Augmented Treap

Background

An augmented data structure (like an augmented treap or an interval tree) is built upon an underlying data structure. New information (data) is added which:

- Is easily maintained and modified by the existing operations of the underlying data structure and
- Is used to efficiently support new operations.

Process of augmenting a data structure*

1. Choose an underlying data structure
2. Determine the additional information (data) to maintain in the underlying data structure
3. Verify that the additional information can be easily maintained by the existing operations (like Insert and Remove)
4. Develop new operations

* Section 14.2 of CLRS

Rationale for an augmented treap

- Suppose we wish to add two methods to the treap: one that returns the item of a given rank and the other that returns the rank of a given item

```
public T Rank(int i) { ... }
public int Rank(T item) { ... }
```

- The rank of an item is defined as its position in the linear order of items

Using the original treap

- To find the item of a given rank or to find the rank of a given item in the original treap requires us to traverse the underlying binary search tree in order. Why?

- The expected time complexity is O(n) since half the items are visited on average

Using the augmented treap

- By adding one data member to the original implementation of the treap, the expected time complexity for the two Rank methods can be reduced to O(log n)

- But what data member is added, where is it added, and can it be easily maintained?

New data member: NumItems

- The Node class is augmented with the data member NumItems which stores the size of the treap rooted at each instance of Node

- Calculated as

```
    p.NumItems = 1;
    if (p.Left != null)
        p.NumItems += p.Left.NumItems;
    if (p.Right != null)
        p.NumItems += p.Right.NumItems;
```

Augmented data structure

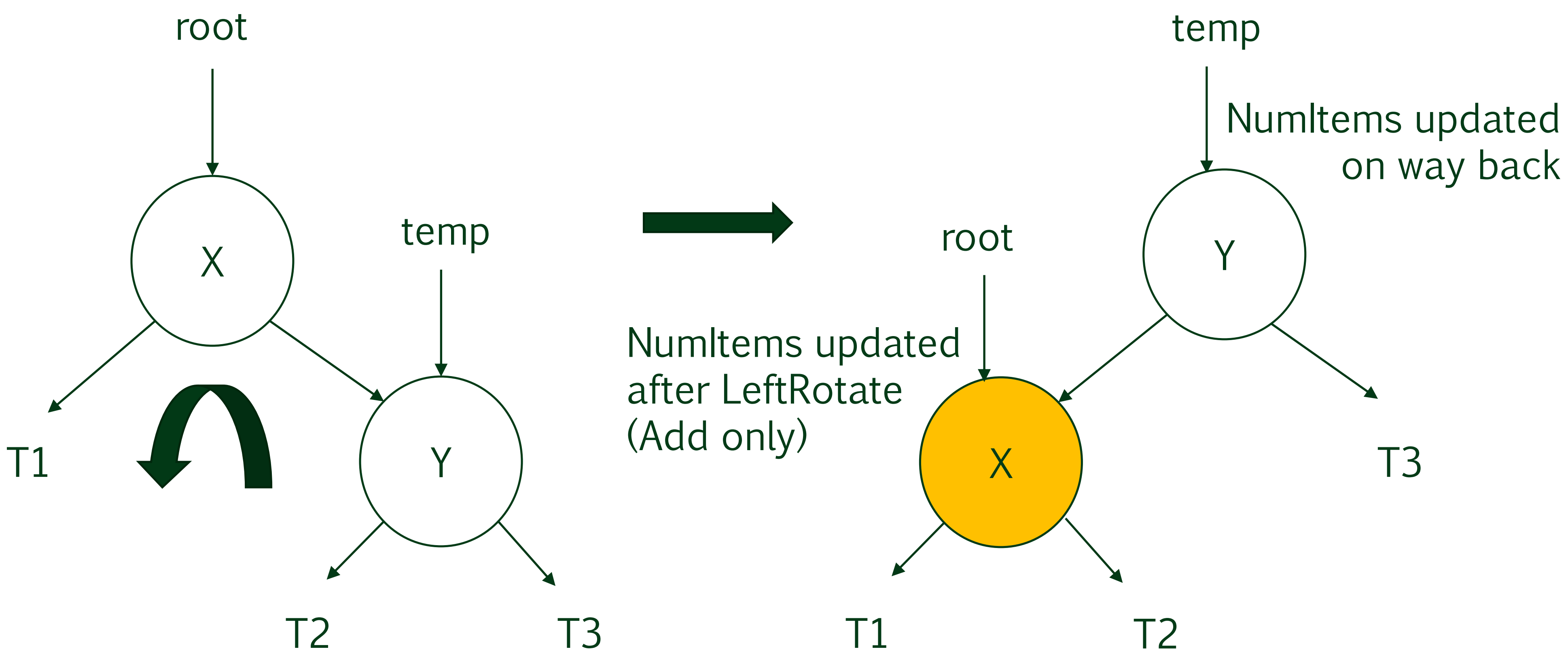
```
public class Node<T> where T : IComparable
{
    private static Random R = new Random();
    public T Item         { get; set; }
    public int Priority   { get; set; }      // Randomly generated
    public int NumItems   { get; set; }      // Augmented information (data)
    public Node<T> Left   { get; set; }
    public Node<T> Right  { get; set; }
    ...
}

class Treap<T> : ISearchable<T> where T : IComparable
{
    private Node<T> Root;  // Reference to the root of the Treap
    ...
}
```

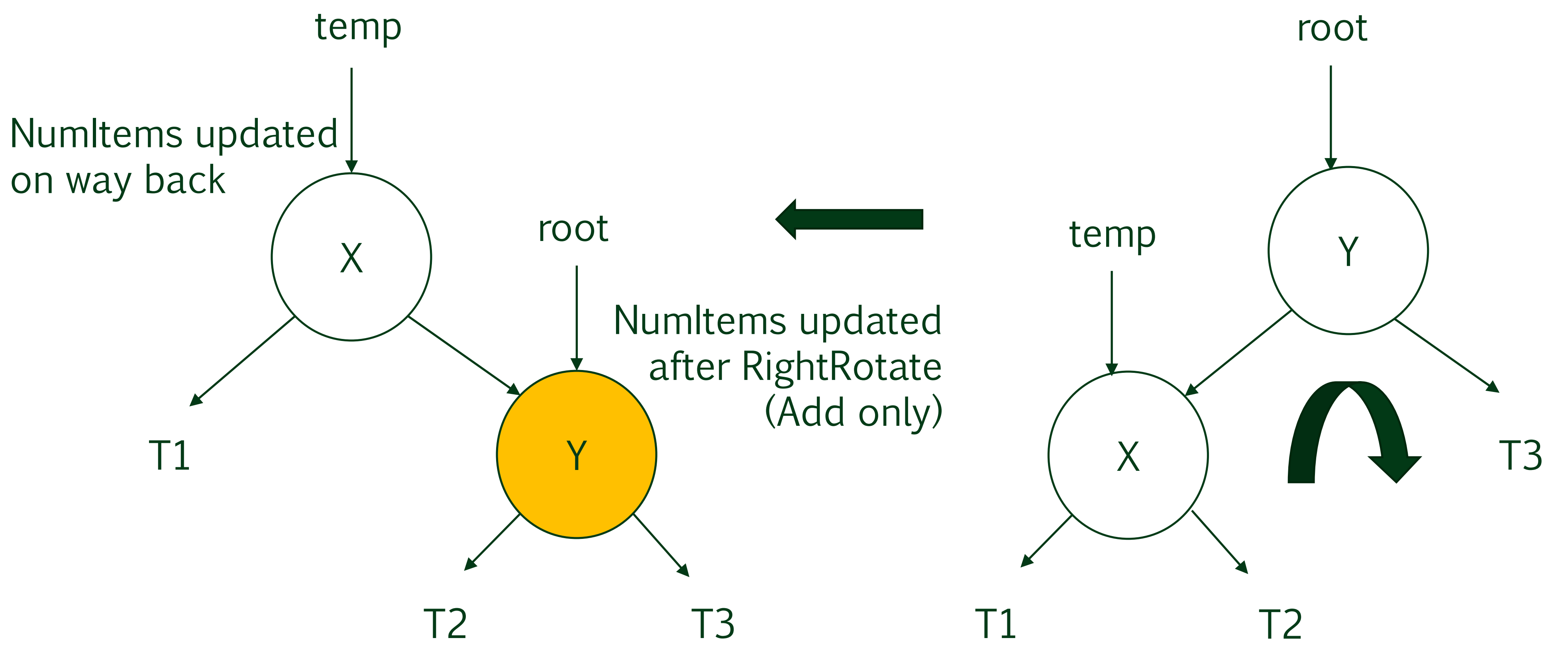
Can NumItems be easily maintained?

- Yes
- When an item is added to or removed from an augmented treap, NumItems is updated in O(1) time for each node along the path from the point of insertion or removal to the root of the treap. Note that both an insertion and removal take place at a leaf node.
- An additional update is also needed whenever a rotation is performed on the way up the treap for the Add method.

LeftRotate


root
X
temp
Y
T1
T2
T3
temp
NumItems updated
on way back
root
Y
NumItems updated
after LeftRotate
(Add only)
X
T3
T1
T2

RightRotate



Exercises

- Explain why NumItems is updated for the left (right) child of the new root after a LeftRotate (RightRotate) is performed in the Add method.
- Explain why NumItems need not be updated after a LeftRotate or RightRotate is performed in the Remove method.

