

Lambda Functions

Data Structures & Balanced Binary Search Trees

CS 225

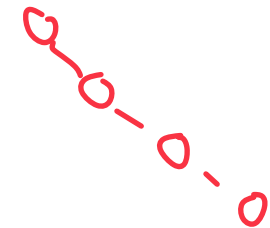
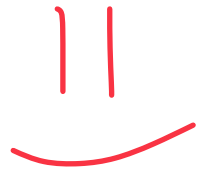
Brad Solomon

September 28, 2026

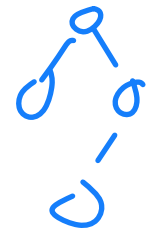


UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



vs



Exam 2 (9/30 — 10/02)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be released on PL

Topics covered can be found on website

Register now

<https://courses.engr.illinois.edu/cs225/exams/>

Additional Extra Credit / Research Opportunity

Preventing False Positives in Plagiarism Detection

Informed Consent Agreement on Prairielearn

2 EC points for filling out survey (regardless of consent)

Learning Objectives

Lambda Functions 101 (and other MP Mosaic tips)

Briefly review BST in the context of height

Discuss the big picture problem with BSTs

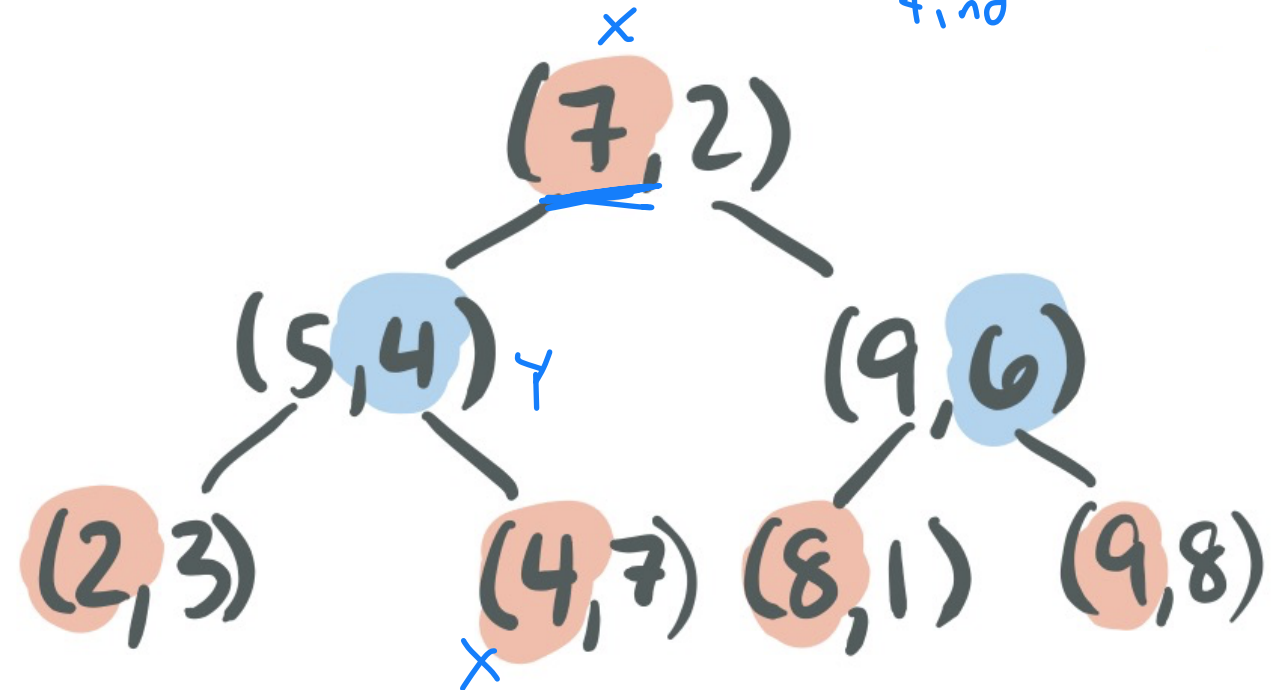
Introduce the self-balancing BST

QuickSelect in a KD Tree

How do we use a 1D algorithm in a k-dimension tree constructor?

