

# COMP2521 26T1

## Balancing Binary Search Trees

Kevin Luxa

`cs2521@cse.unsw.edu.au`

balancing operations  
balancing methods

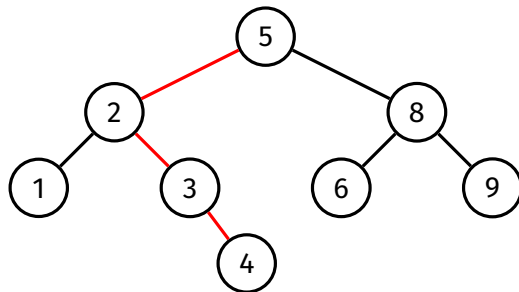
## BSTs Recap

## Balance

Balancing  
OperationsBalancing  
Methods

**Height** of a tree: Maximum distance between the root node and a leaf node

- For example, the height of the following tree is 3

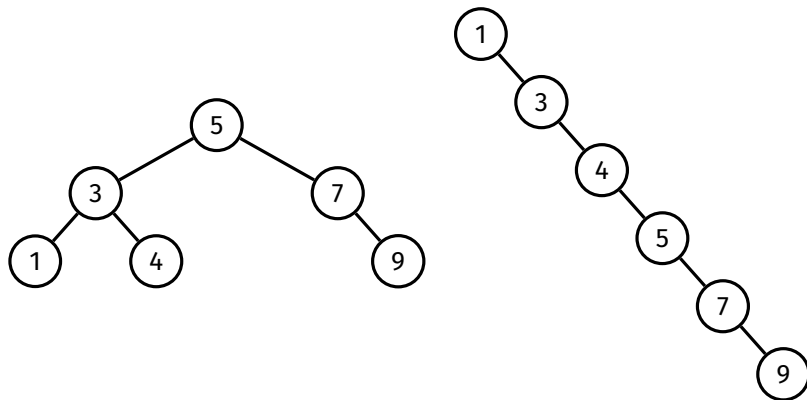


## BSTs Recap

## Balance

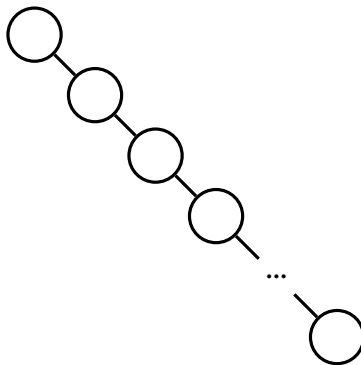
Balancing  
OperationsBalancing  
Methods

The structure, height, and hence  
**performance**  
of a binary search tree  
depends on the order of insertion.



## BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods**Worst case**Height of the tree is  $n - 1 = O(n)$ 

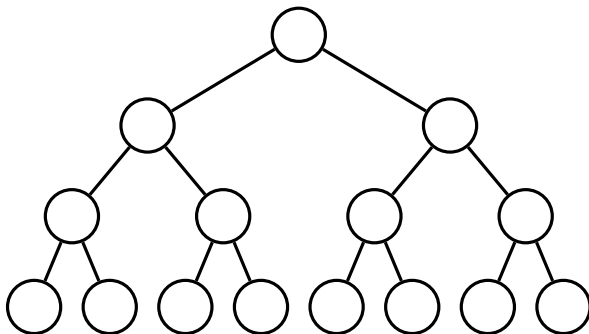
## BSTs Recap

## Balance

Balancing  
OperationsBalancing  
Methods

Best case

Height of the tree is  $\lfloor \log_2 n \rfloor = O(\log n)$



**SIZE-BALANCED**

a *size-balanced* tree has,  
for every node,

$$|\text{SIZE}(l) - \text{SIZE}(r)| \leq 1$$

**HEIGHT-BALANCED**

a *height-balanced* tree has,  
for every node,

$$|\text{HEIGHT}(l) - \text{HEIGHT}(r)| \leq 1$$

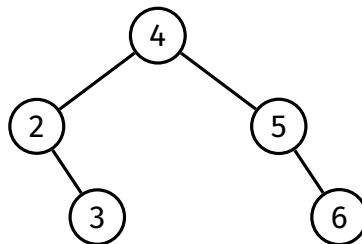
BSTs Recap

Balance

Examples

Balancing  
Operations

Balancing  
Methods



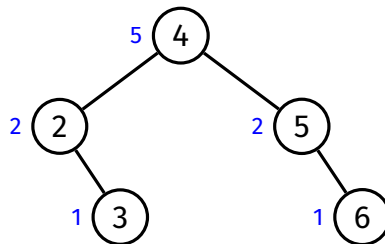
Size-balanced?

Height-balanced?

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

Size-balanced?

Yes

Height-balanced?

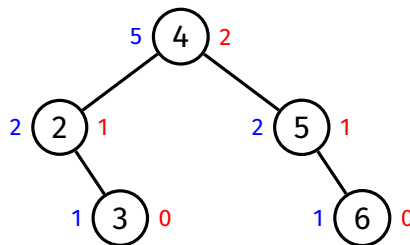
For every node,

$$|\text{SIZE}(l) - \text{SIZE}(r)| \leq 1$$

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

Size-balanced?

Yes

For every node,  
 $|\text{SIZE}(l) - \text{SIZE}(r)| \leq 1$

Height-balanced?

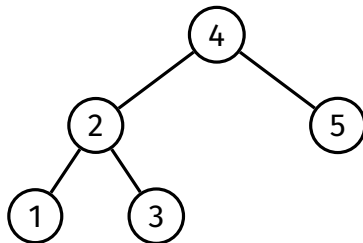
Yes

For every node,  
 $|\text{HEIGHT}(l) - \text{HEIGHT}(r)| \leq 1$

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

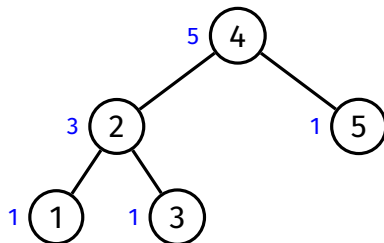
Size-balanced?

Height-balanced?

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

Size-balanced?

No

At node 4,

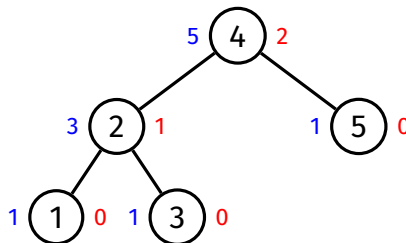
$$\begin{aligned} & |\text{SIZE}(l) - \text{SIZE}(r)| \\ &= |3 - 1| = 2 > 1 \end{aligned}$$

Height-balanced?

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

Size-balanced?

No

At node 4,

$$|\text{SIZE}(l) - \text{SIZE}(r)|$$
$$= |3 - 1| = 2 > 1$$

Height-balanced?

