Department of
Computer Engineering

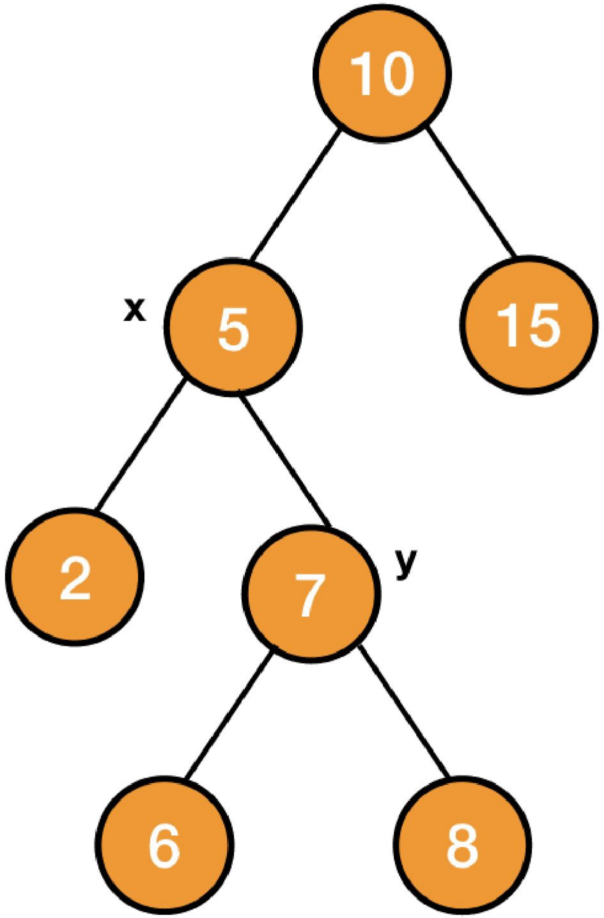
Data Structure & Algorithms

Red Black Trees Rotations

Rotations in Binary Search Tree

- There are two types of rotations:
  1. Left Rotation
  2. Right Rotation
- In left rotation, we assume that the right child is not null. Similarly, in the right rotation, we assume that the left child is not null.
- Consider the following tree:

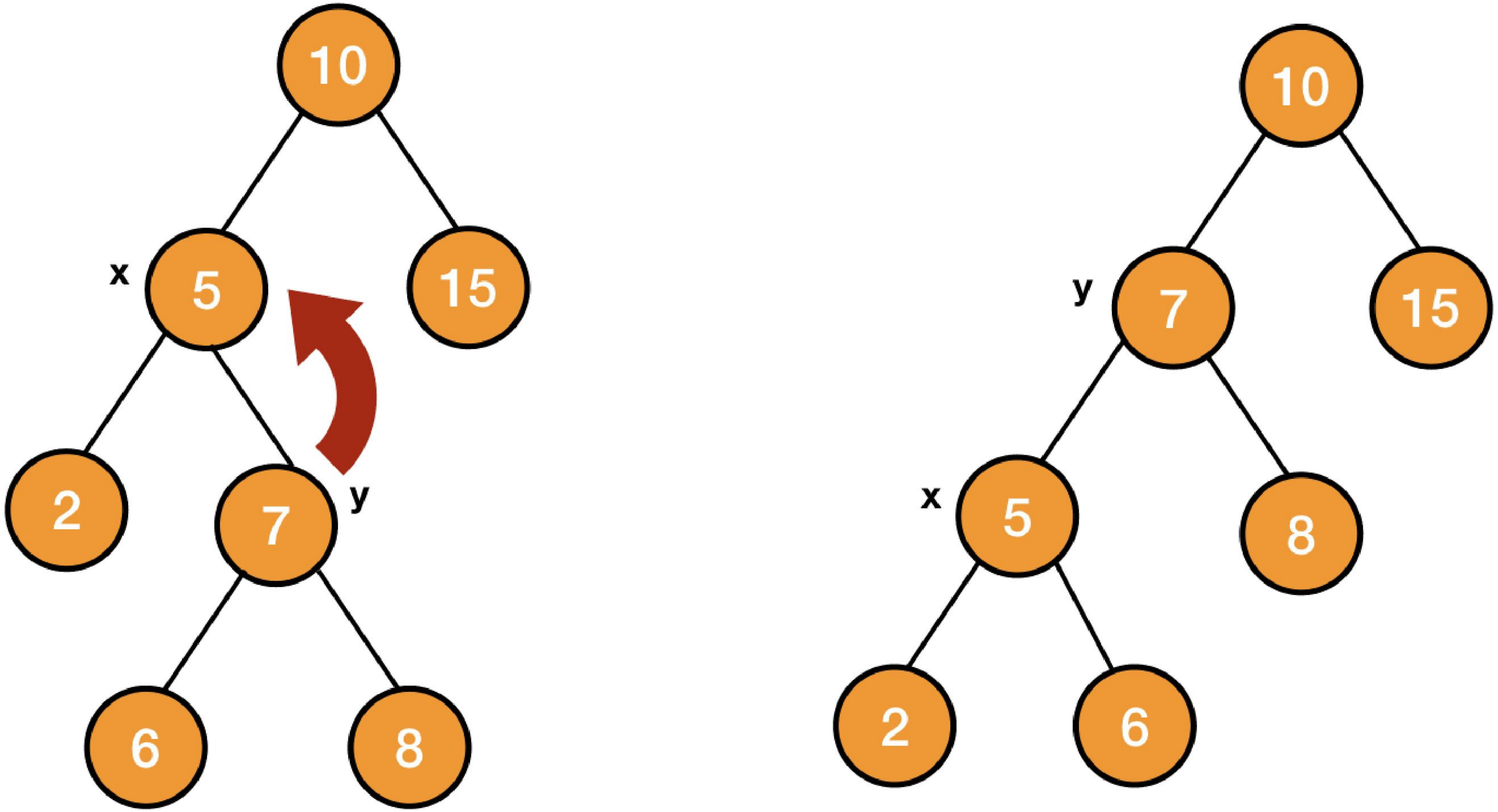
Rotations in Binary Search Tree



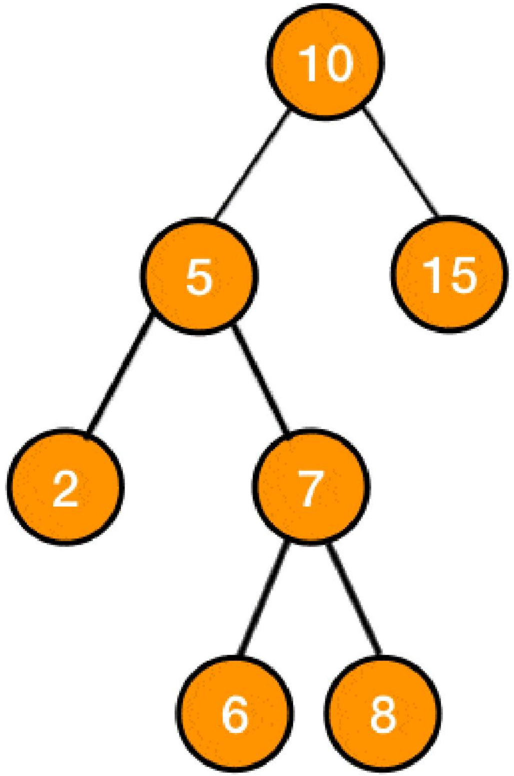
Rotations in Binary Search Tree

- After applying left rotation on the node x, the node y will become the new root of the subtree and its left child will be x. And the previous left child of y will now become the right child of x.

