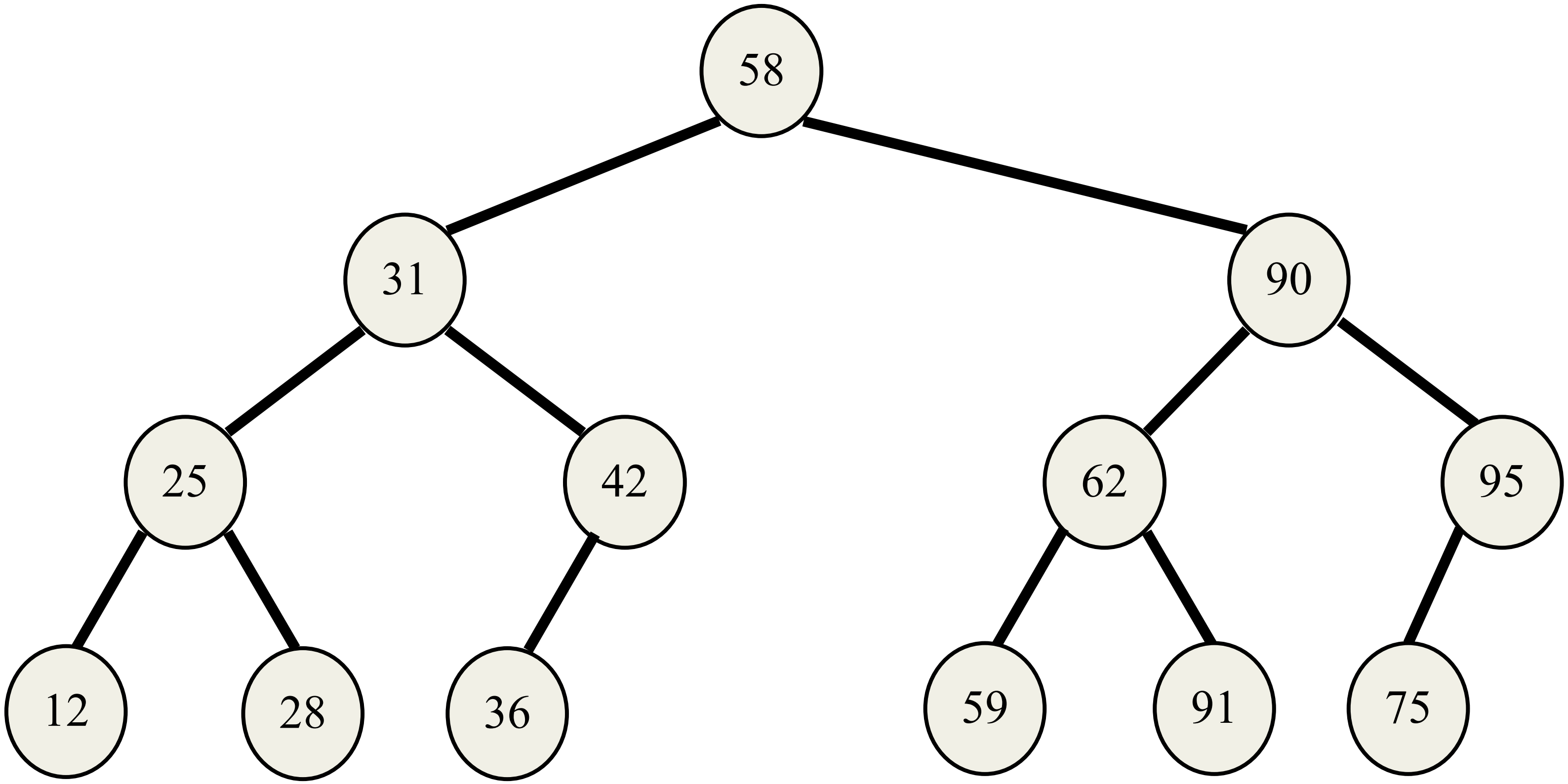
Trees

Binary Search Tree

-left subtree smaller than node

-right subtree larger than node

Example



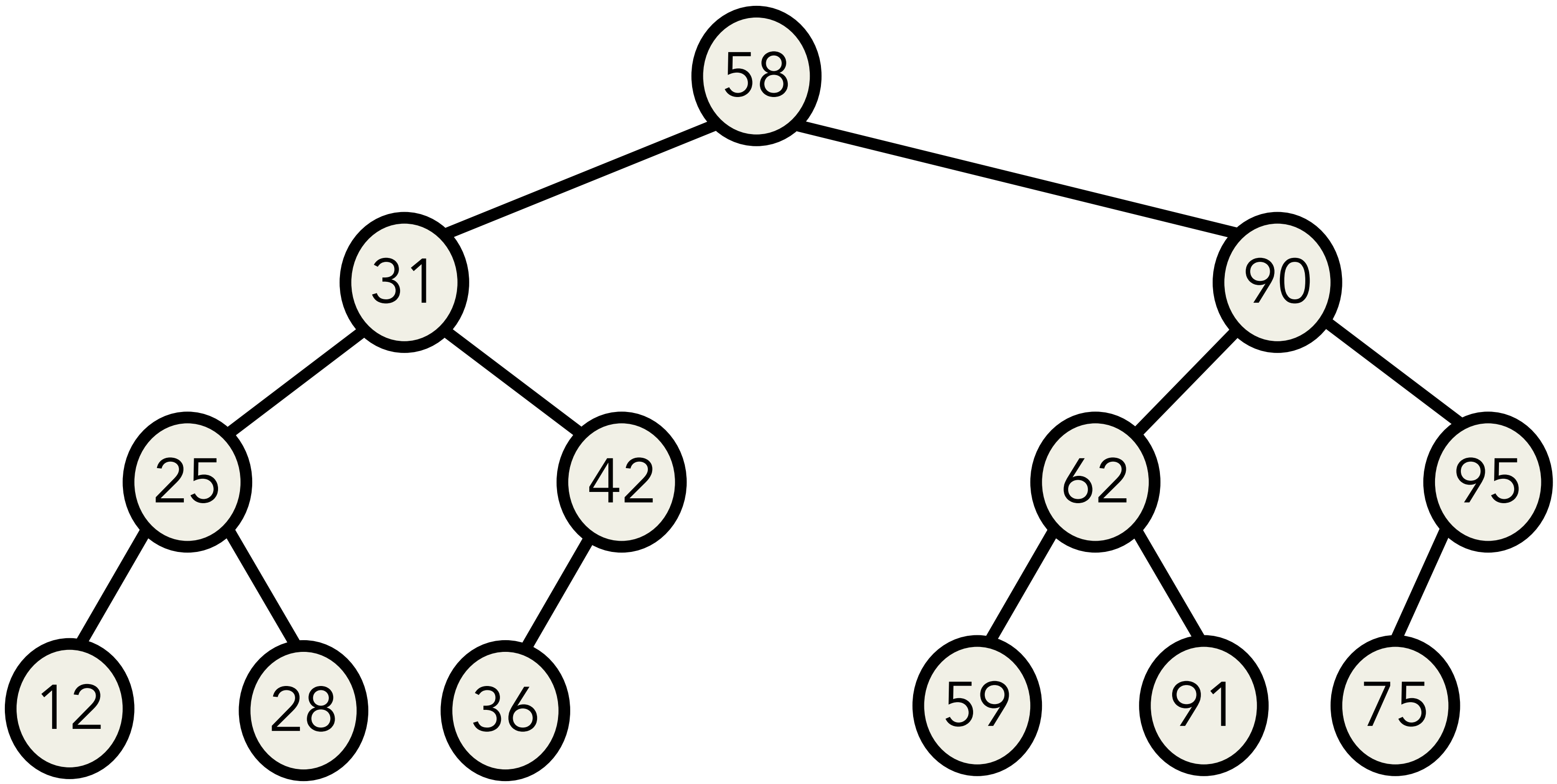
Operations

1. search
2. min/max
3. predecessor/successor
4. insert
5. delete

search

```
Tree-Search(x, k)
    while x ≠ NIL and k ≠ x.key
        if k < x.key
            x = x.left
        else
            x = x.right
    return x
```

min/max



min/max

```
Tree-Min(x)
    while x.left ≠ NIL
        x = x.left
    return x
```

```
Tree-Max(x)
    while x.right ≠ NIL
        x = x.right
    return x
```

successor

```
Tree-Successor(x)
    if x.right ≠ NILL
        return Tree-Min(x.right)
    y=x.p
    while y ≠ NIL and x == y.right
        x=y
        y=x.p
    return y
```

predecessor?

In-order traversal, successor, and predecessor!