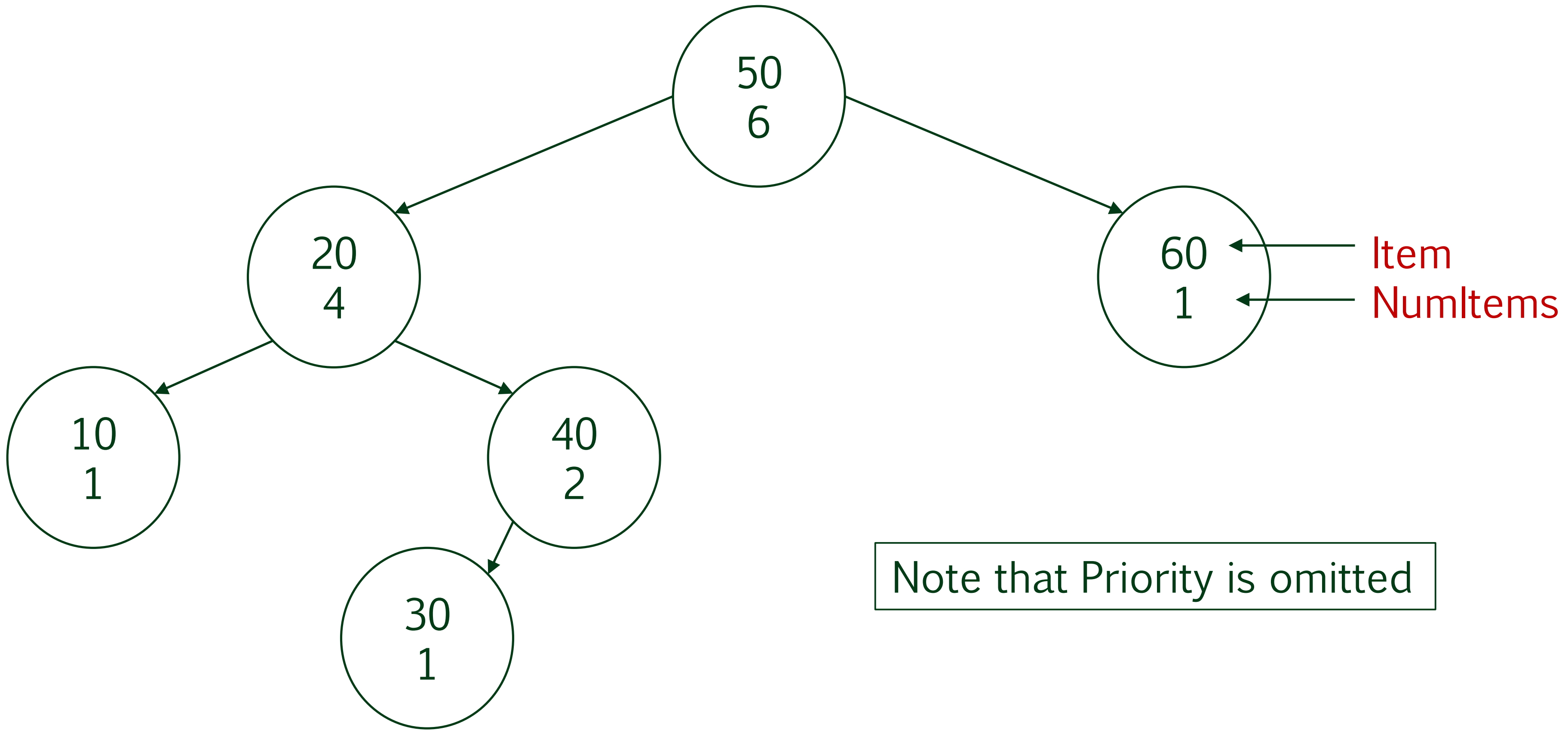
How can NumItems be used to support Rank?

- Two versions of Rank:
  - Return the item of a given rank (call this Rank I)
  - Return the rank of a given item (call this Rank II)