Yes

For every node,

$$|\text{HEIGHT}(l) - \text{HEIGHT}(r)| \leq 1$$

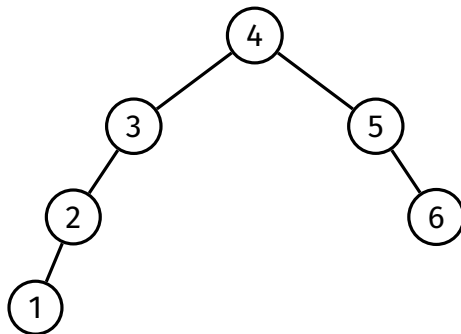
BSTs Recap

Balance

Examples

Balancing  
Operations

Balancing  
Methods



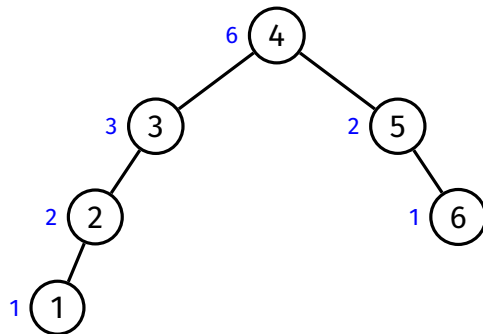
Size-balanced?

Height-balanced?

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

Size-balanced?

No

Height-balanced?

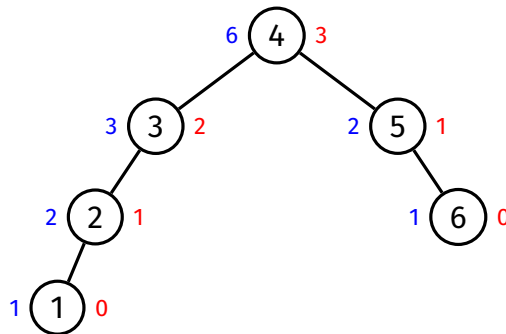
At node 3,

$$|\text{SIZE}(l) - \text{SIZE}(r)|$$
$$= |2 - 0| = 2 > 1$$

BSTs Recap

Balance

Examples

Balancing  
OperationsBalancing  
Methods

Size-balanced?

No

At node 3,

$$|\text{SIZE}(l) - \text{SIZE}(r)|$$
$$= |2 - 0| = 2 > 1$$

Height-balanced?

No

At node 3,

$$|\text{HEIGHT}(l) - \text{HEIGHT}(r)|$$
$$= |1 - (-1)| = 2 > 1$$

BSTs Recap

Balance

**Balancing  
Operations**

Rotations

Partition

Balancing  
Methods

## Rotation

- Left rotation
- Right rotation

## Partition

- Rearrange tree around a specified node by rotating it up to the root

BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

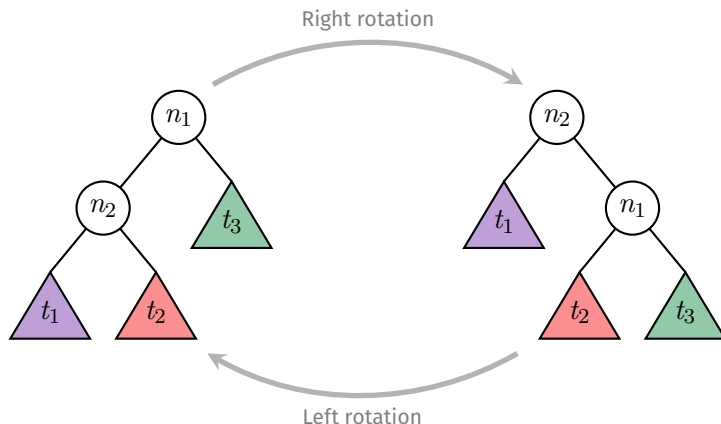
Implementation

Analysis

Partition

Balancing  
Methods

**LEFT ROTATION** and **RIGHT ROTATION**:  
a pair of operations  
that change the structure of a tree



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

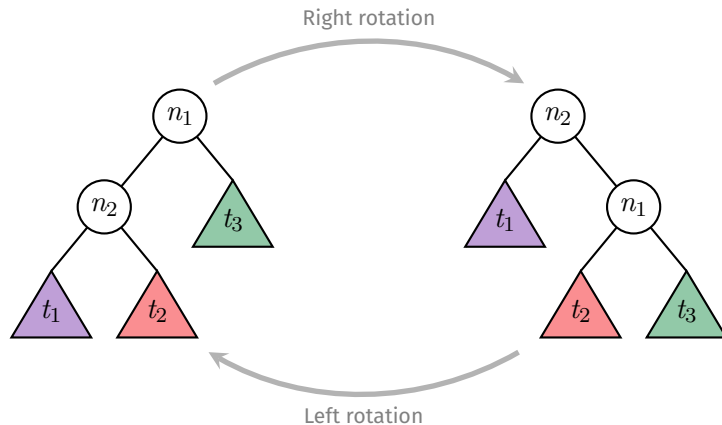
Implementation

Analysis

Partition

Balancing  
Methods

Binary search trees remain valid after a rotation



(all values in  $t_1$ )  $< n_2 <$  (all values in  $t_2$ )  $< n_1 <$  (all values in  $t_3$ )

BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

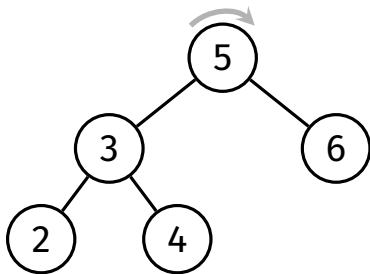
Implementation

Analysis

Partition

Balancing  
Methods

Rotate right at 5



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

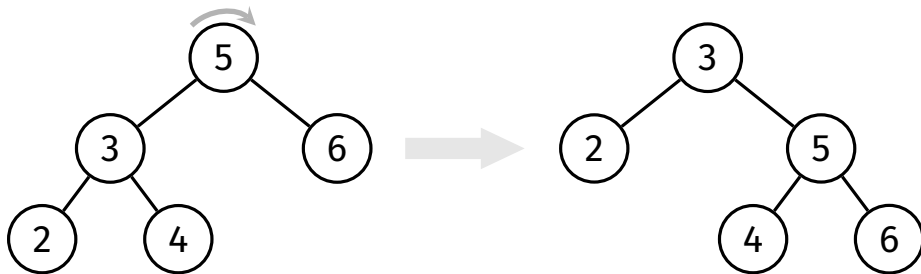
Implementation

Analysis

Partition

Balancing  
Methods

Rotate right at 5



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

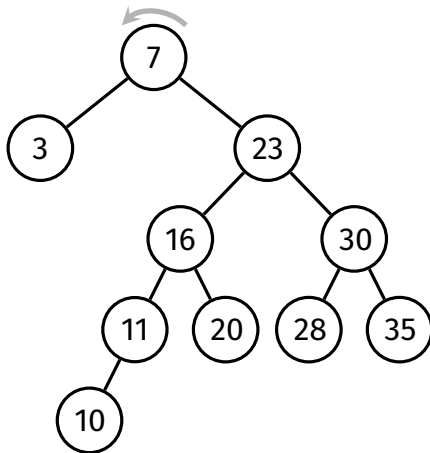
Implementation

Analysis

Partition

Balancing  
Methods

Rotate left at 7



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

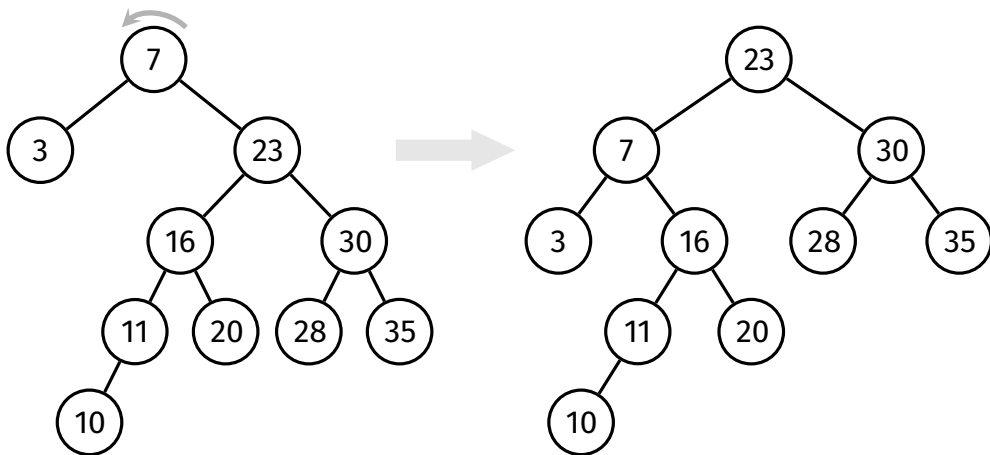
Implementation

Analysis

Partition

Balancing  
Methods

Rotate left at 7



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

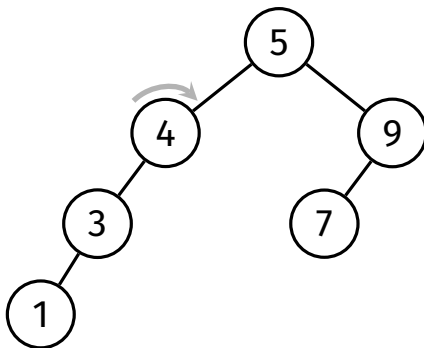
Implementation

Analysis

Partition

Balancing  
Methods

Rotate right at 4



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

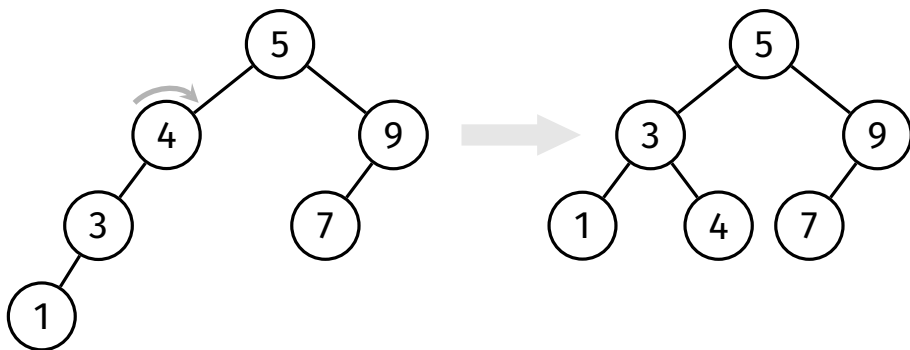
Implementation

Analysis

Partition

Balancing  
Methods

Rotate right at 4



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

Implementation

Analysis

Partition

Balancing  
Methods

```
struct node *rotateRight(struct node *root) {  
    if (root == NULL || root->left == NULL) return root;  
    struct node *newRoot = root->left;  
    root->left = newRoot->right;  
    newRoot->right = root;  
    return newRoot;  
}
```

```
struct node *rotateLeft(struct node *root) {  
    if (root == NULL || root->right == NULL) return root;  
    struct node *newRoot = root->right;  
    root->right = newRoot->left;  
    newRoot->left = root;  
    return newRoot;  
}
```

BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

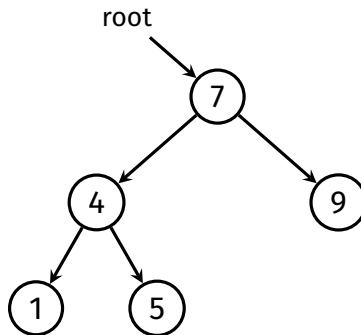
Implementation

Analysis

Partition

Balancing  
Methods

```
struct node *rotateRight(struct node *root) {  
    if (root == NULL || root->left == NULL) return root;  
  
    }  
}
```



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

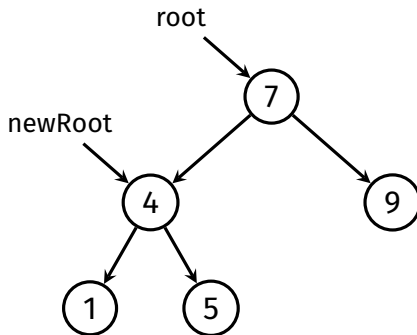
Implementation

Analysis

Partition

Balancing  
Methods

```
struct node *rotateRight(struct node *root) {  
    if (root == NULL || root->left == NULL) return root;  
    struct node *newRoot = root->left;  
  
    }
```



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

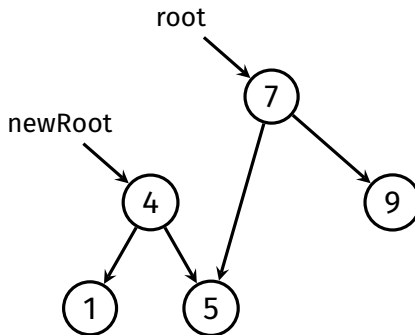
Implementation

Analysis

Partition

Balancing  
Methods

```
struct node *rotateRight(struct node *root) {  
    if (root == NULL || root->left == NULL) return root;  
    struct node *newRoot = root->left;  
    root->left = newRoot->right;  
  
}
```



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

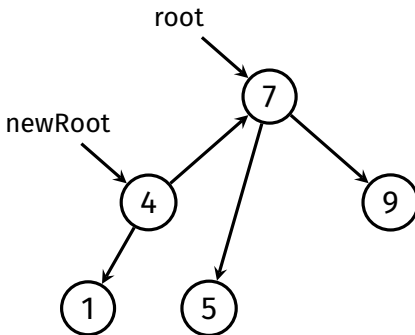
Implementation

Analysis

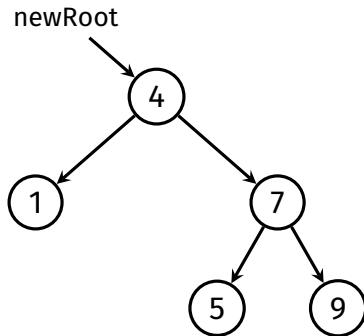
Partition

Balancing  
Methods

```
struct node *rotateRight(struct node *root) {  
    if (root == NULL || root->left == NULL) return root;  
    struct node *newRoot = root->left;  
    root->left = newRoot->right;  
    newRoot->right = root;  
}
```



```
struct node *rotateRight(struct node *root) {  
    if (root == NULL || root->left == NULL) return root;  
    struct node *newRoot = root->left;  
    root->left = newRoot->right;  
    newRoot->right = root;  
    return newRoot;  
}
```



BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

Implementation

**Analysis**

Partition

Balancing  
Methods

What is the time complexity of rotation?

BSTs Recap

Balance

Balancing  
Operations

Rotations

Examples

Implementation

**Analysis**

Partition

Balancing  
Methods

What is the time complexity of rotation?