Left Rotations in Binary Search Tree



Left Rotations in Binary Search Tree

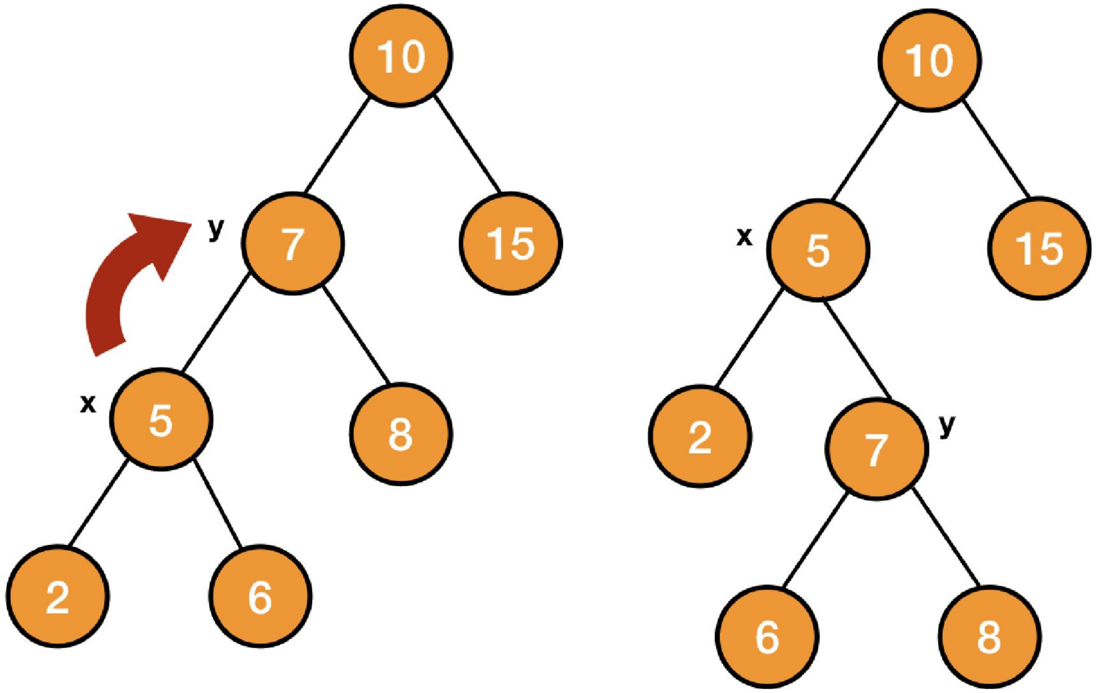


Right Rotations in Binary Search Tree

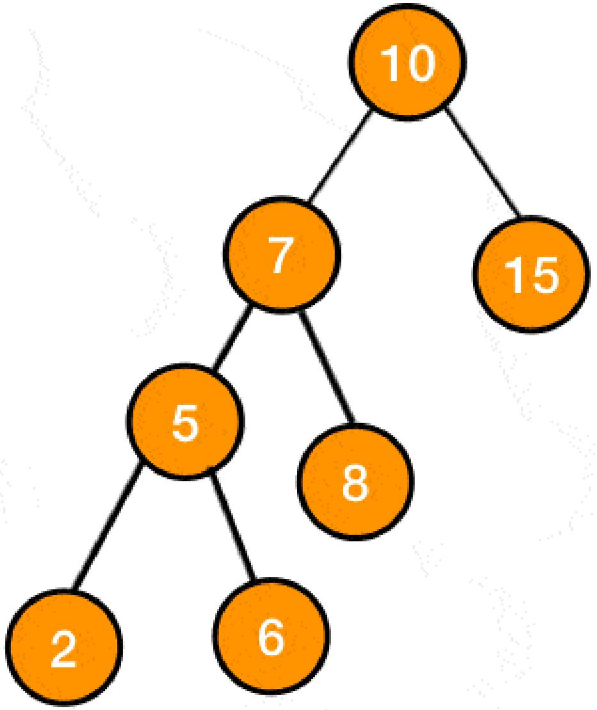
- Now applying right rotation on the node y of the rotated tree, it will transform back to the original tree.