Rank I

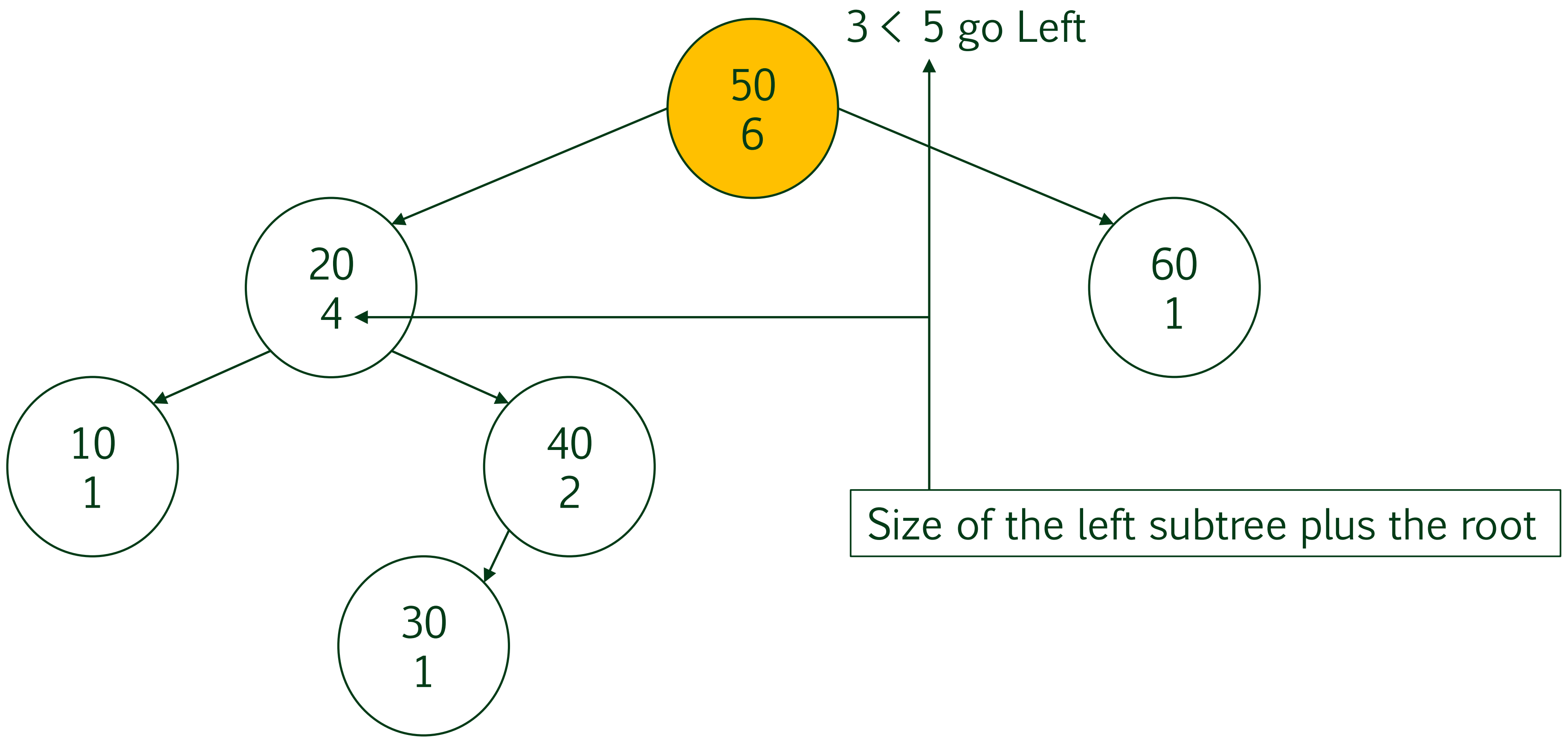
Return the item of a given rank

Consider the following treap



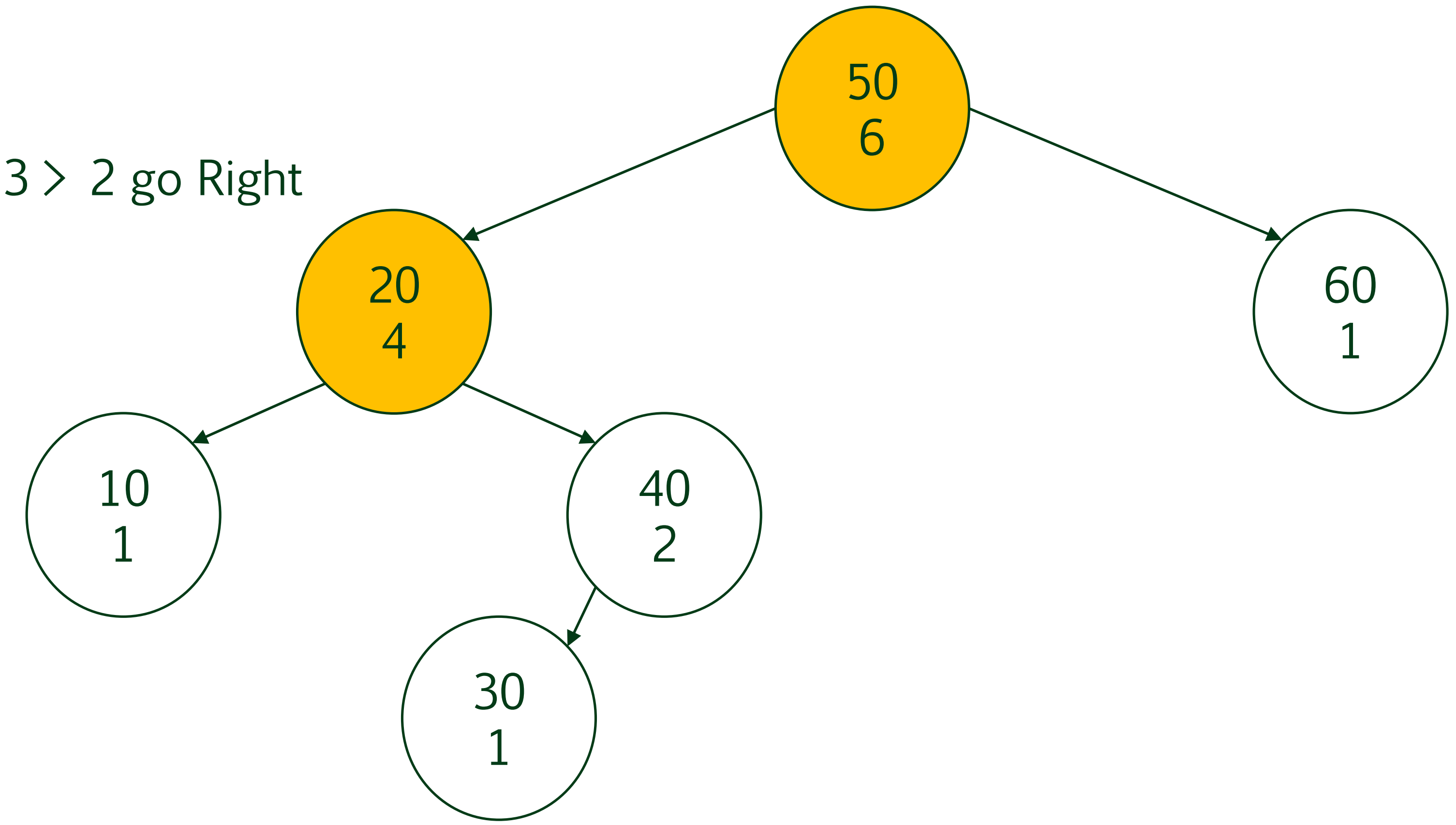
Note that Priority is omitted

Find the item of rank 3

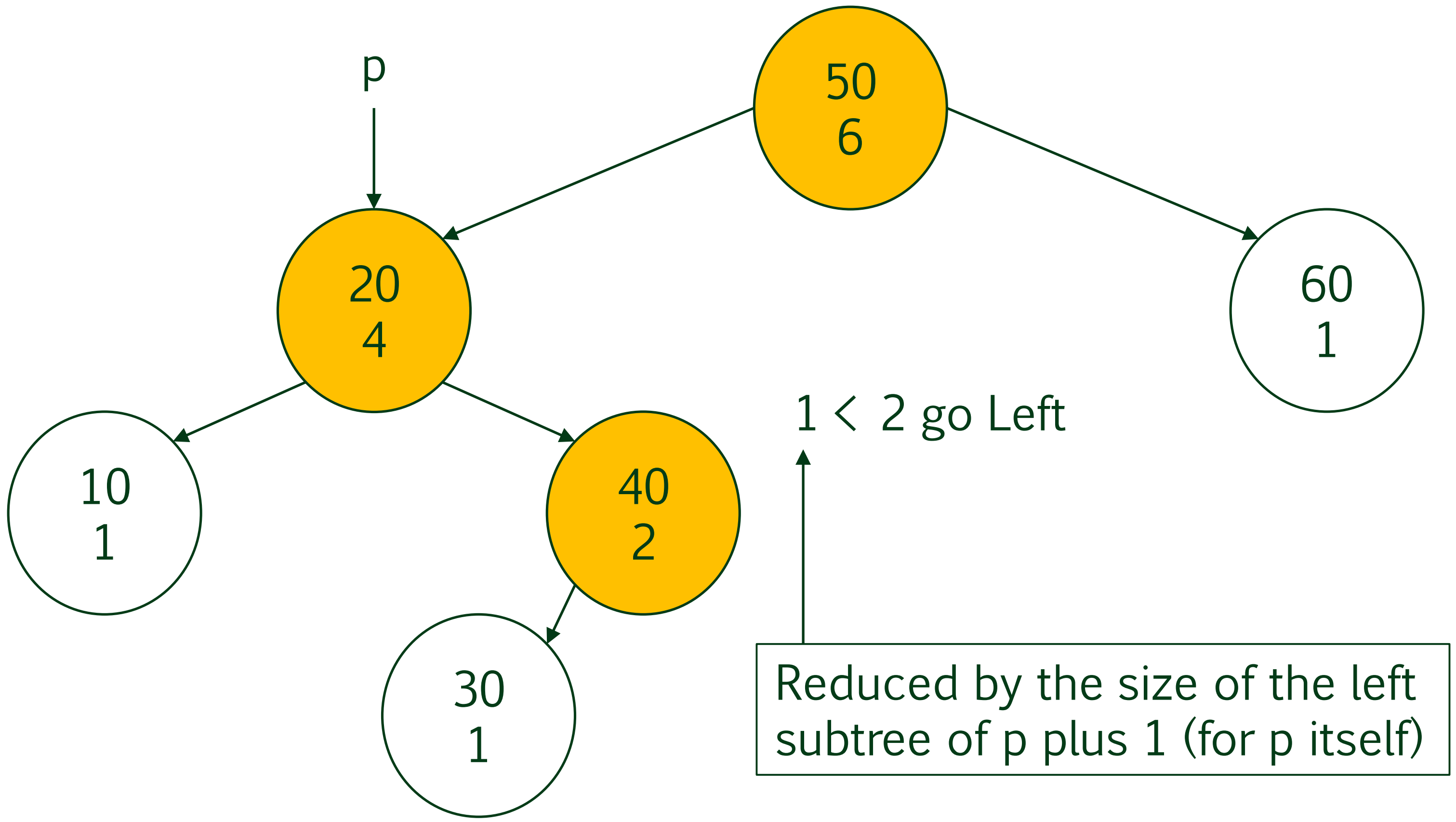


Find the item of rank 3

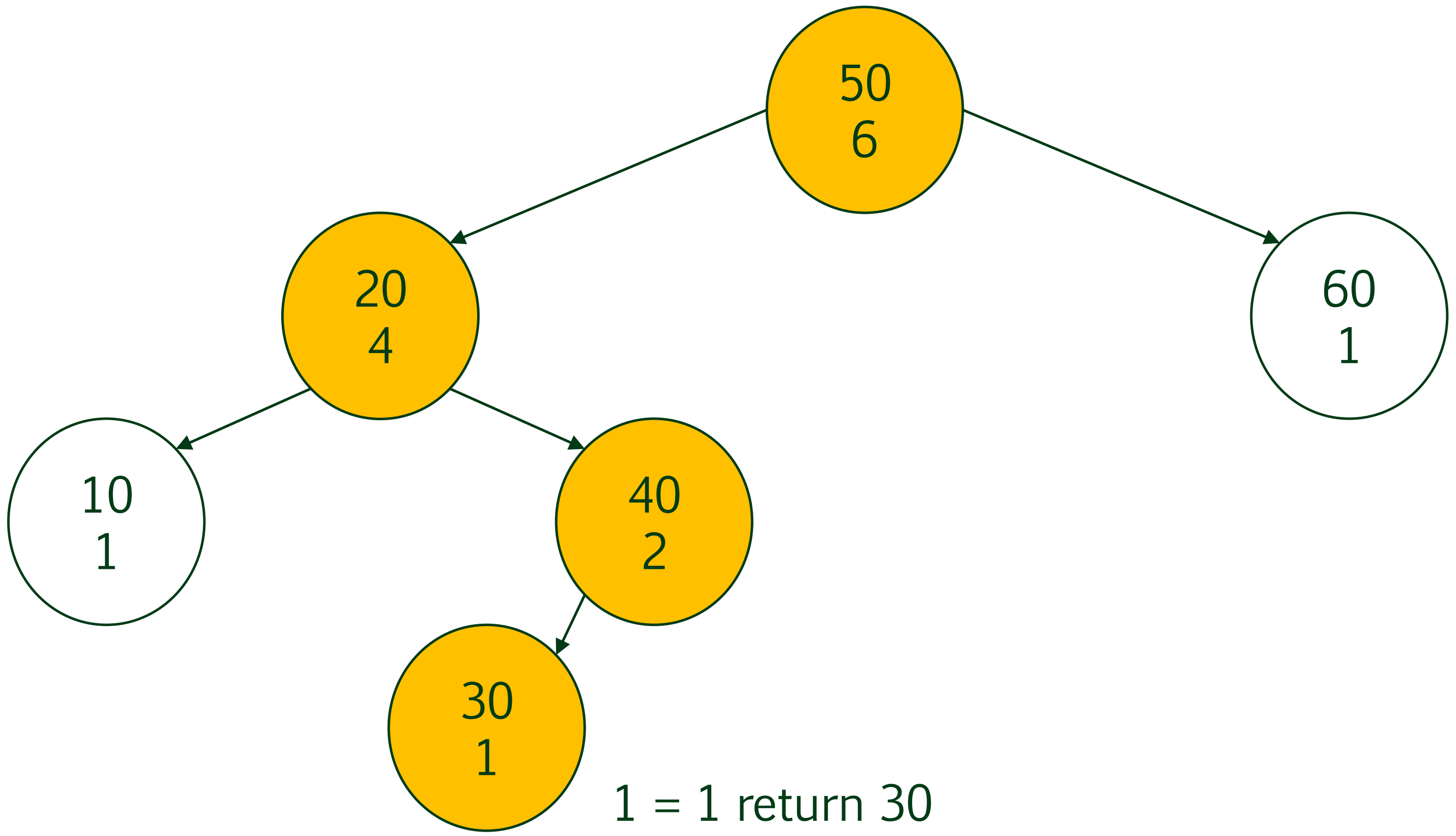
3 > 2 go Right



Find the item of rank 3

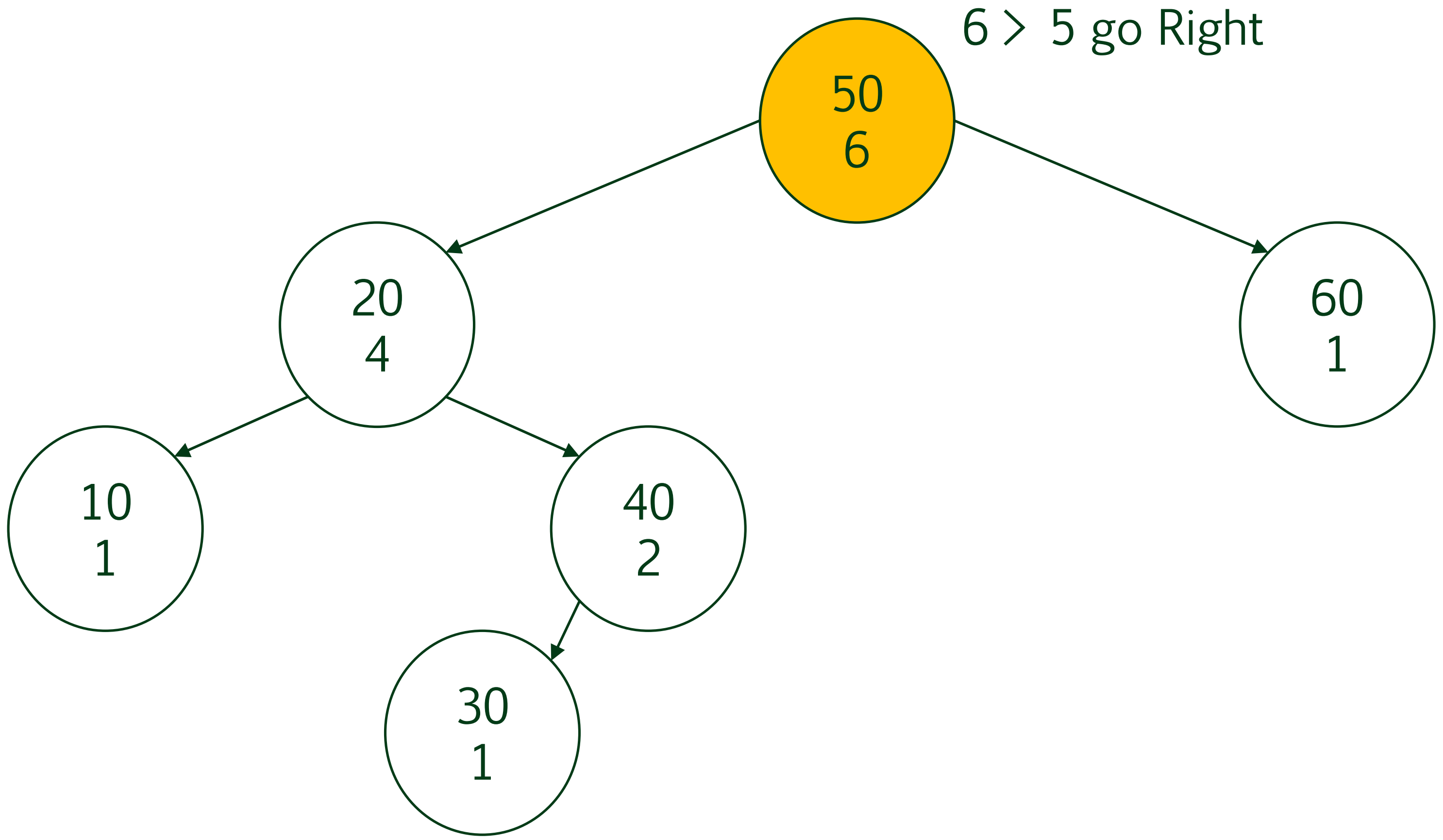

p
50
6
20
4
60
1
10
1
40
2
30
1
1 < 2 go Left
Reduced by the size of the left subtree of p plus 1 (for p itself)