In-order can be though of as a projection on a horizontal line!

insert

1. find place where z belongs
2. insert z there

insert

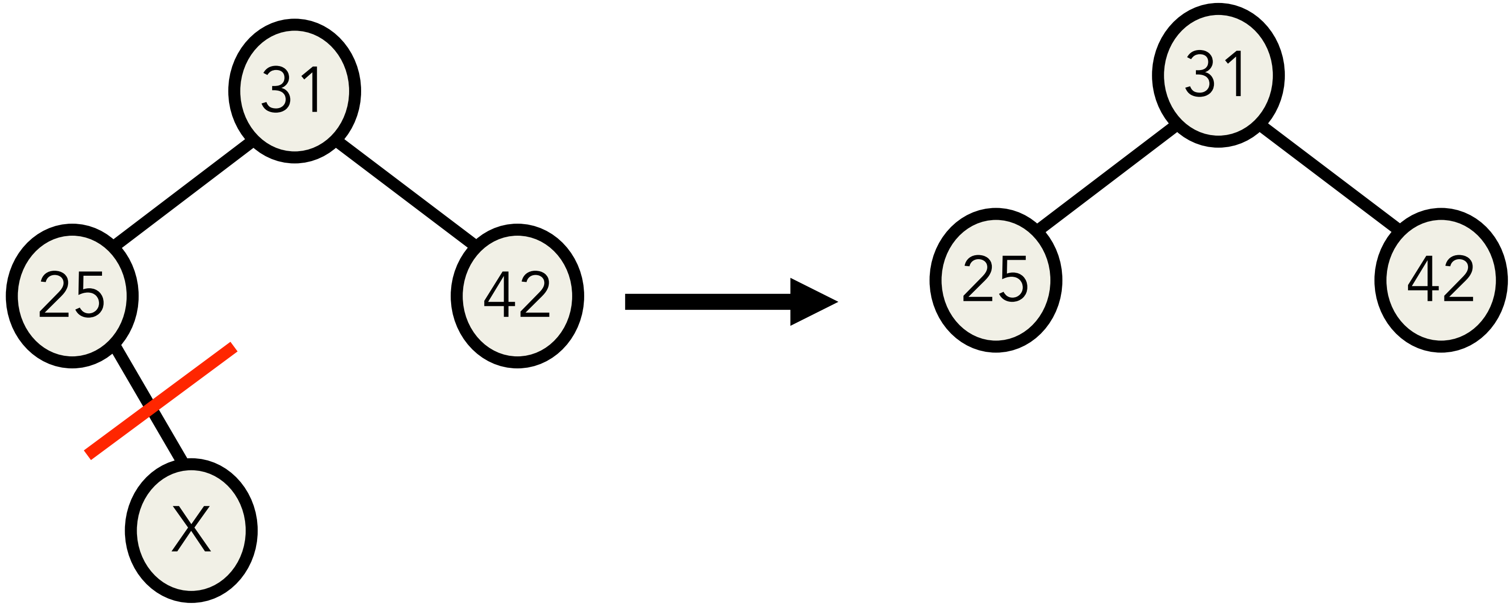
```
Tree-Insert(T,z)
    y = NIL
    x = T.root
    while x ≠ NIL
        y=x
        if z.key < x.key
            x = x.left
        else
            x = x.right
    z.p = y
        if z.key < y.key
            y.left = z
        else
            y.right = z
```

delete(x)