Answer:  $O(1)$

Rotation only requires a few localised pointer re-arrangements

BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

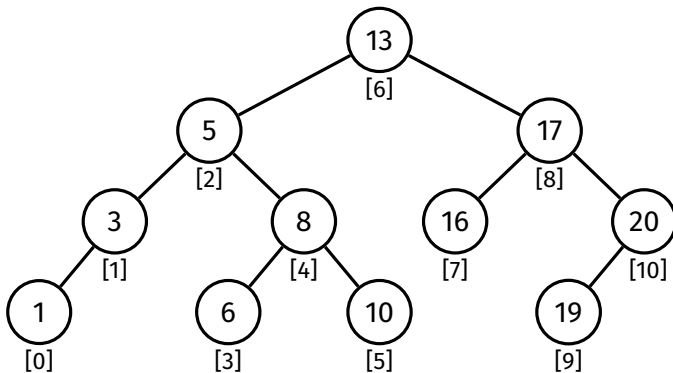
Example

Pseudocode

Analysis

Balancing  
Methods`partition(tree, i)`

Rearrange the tree so that the element with index  $i$  becomes the root



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

Example

Pseudocode

Analysis

Balancing  
Methods

## Method:

- Find element with index  $i$
- Perform rotations to lift it to the root

BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

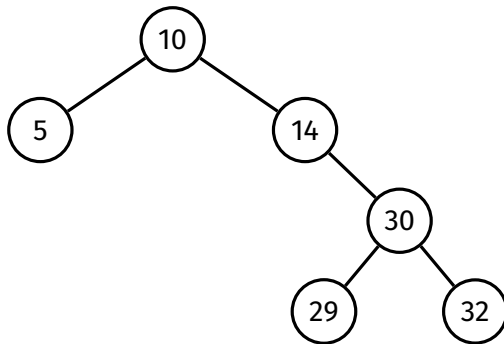
**Example**

Pseudocode

Analysis

Balancing  
Methods

Partition this tree around index 3:



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

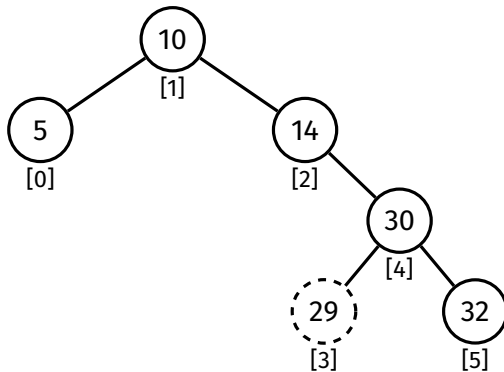
**Example**

Pseudocode

Analysis

Balancing  
Methods

Partition this tree around index 3:



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

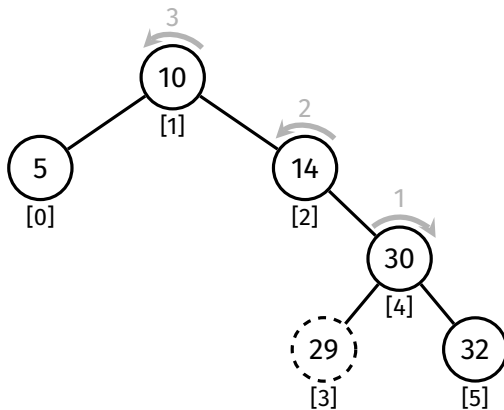
**Example**

Pseudocode

Analysis

Balancing  
Methods

Partition this tree around index 3:



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

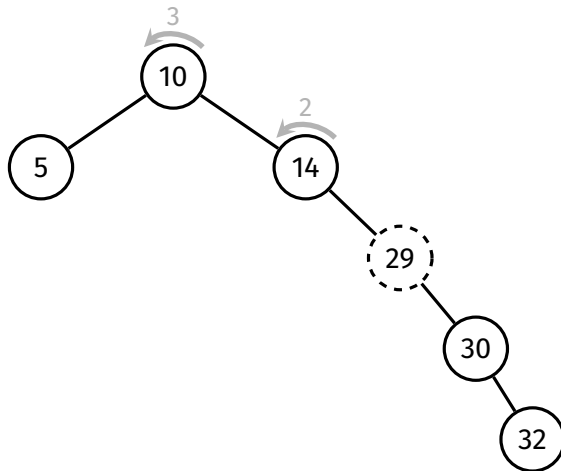
**Example**

Pseudocode

Analysis

Balancing  
Methods

After right rotation at 30:



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

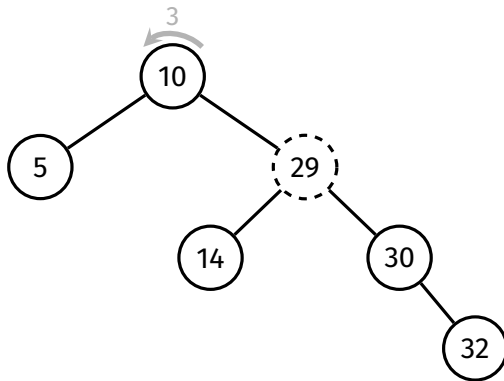
**Example**

Pseudocode

Analysis

Balancing  
Methods

After left rotation at 14:



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

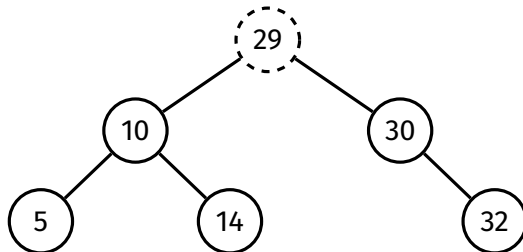
Example

Pseudocode

Analysis

Balancing  
Methods

After left rotation at 10:



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

Example

Pseudocode

Analysis

Balancing  
Methods

```
partition(t, i):
```

```
    Input: tree t, index i
```

```
    Output: tree with i-th item moved to root
```

```
    leftSize = size(t->left)
```

```
    if i < leftSize:
```

```
        t->left = partition(t->left, i)
```

```
        t = rotateRight(t)
```

```
    else if i > leftSize:
```

```
        t->right = partition(t->right, i - leftSize - 1)
```

```
        t = rotateLeft(t)
```

```
    return t
```

BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

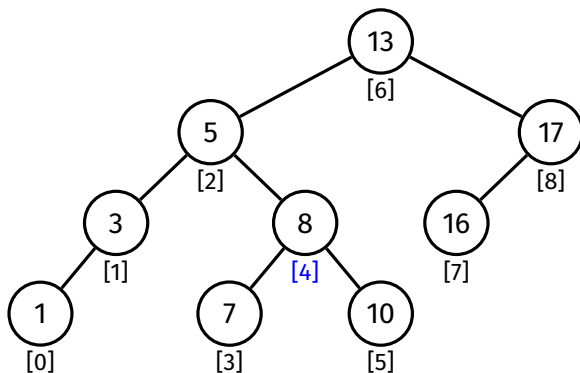
Example

Pseudocode

Analysis

Balancing  
Methods

Partition this tree around index 4



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

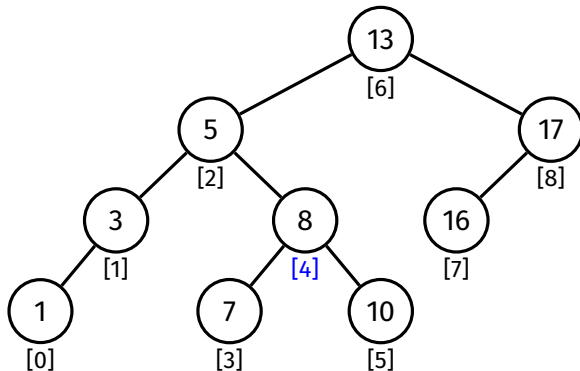
Example

Pseudocode

Analysis

Balancing  
Methods

Size of left subtree is 6, and  $4 < 6$ ...