Find the item of rank 3

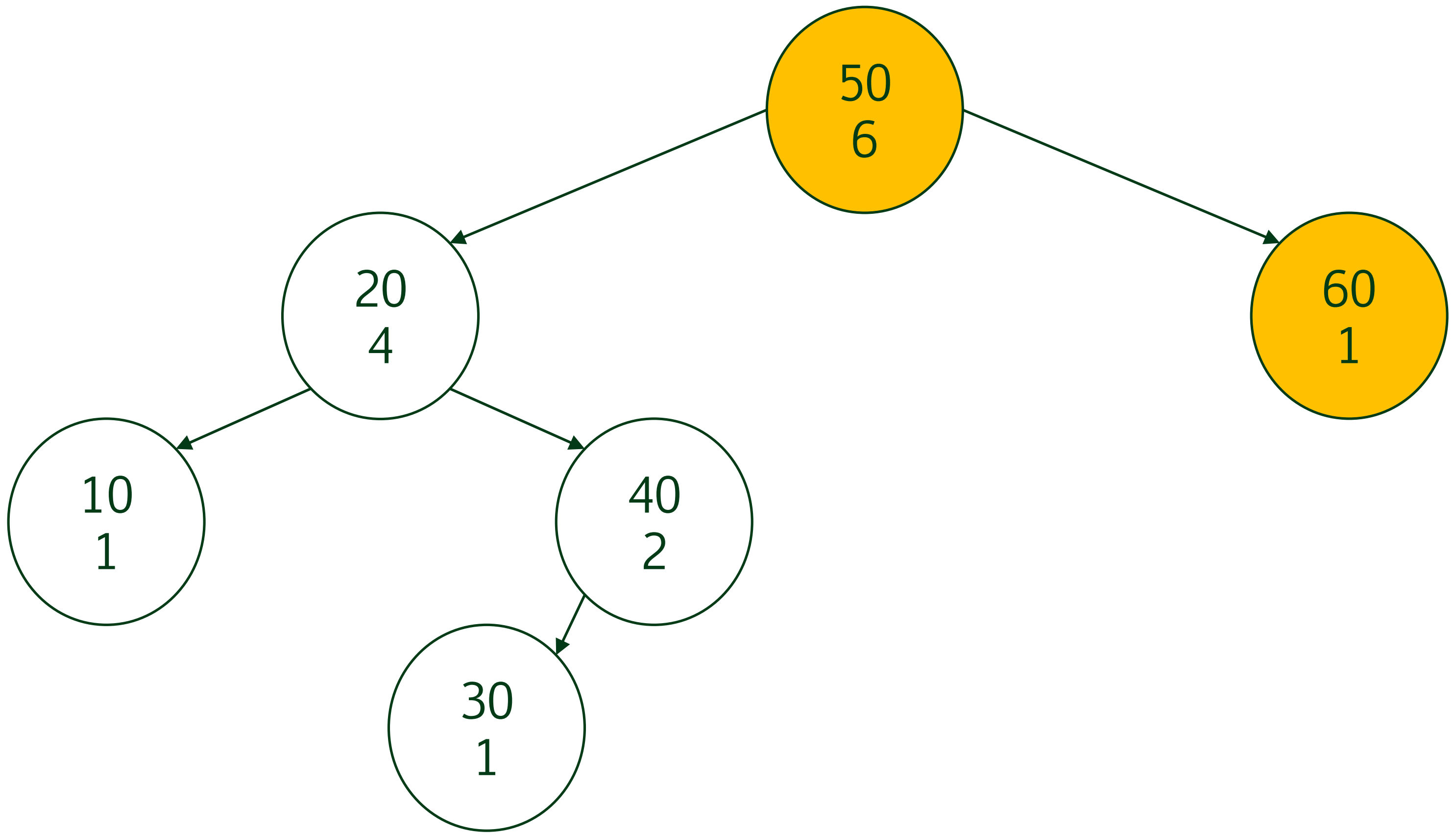


1 = 1 return 30

Find the item of rank 6

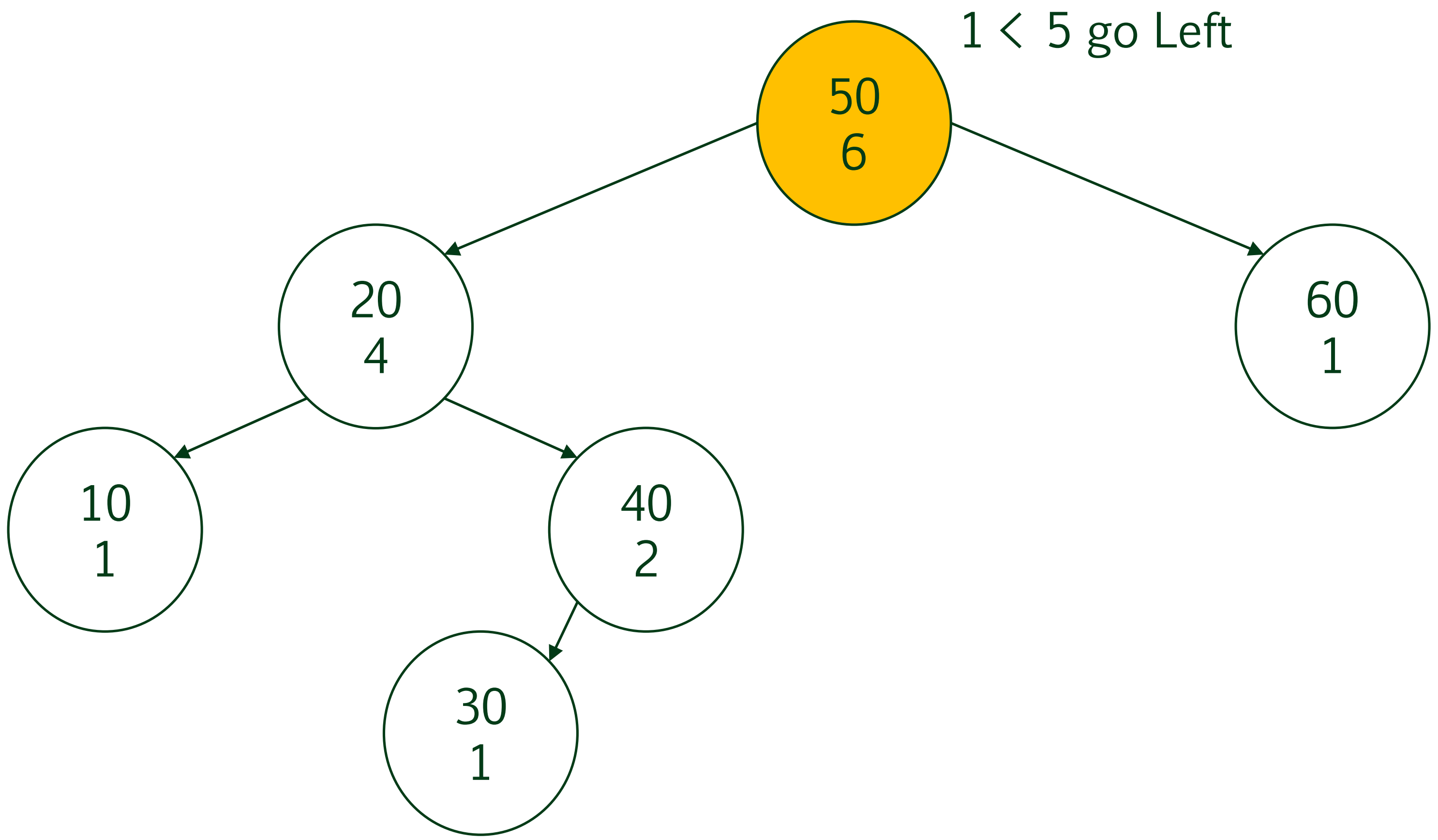


Find the item of rank 6



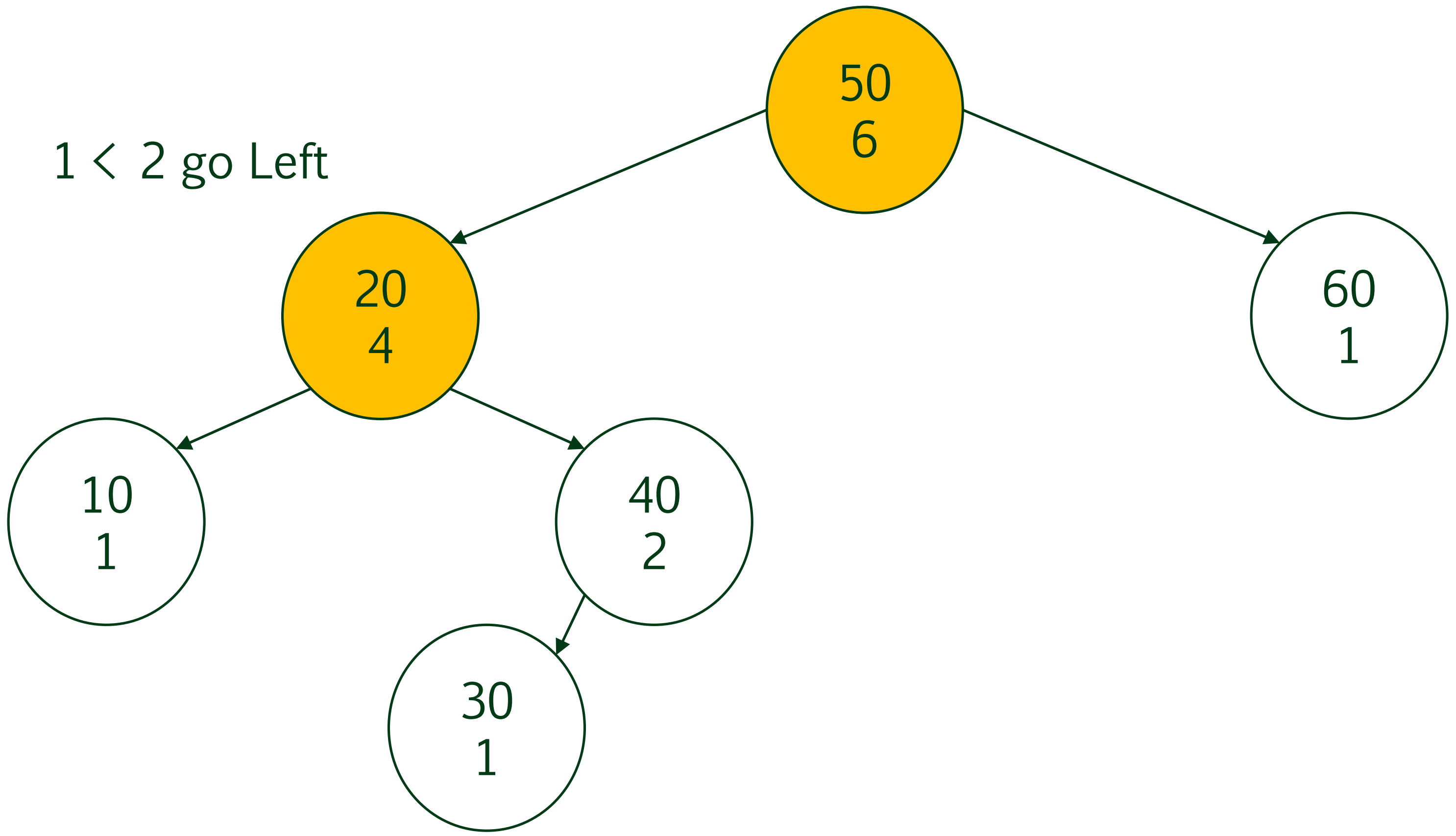
1 = 1 return 60

Find the item of rank 1

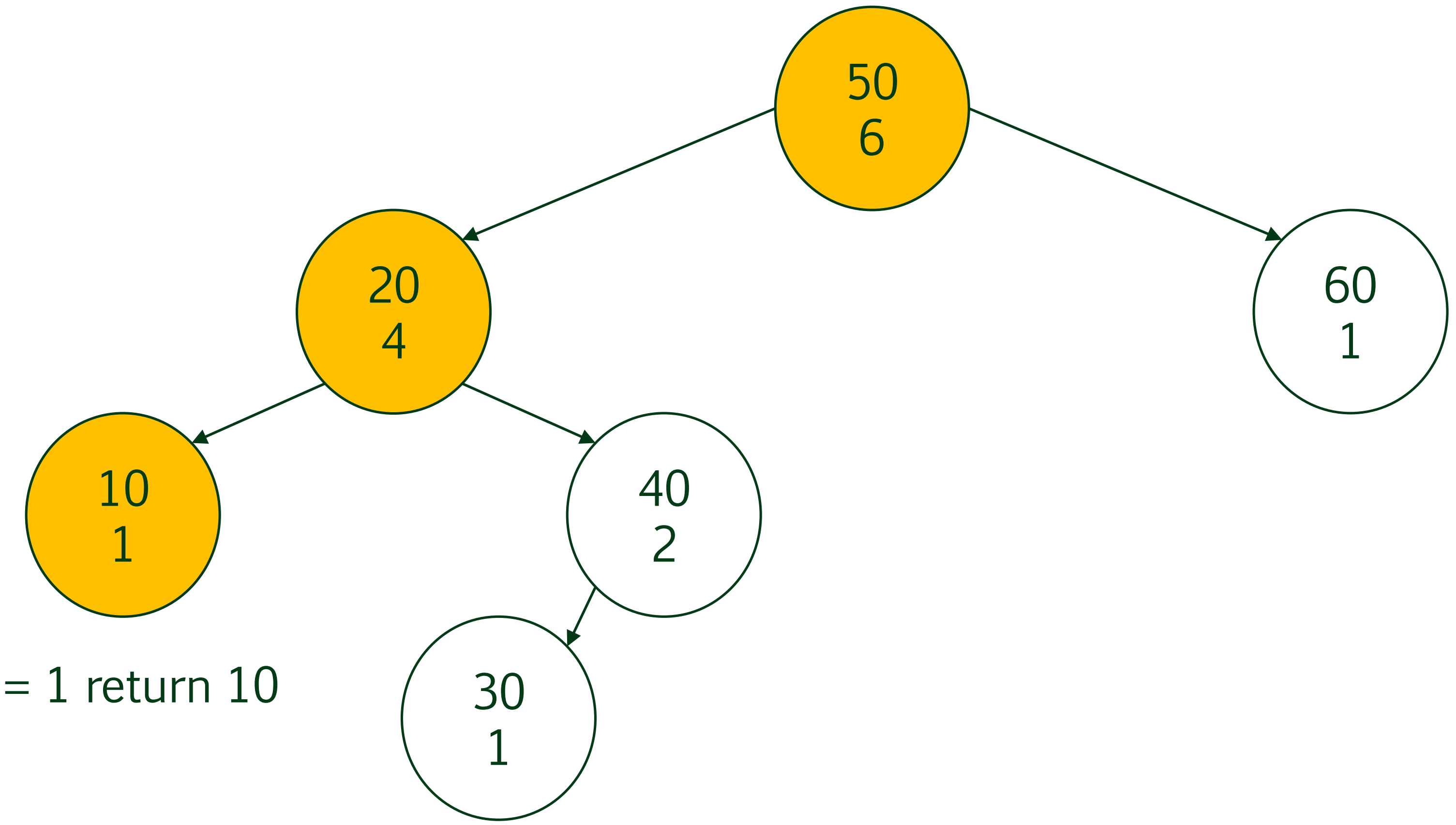