1. x has no children
2. x has one children
3. x has two children

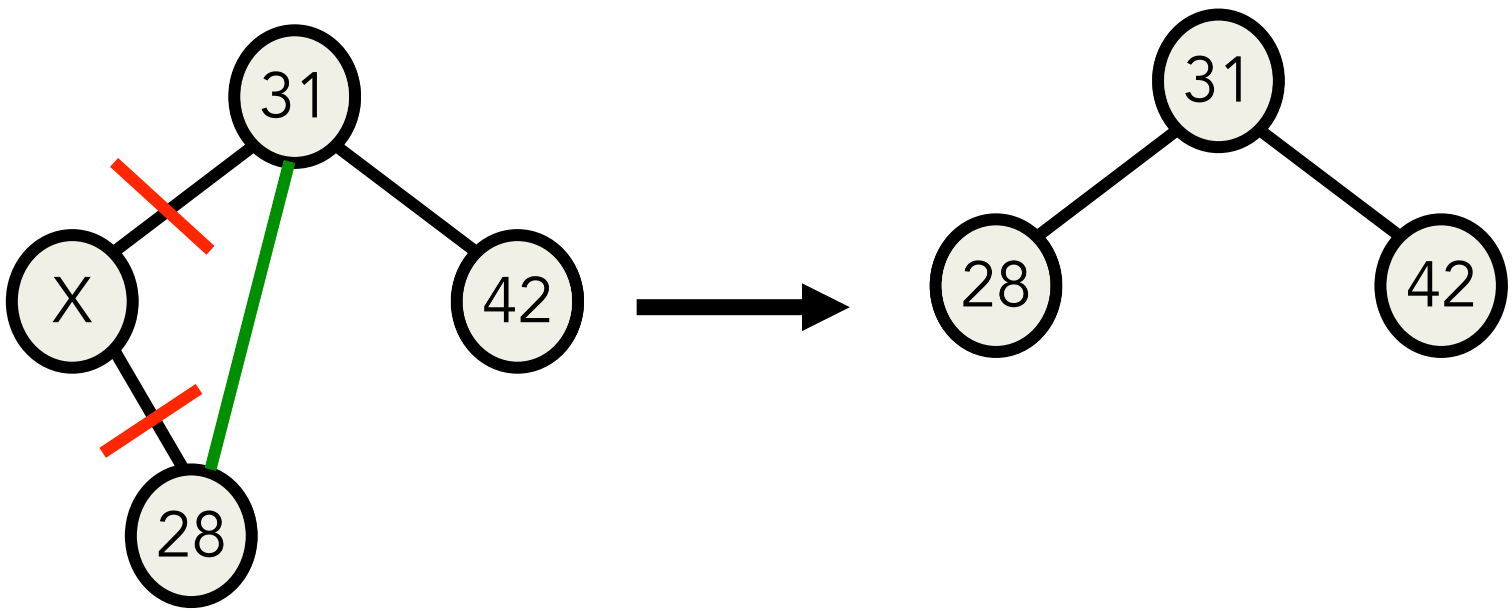
Deletion case 1

- If x has no children, just remove x



Deletion case 2

- If x has exactly one child, just remove x and make x.p point to that child



Deletion case 3

- If x has two children, then to delete
  1. find its successor (or predecessor) y
  2. remove y
  3. replace x with y

Given a set of n numbers, how much time does it take to create a BST of n numbers!

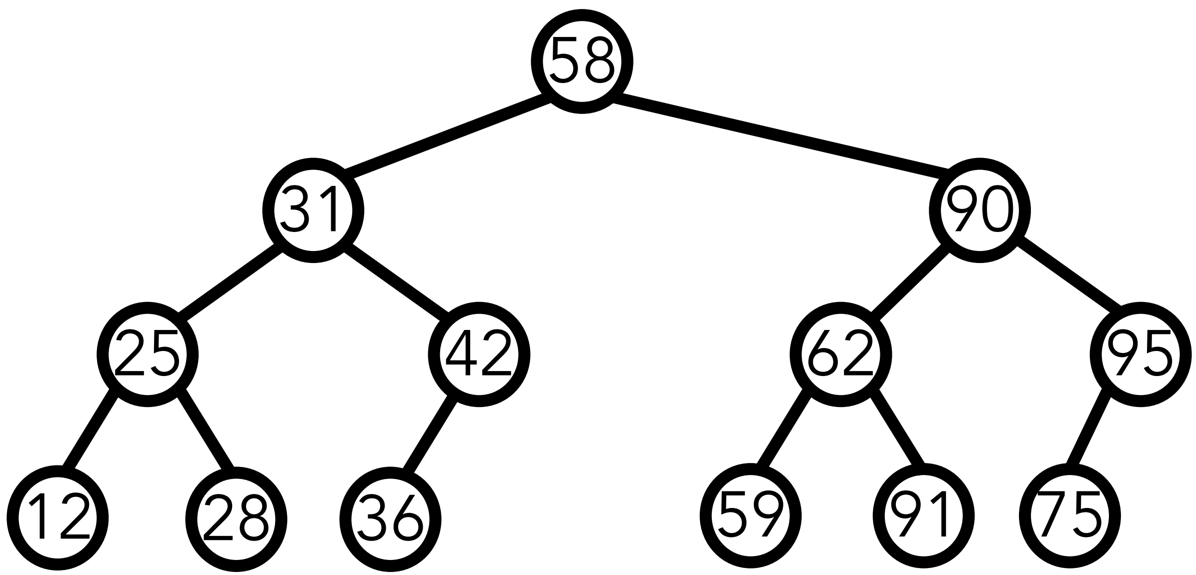
Create a BST of n sorted numbers!

Time complexity?