Right Rotations in Binary Search Tree

- So right rotation on the node y will make x the root of the tree, y will become x's right child. And the previous right child of x will now become the left child of y.



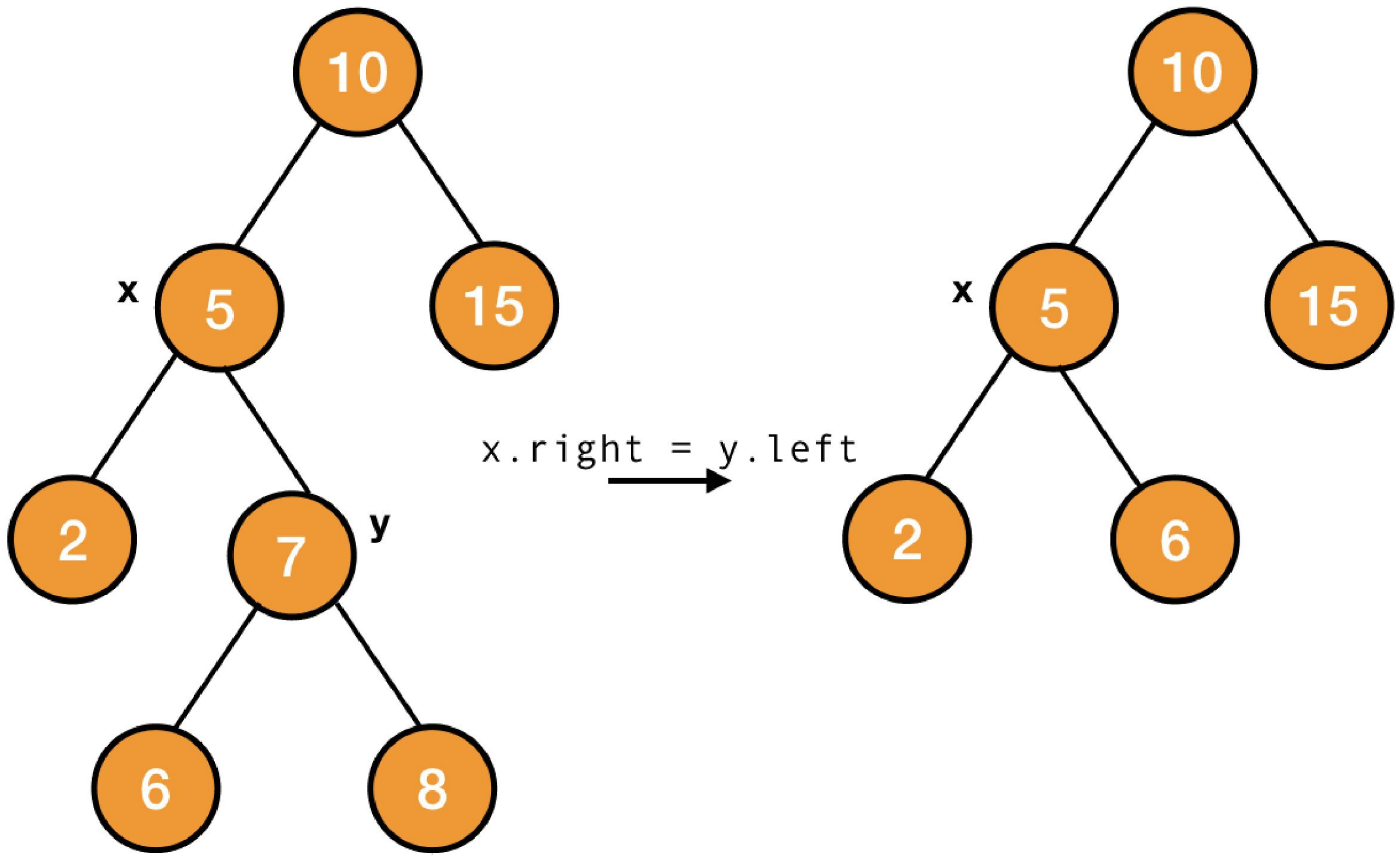
Right Rotations in Binary Search Tree



Code of Rotations

- We are going to explain the code for left rotation here. The code for the right rotation will be symmetric.
- We need the tree T and the node x on which we are going to apply the rotation – $LEFT\_ROTATION(T, x)$.
- The left grandchild of x (left child of the right child x) will become the right child of it after rotation. To do so let's mark the right child of x as left child of y.

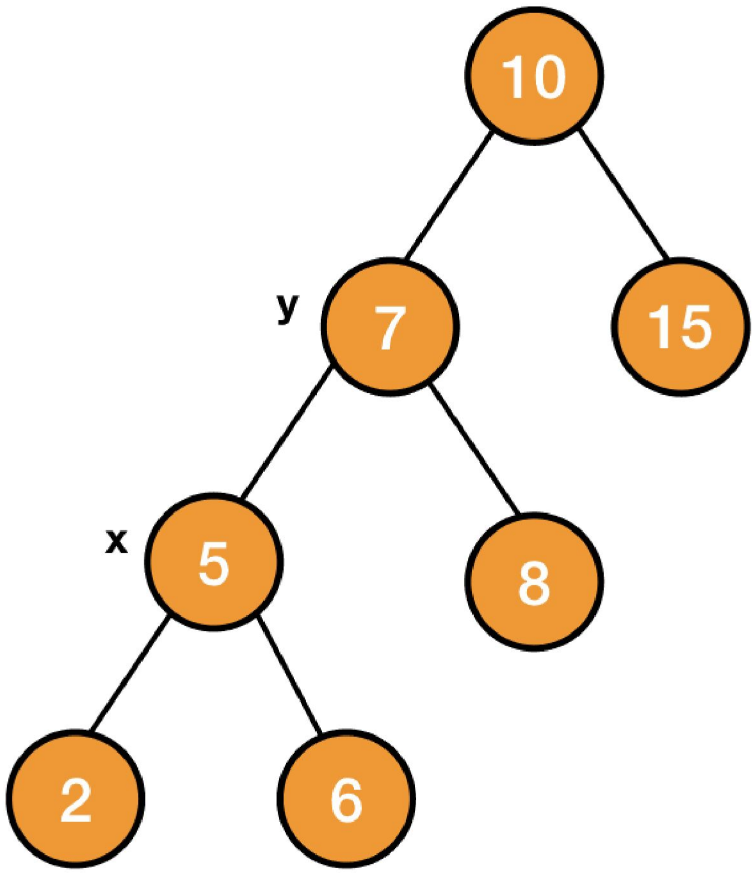
Code of Rotations



Code of Rotations

- The left child of y is going to be the right child of $x - x.right \; = \; y.left$. We also need to change the parent of $y.left$ to x. We will do this if the left child of y is not $NULL$.
- Then we need to put y to the position of x. We will first change the parent of y to the parent of x - $y.parent \; = \; x.parent$. After this, we will make the node x the child of $y's$ parent instead of y. We will do so by checking if y is the right or left child of its parent. We will also check if y is the root of the tree.
- At last, we need to make x the left child of y.

Code of Rotations



Algorithm of Rotations

```
LEFT_ROTATETION(T, x)
  y = x.right
  x.right = y.left
  if y.left != NULL
      y.left.parent = x
  y.parent = x.parent
  if x.parent == NULL //x is root
      T.root = y
  elseif x == x.parent.left // x is left child
      x.parent.left = y
  else // x is right child
      x.parent.right = y
  y.left = x
  x.parent = y
```

Summary of Rotations

- From the above code, you can easily see that rotation is a constant time taking process $O(1)$.
- Now that we know how to perform rotation, we will use this to restore **red-black** properties when they get violated after adding or deleting any node.