Find the item of rank 1

1 < 2 go Left



Find the item of rank 1



1 = 1 return 10

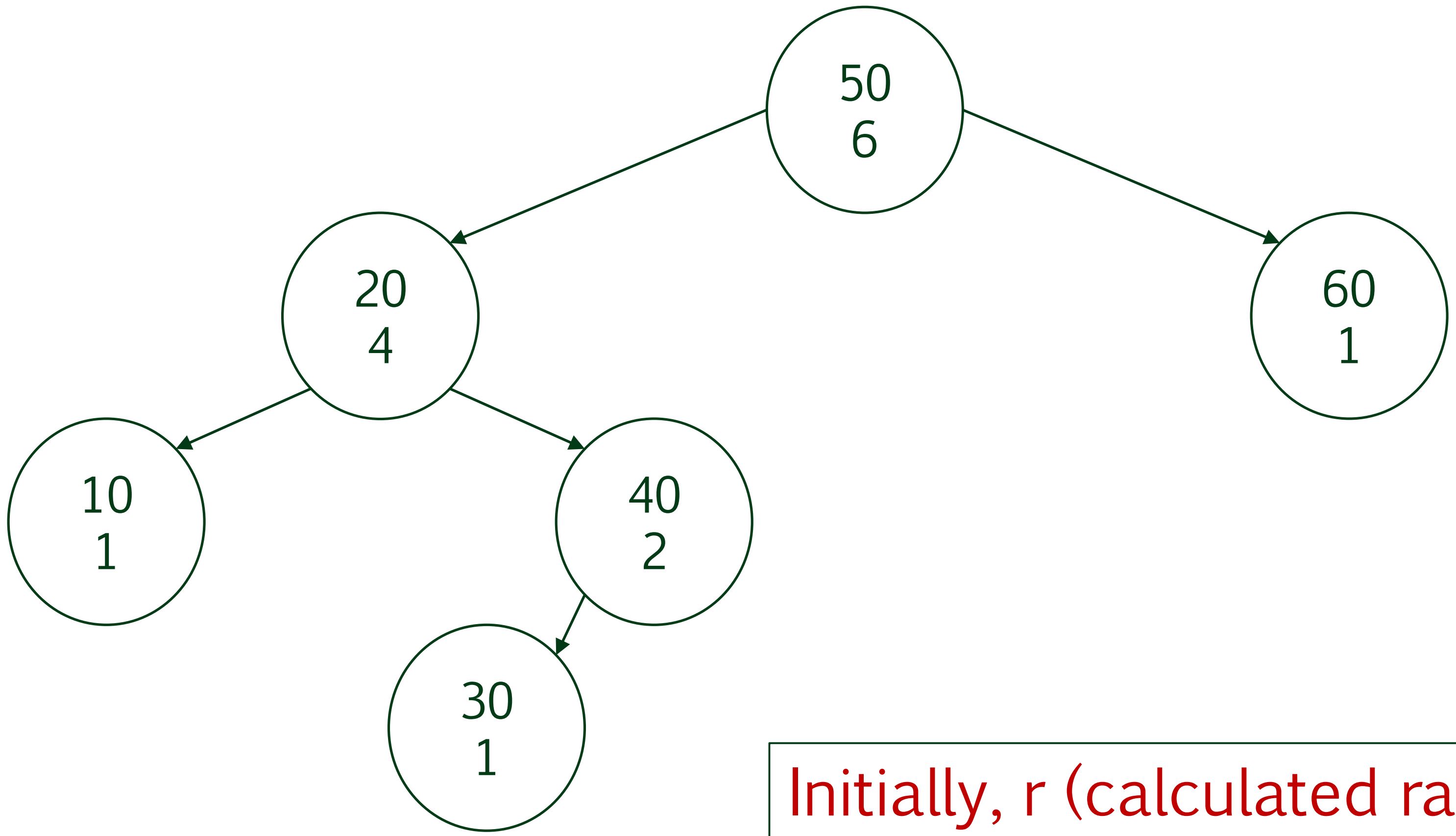
Exercises

- Add two children to each leaf node (10, 30 and 60) and adjust NumItems for each Node in the treap.
- Find the items of rank 4, 8 and 11 in your treap above.