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

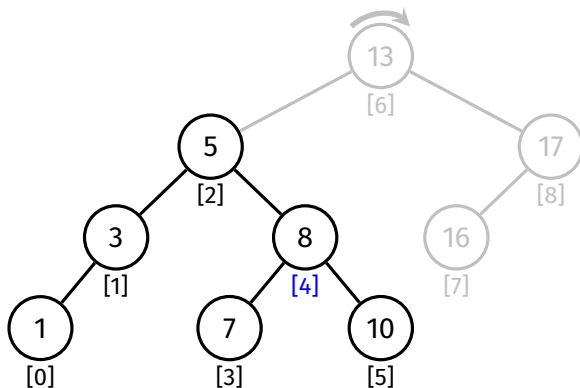
Example

Pseudocode

Analysis

Balancing  
Methods

Size of left subtree is 6, and  $4 < 6...$   
so partition left subtree around index 4  
and then rotate right at 13



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

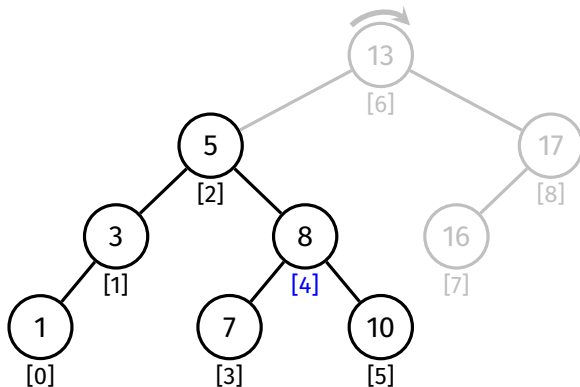
Example

Pseudocode

Analysis

Balancing  
Methods

Size of left subtree is 2, and  $4 > 2...$



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

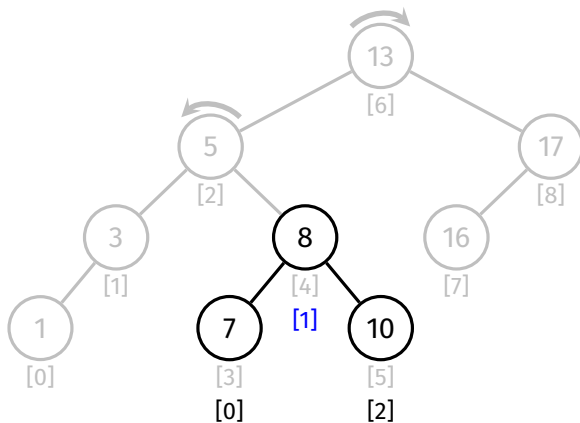
Example

Pseudocode

Analysis

Balancing  
Methods

Size of left subtree is 2, and  $4 > 2...$   
so partition right subtree around index  $(4 - 2 - 1 = 1)$   
and then rotate left at 5