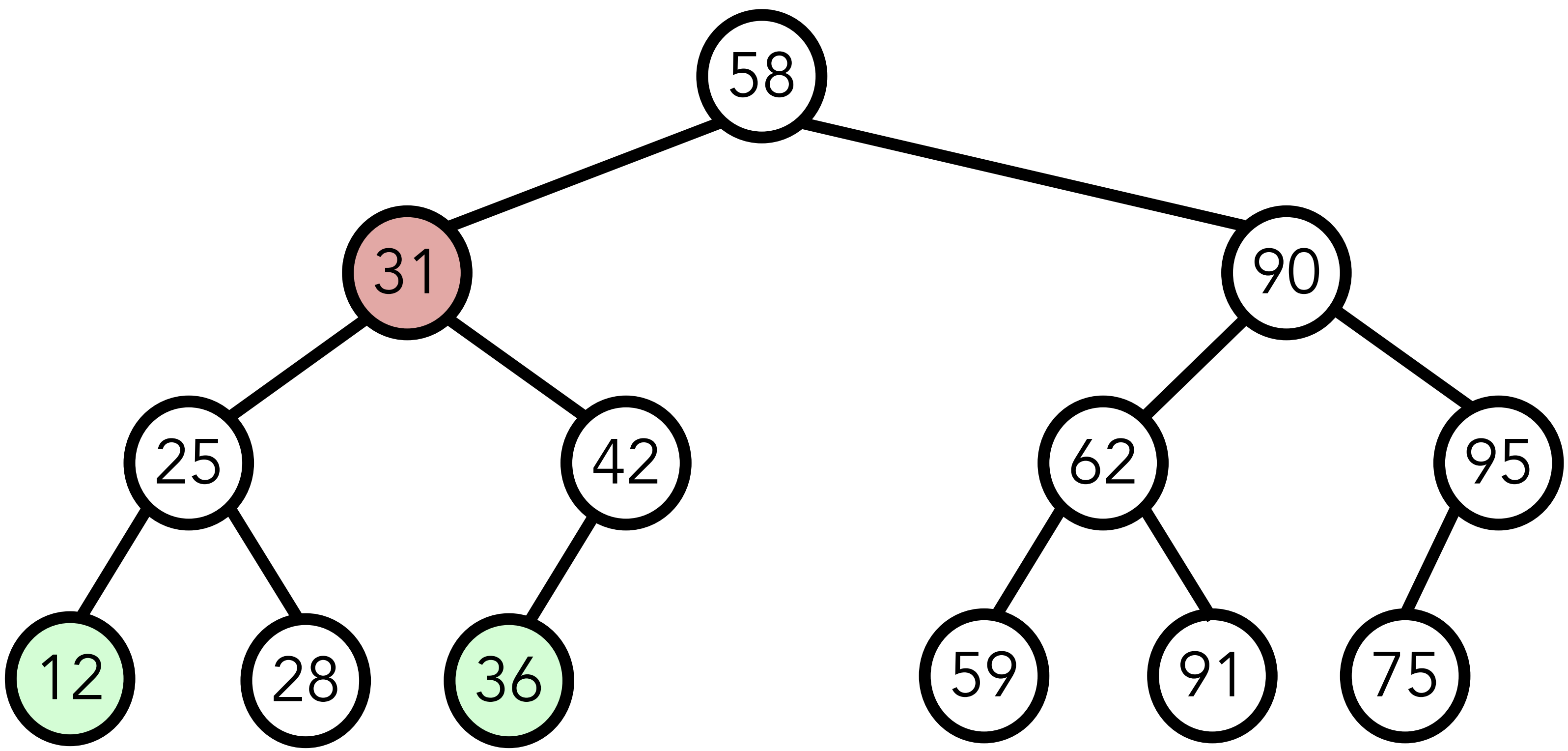
Given two sorted arrays (distinct numbers) of size n each, find pairs whose some is equal to N.

A1 = (1, 2, 3)
A2 = (4, 5, 6)
N = 6
Output: (1,5) (2,4)

Suppose a tree represents road network, where edge is the road and nodes are the cities, find shortest distance between two cities!



Given n_1 and n_2, find lowest common ancestor of n_1 and n_2 in a BST



Bottom-up Solution

1. Search for n_1 – log n
2. Search for n_2 – log n
3. Compare node n_1 with node n_2, if they are the same, done
4. Include nodes alternatively from L_1 and L_2 until a match is found

Bottom-up Solution

1. Search for n_1 – log n, store nodes in a list L1
2. Search for n_2 – log n, store nodes in a list L2
3. Start from deepest node in L1,
4. Compare with all nodes in L2
   1. If matched, LCA is found so abort
5. Else go up in L2 and repeat 3

Top-down Approach

1. Search for n_1 – log n, store nodes in a list L1
2. Search for n_2 – log n, store nodes in a list L2

- Start Matching corresponding elements in L1 and L2, the first mismatch is LCA

Top-down Solution

- Start from root and keep going down until
  - The node n is equal to n_1 or n_2, that node is LCA
  - The node n is greater than n_1 and less than n_2
  - If n_1 and n_2 are less than n, go left, otherwise go right