Rank II

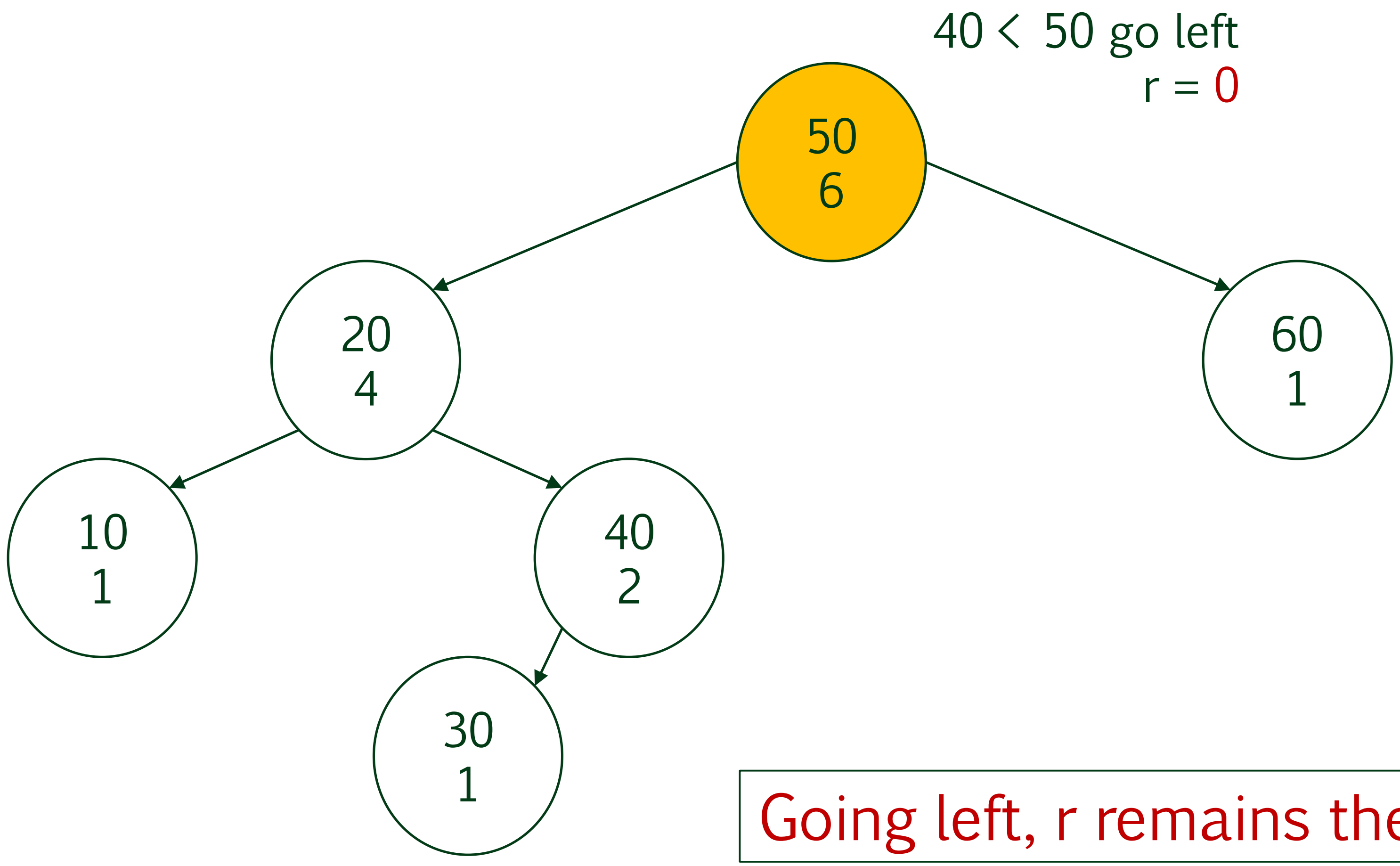
Return the rank of a given item

Consider the following treap



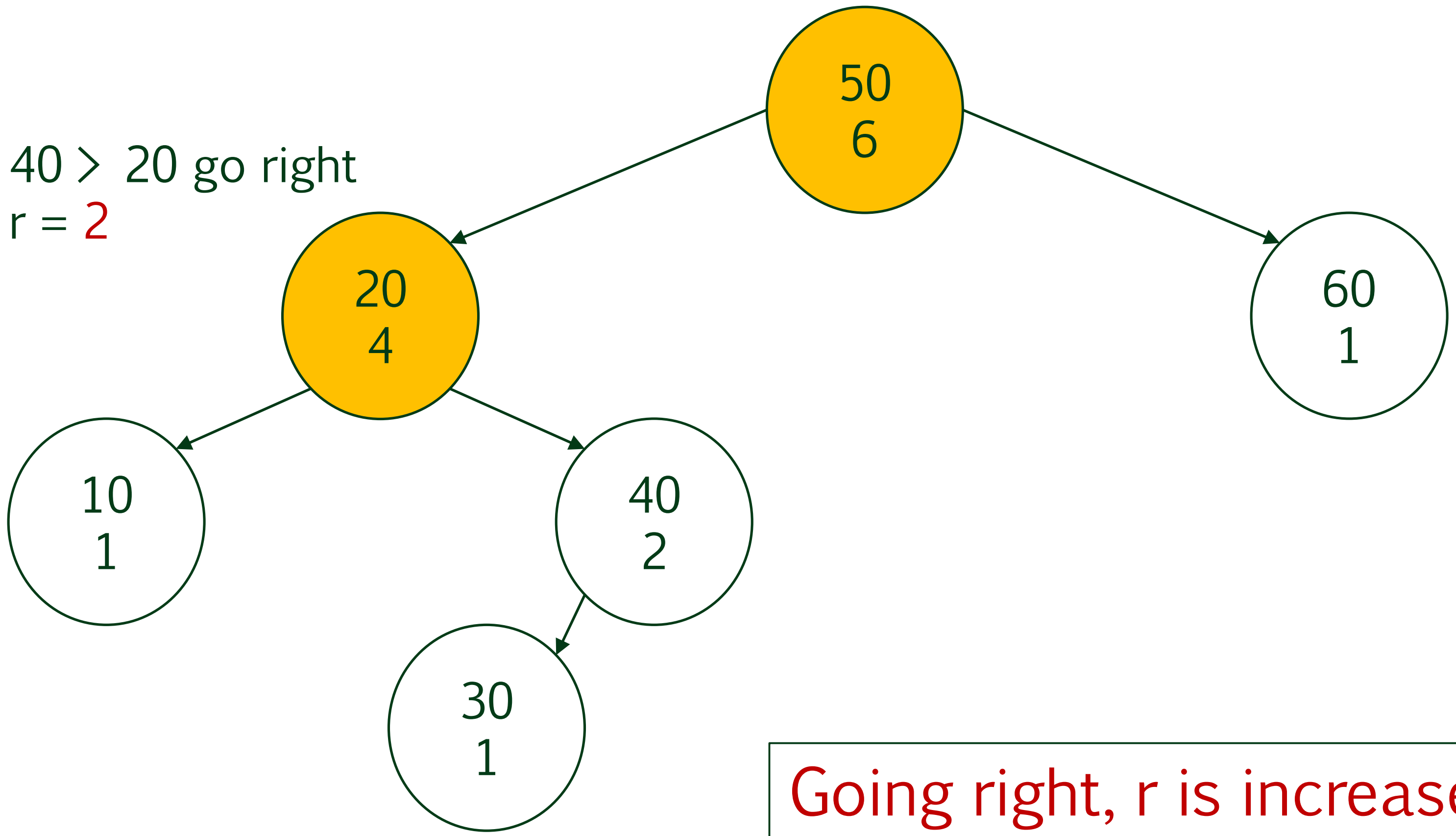
Initially, r (calculated rank) is initialized to 0