BSTs Recap

Balance

Balancing  
Operations

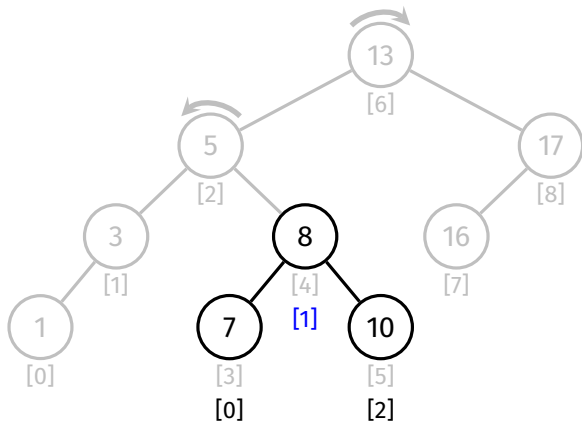
Rotations

Partition

Example

**Pseudocode**

Analysis

Balancing  
MethodsSize of left subtree is 1, and  $1 = 1...$ 

BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

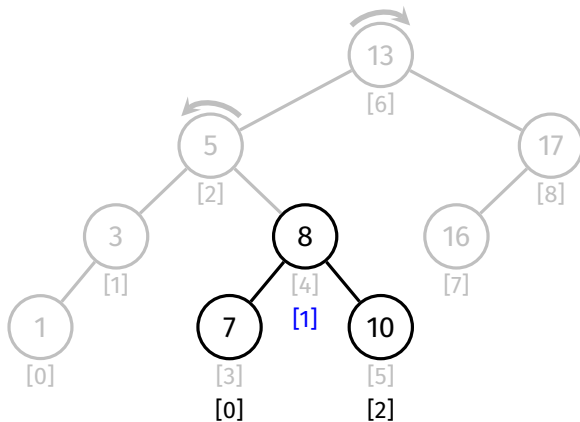
Example

Pseudocode

Analysis

Balancing  
Methods

Size of left subtree is 1, and  $1 = 1...$   
so we have found the desired node



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

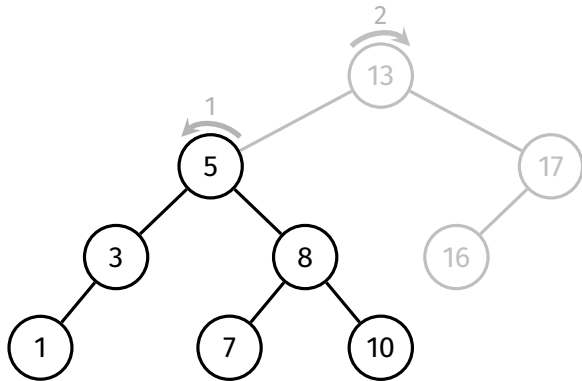
Example

Pseudocode

Analysis

Balancing  
Methods

Unwinding...  
Rotate left at 5



BSTs Recap

Balance

Balancing  
Operations

Rotations

Partition

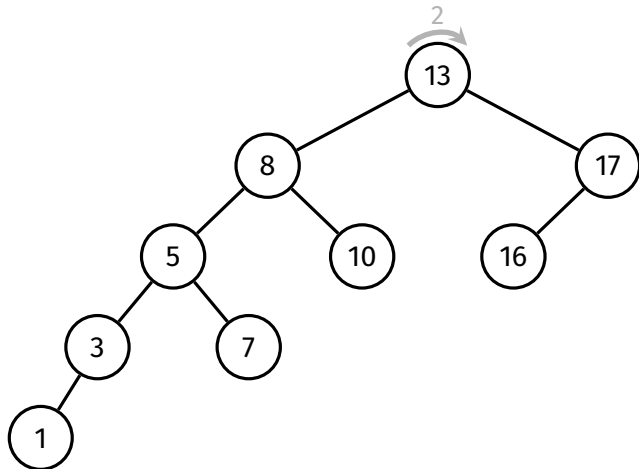
Example

Pseudocode

Analysis

Balancing  
Methods

Unwinding...  
Rotate right at 13



BSTs Recap

Balance

Balancing  
Operations

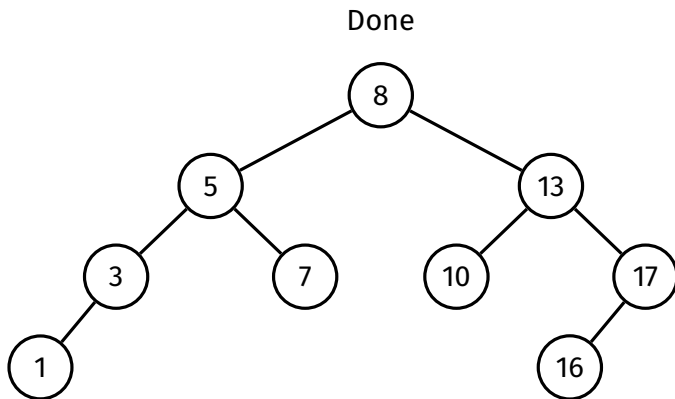
Rotations

Partition

Example

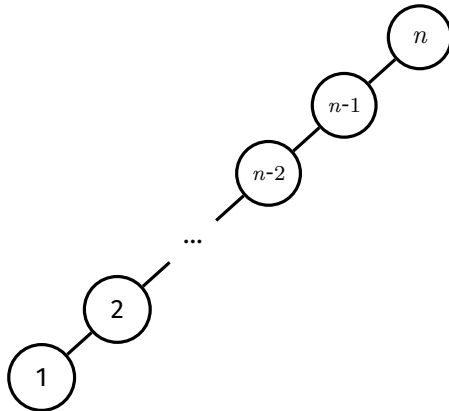
**Pseudocode**

Analysis

Balancing  
Methods

## Analysis:

- `size()` operation is expensive
- Can cause partition to be  $O(n^2)$  in the worst case
  - For example, in the following tree:



## Analysis (continued):

- To improve efficiency, can change node structure so that each node stores the size of its subtree in the node itself
  - This improves the worst case to  $O(n)$

```
struct node {  
    int item;  
    struct node *left;  
    struct node *right;  
    int size;  
};
```

BSTs Recap

Balance

Balancing  
Operations

Balancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

Two categories:

## GLOBAL REBALANCING

visit every node and balance its subtree;  
⇒ perfectly balanced tree — at cost.

## LOCAL REBALANCING

perform small, efficient, localised operations  
to try to improve the overall balance of the tree  
... at the cost of imperfect balance

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

**Idea:**

Completely rebalance whole tree so it is size-balanced

**Method:**

Lift the median node to the root  
by partitioning on index  $\text{SIZE}(t)/2$ ,  
then rebalance both subtrees (recursively)

BSTs Recap

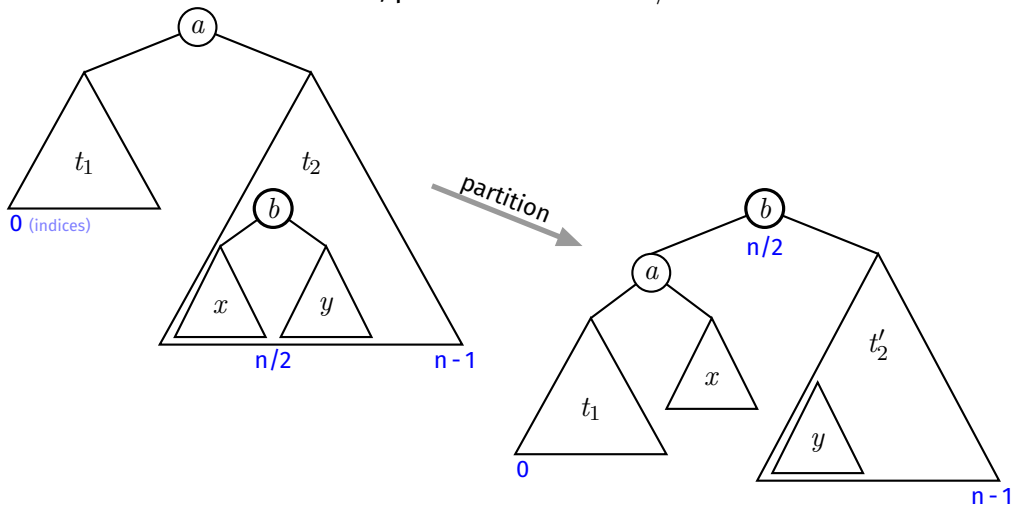
Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

First, partition on index  $n/2$ ...

...then rebalance both subtrees

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

```
rebalance( $t$ ):
```

```
    Input: tree  $t$ 
```

```
    Output: rebalanced  $t$ 
```

```
    if size( $t$ ) < 3:
```

```
        return  $t$ 
```

```
     $t$  = partition( $t$ , size( $t$ ) / 2)
```

```
     $t$ ->left = rebalance( $t$ ->left)
```

```
     $t$ ->right = rebalance( $t$ ->right)
```

```
    return  $t$ 
```

BSTs Recap

Balance

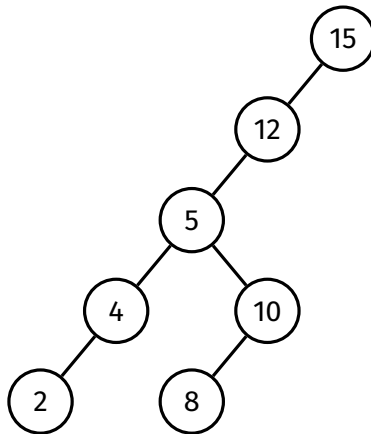
Balancing  
OperationsBalancing  
Methods

Global Rebalancing

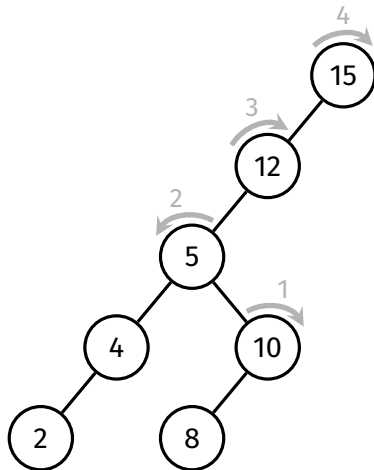
Local Rebalancing

Summary

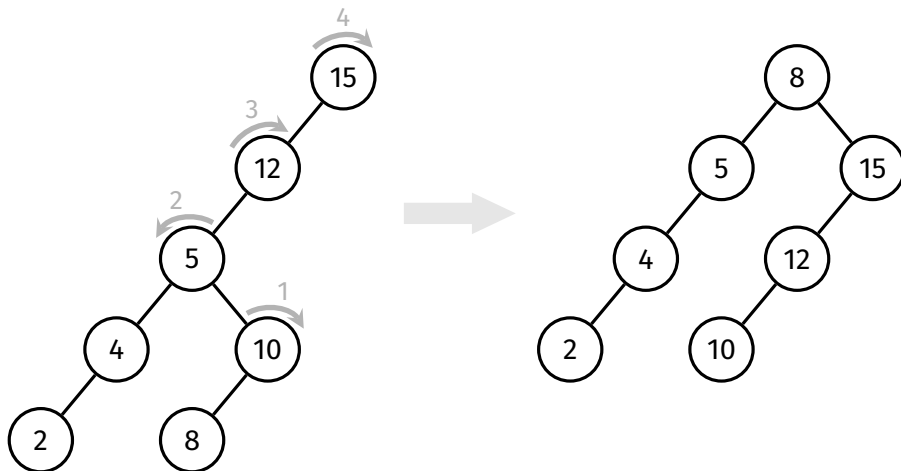
Rebalance the following tree:



First, partition the tree on index  $7/2 = 3$  (node 8)



First, partition the tree on index  $7/2 = 3$  (node 8)



BSTs Recap

Balance

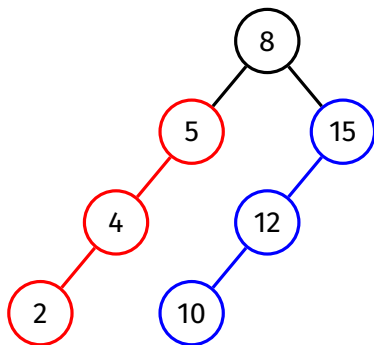
Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

Then, recursively rebalance subtrees



BSTs Recap

Balance

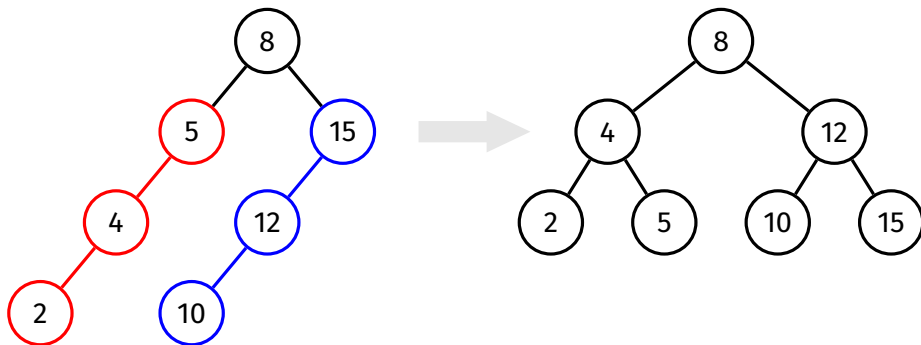
Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

Then, recursively rebalance subtrees



Worst-case time complexity:  $O(n \log n)$

- Assume nodes store the size of their subtrees
- First step: partition entire tree on index  $n/2$ 
  - This takes at most  $n$  recursive calls,  $n$  rotations  $\Rightarrow O(n)$
  - Result is two subtrees of size  $\approx n/2$
- Then partition both subtrees
  - This takes at most  $n/2$  recursive calls and rotations *each*  $\Rightarrow O(n)$
  - Result is four subtrees of size  $\approx n/4$
- ...and so on...
- About  $\log_2 n$  levels of partitioning in total, each taking  $O(n)$  time  
 $\Rightarrow O(n \log n)$

## What if we insert more items?

- Options:
  - Rebalance on every insertion
    - Not feasible
  - Rebalance every  $k$  insertions; what  $k$  is good?
  - Rebalance when imbalance exceeds threshold
- It's a tradeoff...
  - We either have more costly insertions
  - Or we have degraded performance for periods of time

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

```
bstInsert( $t$ ,  $v$ ):
```

```
    Input: tree  $t$ , value  $v$ 
```

```
    Output:  $t$  with  $v$  inserted
```

```
     $t$  = insertAtLeaf( $t$ ,  $v$ )
```

```
    if size( $t$ ) mod  $k$  = 0:  
         $t$  = rebalance( $t$ )
```

```
    return  $t$ 
```

BSTs Recap

Balance

Balancing  
Operations

Balancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

- Good if tree is not modified very often
- Otherwise...
  - Insertion will be slow occasionally due to rebalancing
  - Performance will gradually degrade until next rebalance

BSTs Recap

Balance

Balancing  
Operations

Balancing  
Methods

Global Rebalancing

Local Rebalancing

Root Insertion

Randomised  
Insertion

Summary

Perform small, efficient, localised operations  
in an attempt to improve the overall balance of the tree

1. root insertion

2. randomised insertion

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Root Insertion

Randomised

Insertion

Summary

**Idea:**

Rotations change the structure of a tree

If we perform some rotations every time we insert,  
that may restructure the tree sufficiently enough  
such that it is more balanced

One systematic way to perform these rotations:  
Insert new values at the root

BSTs Recap

Balance

Balancing  
Operations

Balancing  
Methods

Global Rebalancing

Local Rebalancing

**Root Insertion**

Randomised

Insertion

Summary

## Method:

Insert new value normally (at the leaf) ...  
... and then rotate the new node up to the root.

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

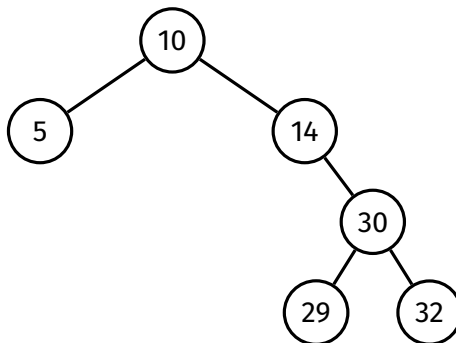
Local Rebalancing

Root Insertion

Randomised  
Insertion

Summary

Insert 24 at the root of this tree:



BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

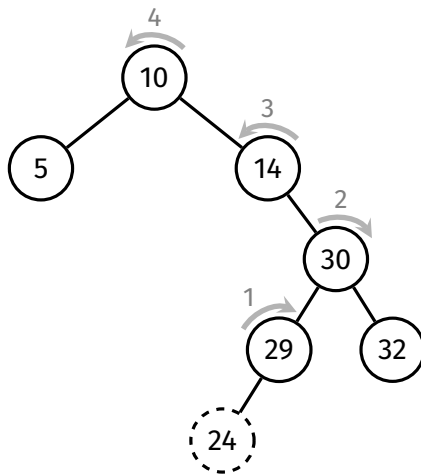
Root Insertion

Randomised

Insertion

Summary

Insert 24 at the root of this tree:



BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

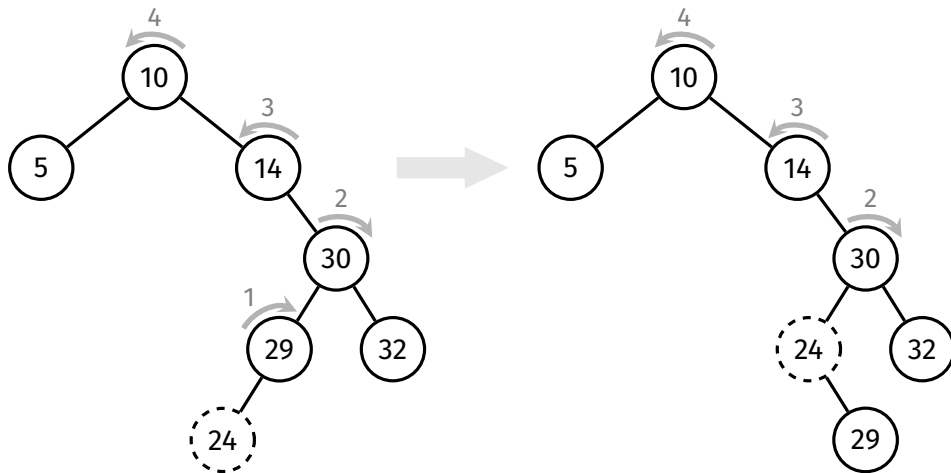
Root Insertion

Randomised

Insertion

Summary

Rotate right at 29



BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

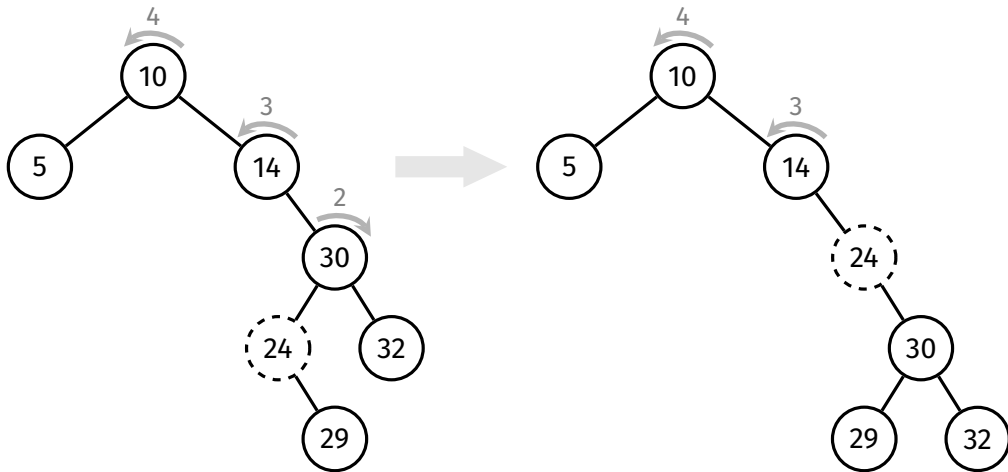
Root Insertion

Randomised

Insertion

Summary

Rotate right at 30



BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

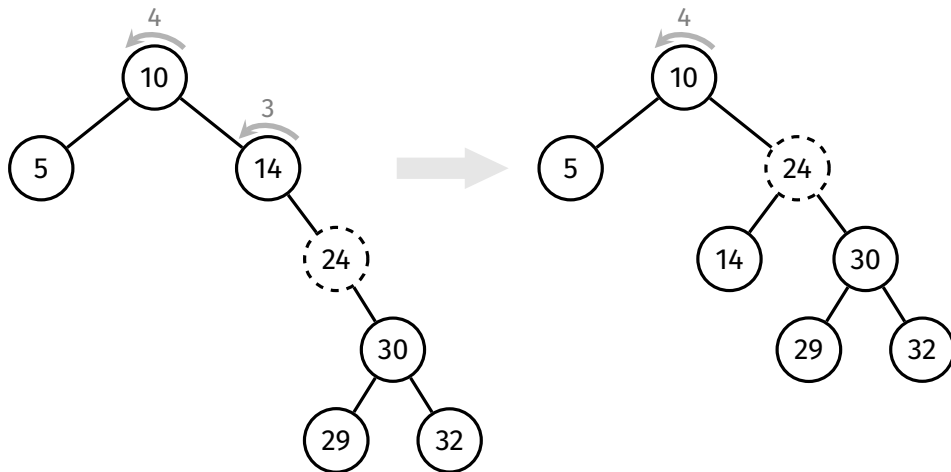
Root Insertion

Randomised

Insertion

Summary

Rotate left at 14



BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

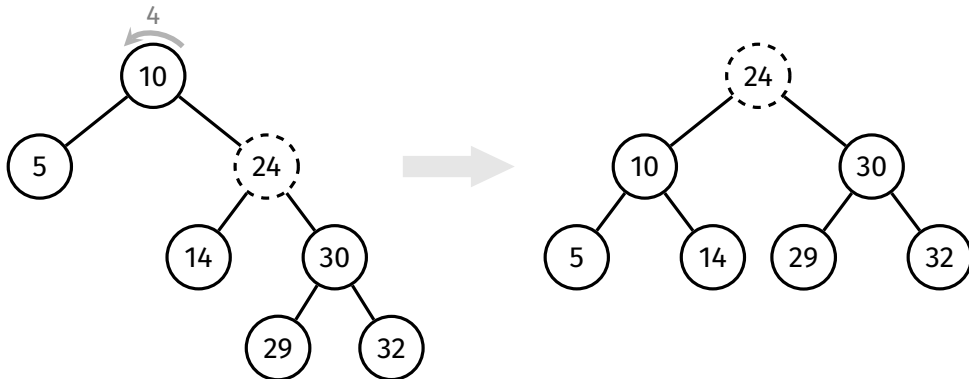
Root Insertion

Randomised

Insertion

Summary

Rotate left at 10



```
insertAtRoot(t, v):
```

```
    Input: tree t, value v
```

```
    Output: t with v inserted at the root
```

```
    if t is empty:
```

```
        return new node containing v
```

```
    else if  $v < t \rightarrow \text{item}$ :
```

```
         $t \rightarrow \text{left} = \text{insertAtRoot}(t \rightarrow \text{left}, v)$ 
```

```
         $t = \text{rotateRight}(t)$ 
```

```
    else if  $v > t \rightarrow \text{item}$ :
```

```
         $t \rightarrow \text{right} = \text{insertAtRoot}(t \rightarrow \text{right}, v)$ 
```

```
         $t = \text{rotateLeft}(t)$ 
```

```
    return t
```

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Root Insertion

Randomised

Insertion

Summary

## Analysis:

- Same time complexity as normal insertion:  $O(h)$
- Tree is more likely to be balanced, but no guarantee
- Root insertion ensures recently inserted items are close to the root
  - Useful for applications where recently added items are more likely to be searched
- Major problem: ascending-ordered and descending-ordered data is still a worst case for root insertion

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Root Insertion

Randomised  
Insertion

Summary

BSTs don't have control over insertion order.  
Worst cases — (partially) ordered data — are common.

**Idea:**

Introduce some randomness into insertion algorithm:  
Randomly choose whether to insert normally or insert at root

```
insertRandom( $t$ ,  $v$ ):
```

```
    Input: tree  $t$ , value  $v$ 
```

```
    Output:  $t$  with  $v$  inserted
```

```
    if  $t$  is empty:
```

```
        return new node containing  $v$ 
```

```
    // p/q chance of inserting at root
```

```
    if random() mod  $q < p$ :
```

```
        return insertAtRoot( $t$ ,  $v$ )
```

```
    else:
```

```
        return insertAtLeaf( $t$ ,  $v$ )
```

Note: random() is a pseudo-random number generator  
30% chance of root insertion  $\Rightarrow$  choose  $p = 3$ ,  $q = 10$

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Root Insertion

Randomised  
Insertion

Summary

Randomised insertion creates similar results to  
inserting items in random order.

Tree is more likely to be balanced (but no guarantee)

BSTs Recap

Balance

Balancing  
OperationsBalancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

|                    | Advantages                                                    | Disadvantages                                                                  |
|--------------------|---------------------------------------------------------------|--------------------------------------------------------------------------------|
| Global rebalancing | Guarantees a balanced tree                                    | Inefficient ( $O(n \log n)$ per rebalance), or periods of degraded performance |
| Local rebalancing  | Efficient (adds only a constant factor overhead to insertion) | Not guaranteed to produce a balanced tree                                      |

BSTs Recap

Balance

Balancing  
Operations

Balancing  
Methods

Global Rebalancing

Local Rebalancing

Summary

<https://forms.office.com/r/ucsJ4r99Ag>