Find the rank of 40



Going left, r remains the same

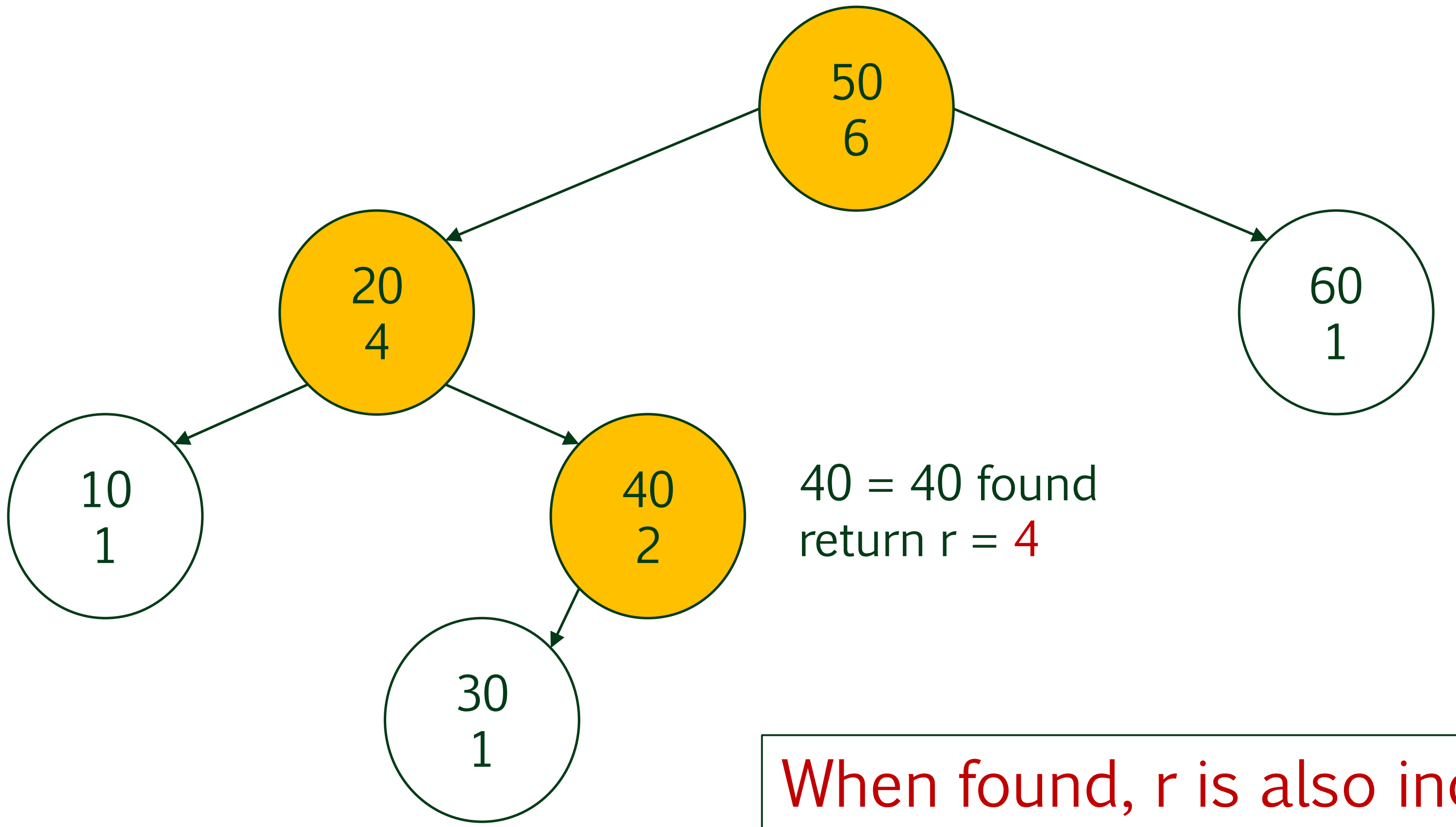
Find the rank of 40



40 > 20 go right
r = 2

Going right, r is increased by the size of the left subtree plus 1

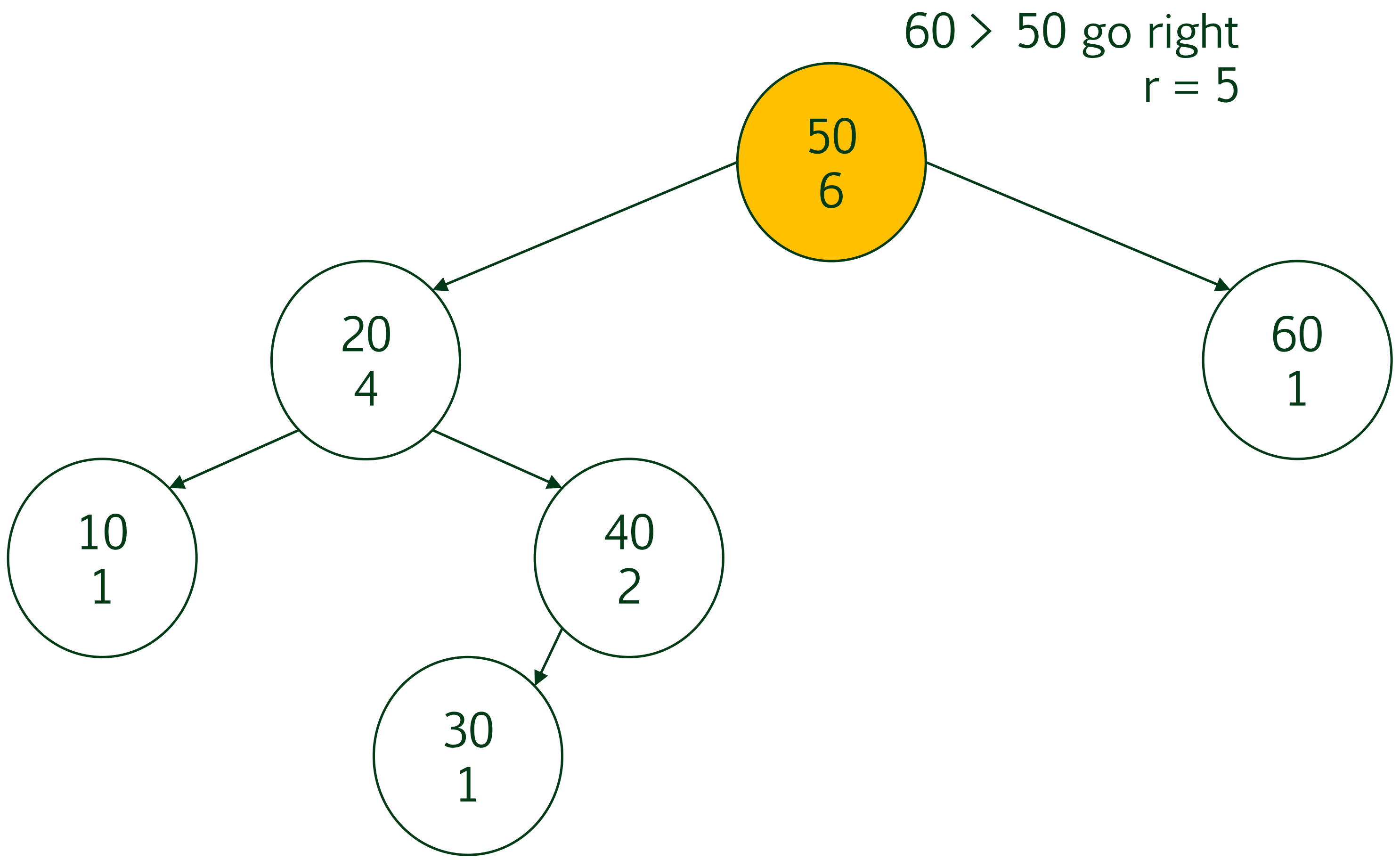
Find the rank of 40



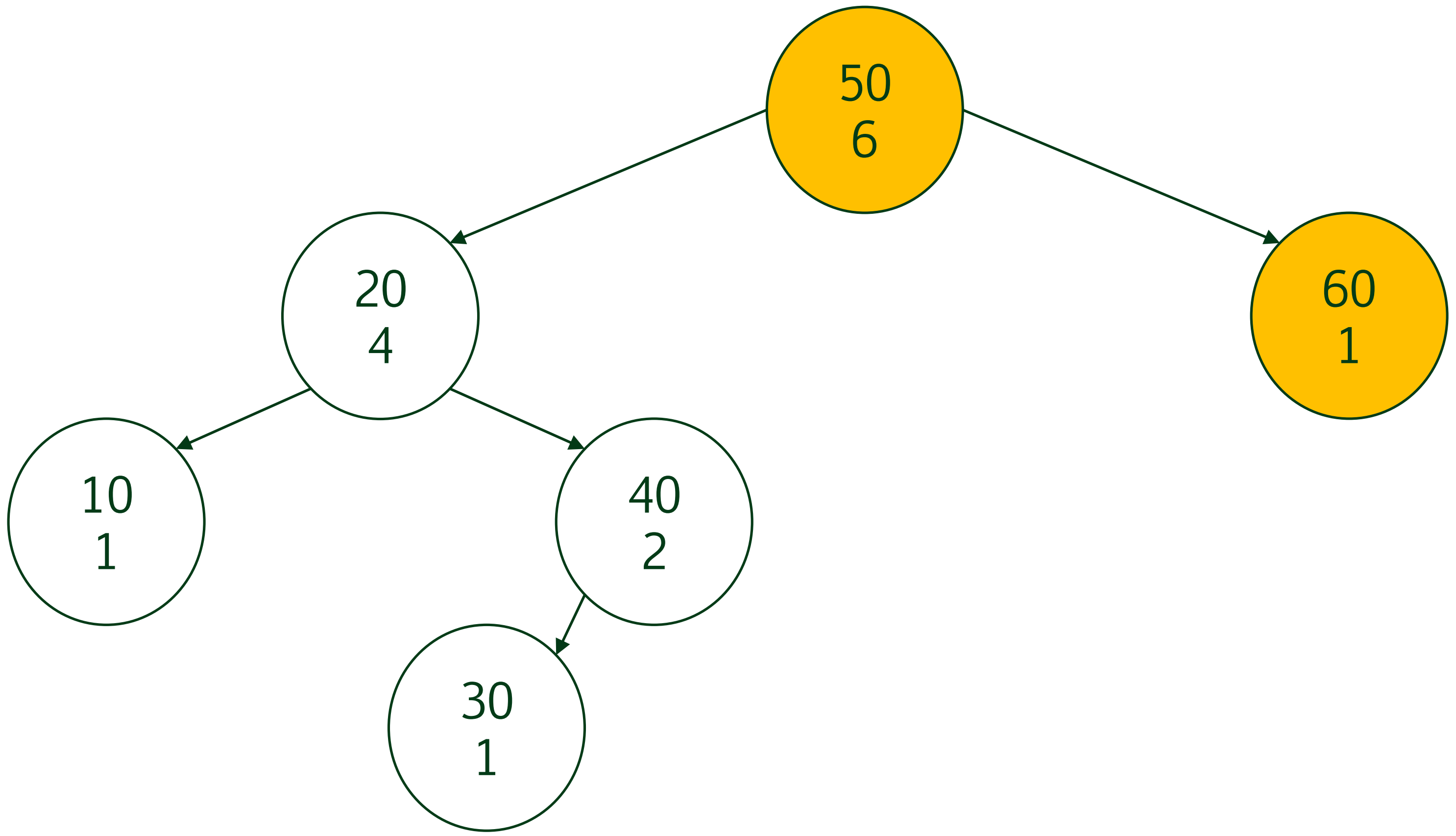
40 = 40 found
return r = 4

When found, r is also increased by the size of the left subtree plus 1

Find the rank of 60

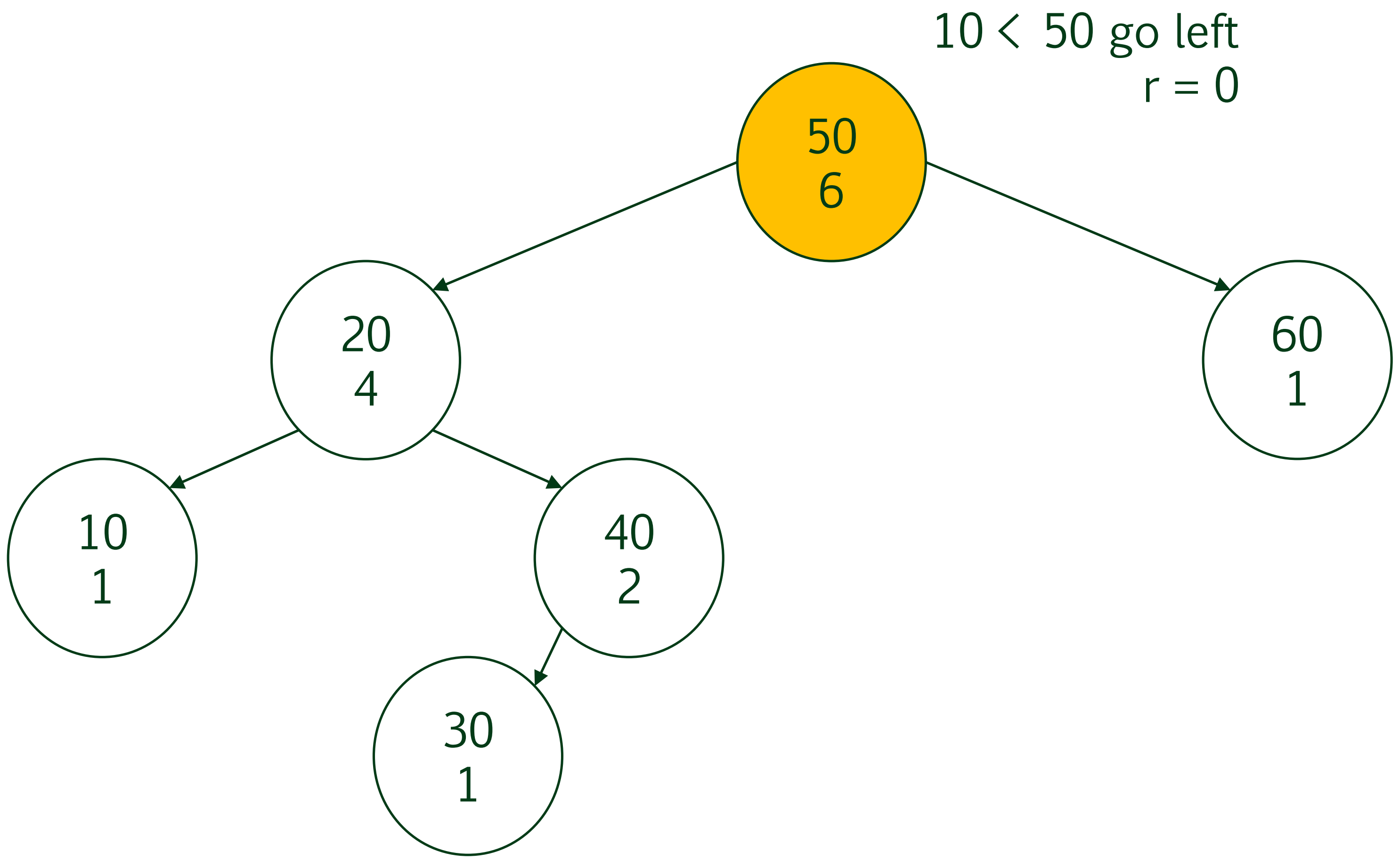


Find the rank of 60

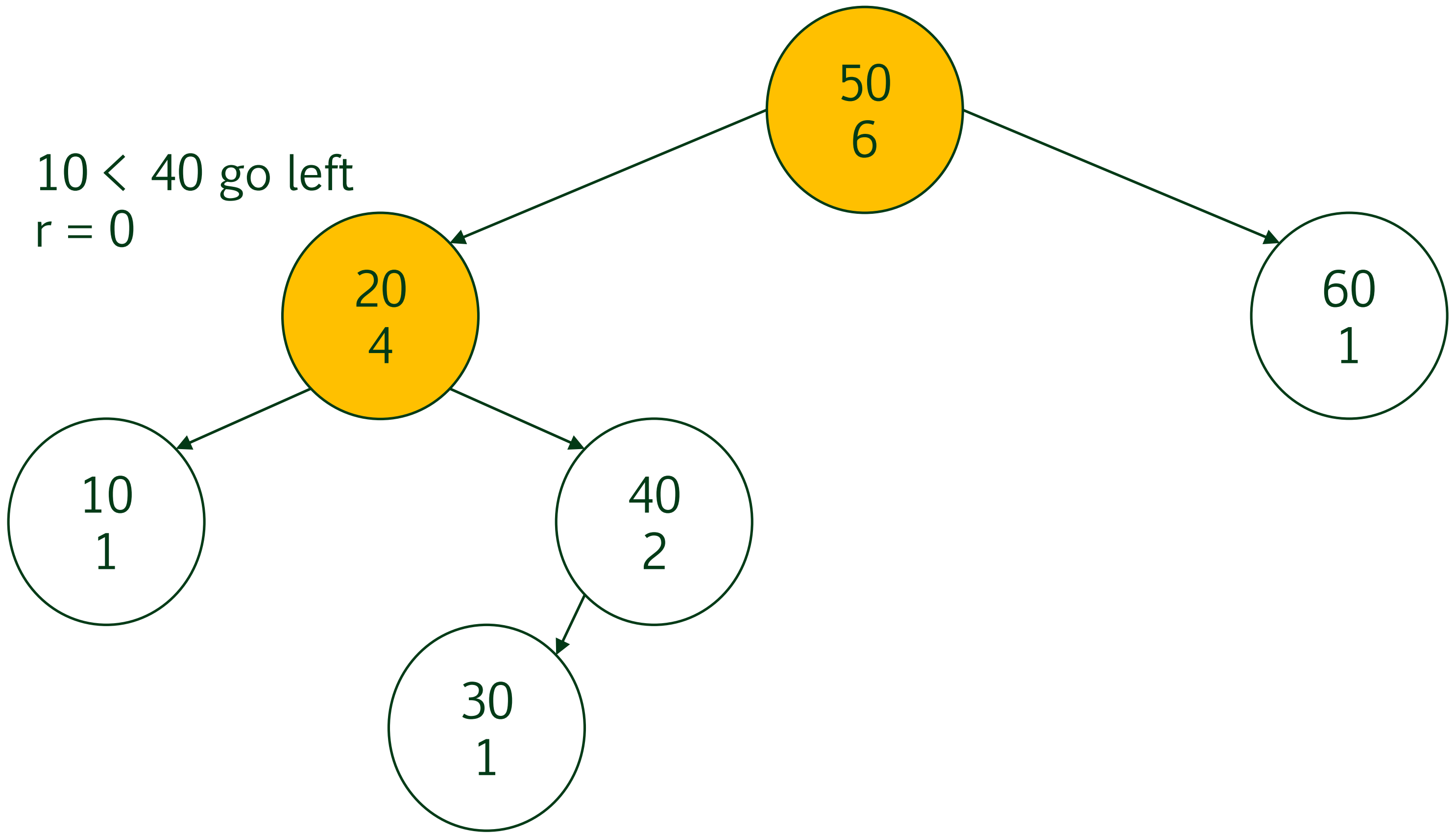


60 = 60 found
return r = 6

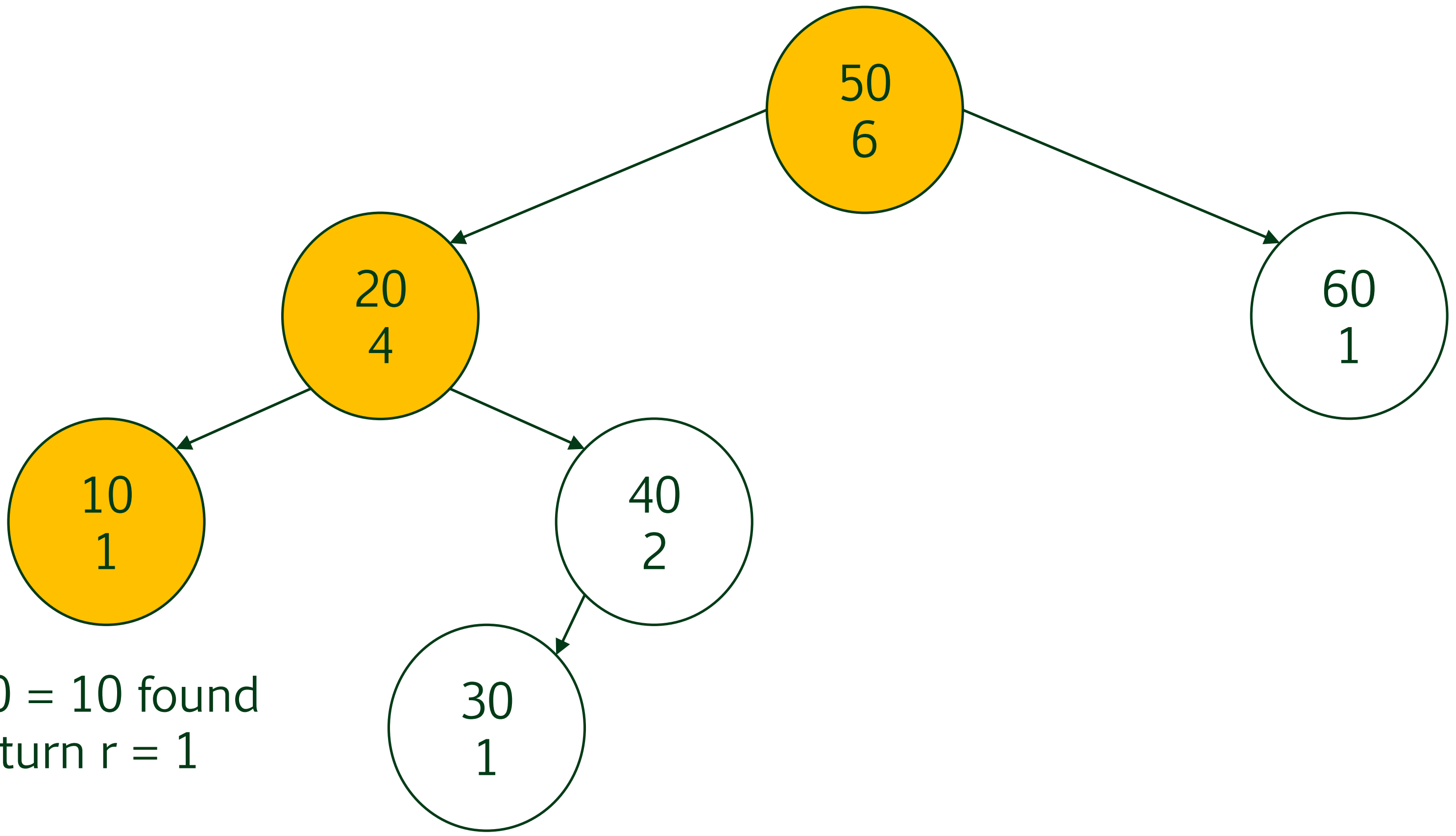
Find the rank of 10



Find the rank of 10



Find the rank of 10



10 = 10 found
return r = 1

Exercises

- Using the implementation of the augmented treap given on Blackboard, initialize an augmented treap with 15 items.
- Select 4 items in the treap and find their ranks.

Time complexities of Rank I and Rank II

- In both cases, the maximum number of comparisons is h+1 where h is the height of the treap
- Since the expected height of the treap is O(log n), the expected time complexities of Rank I and Rank II are also O(log n)
- The expected time complexity of O(log n) is far superior to the average case time complexity of O(n) without NumItems