Pass a comparator function as an input parameter

find



Hint: **Lambda functions!**

Functions as arguments

Consider the function from Excel
`COUNTIF(range, criteria)`

CountIf (*A1:A9* , *"<0"*)
 ↑ ↑ ↑
 iterator iterator Some
 begin end function

↓↑

A10			
	✕	✓	<i>fx</i> =COUNTIF(A1:A9,"<0")
	A	B	C
1	1		
2	102		
3	105		
4	4		
5	5		
6	27		
7	41		
8	-7		
9	999		
10	1		
11			

Functions as arguments

How do we know inputs/output
for this function? You don't!

Countif.hpp

```
10 template <typename Iter, typename Pred>
11 int Countif(Iter begin, Iter end, Pred pred) {
12     int count = 0;
13     auto cur = begin;
14
15     while(cur != end) {
16         if(pred(*cur))
17             ++count;
18         ++cur;
19     }
20
21     return count;
22 }
```

Comparator

three (?) slides from now is Big (the lambda function)

Lambda Functions in C++

Here are several ways to write a function as an object

main.cpp

```
1 bool isNegative(int num) { return (num < 0); } ← function
2
3 class IsNegative {
4 public:
5     bool operator() (int num) { return (num < 0); }
6 }; Can have class variables } functor
7
8 int main() {
9     std::vector<int> numbers = {1, 102, 105, 4, 5, 27, 41, -7, 999};
10
11     [auto isnegl = [](int num) { return (num < 0); };] lambda function
12     auto isnegfp = isNegative;
13     auto isnegfunctor = IsNegative();
14
15     cout << "There are " << Countif(numbers.begin(), numbers.end(), _____)
16         << " negative numbers" << std::endl;
17
```

Are very similar

Lambda Functions in C++

An inline function without a name

[Capture](Arg List) { Function Body }

int c = 2;
auto f = [c] (int x) { return x % c == 0; }

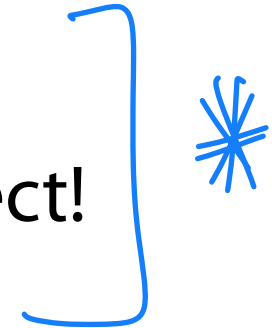
int c = 10;
auto divideBy10 = [c] (int x) { }

Lambda Functions in C++

An inline function without a name

[Capture](Arg List){ Function Body }

Capture: Takes the value of object based on when the lambda was defined, NOT the current value of the object!



Arg List: Standard way of inputting into a function

Function Body: Code can use both capture vars and arg vars

Lambda Functions in C++



main.cpp

```
29  int big;
30  std::cout << "How big is big? ";
31  std::cin >> big;
32
33  auto isbig = [big](int num) { return (num >= big); };
34
35  // 5
36  // big = 999
37  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)
38  << " big numbers" << std::endl;
}
```

big is captured at specific value
↳ "isbig" will always have big = 5

↳ The arg (int num) is not filled in until used

isbig (number [0])
↑

Lambda Functions in C++



main.cpp

isbig(x)

```
29  int big;
30  std::cout << "How big is big? ";
31  std::cin >> big;
32
33  auto isbig = [big](int num) { return (num >= big); };
34
35
36
37  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)
38  << " big numbers" << std::endl;
}
```

↑
arg list

Useful for mp_mosaics!

KD-Tree will split points in one dimension

When comparing, we need to remember what dimension we are in!

↙ ↘
BST * (in one dim)

Final Tips

The mp_mosaic writeup / documentation is long. **READ IT**

The suggestions in the writeup should be followed carefully

```
1  template <typename RandIter, typename Comparator>
2  void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
3  {
4      /**
5       * @todo Implement this function!
6       */
7
8  }
9
```

Understanding 'randIter'

An iterator is a container giving access in different ways:

Forward

Bidirectional

Random Access



Implementing quickselect with RandIter

Random Access Iterator lets you:

Swap items using `std::swap()`

```
1  template <typename RandIter, typename Comparator>
2  void BlackBox(RandIter A, RandIter B)
3  {
4
5      std::swap(*A, *B);
6
7
8  }
9
```

Hint: We just covered when you might want to swap in quick select!

Implementing quickselect with RandIter

Random Access Iterator lets you:

Access container indices using math operations

```
randIter A;
```

```
auto nth = *(A + n);
```

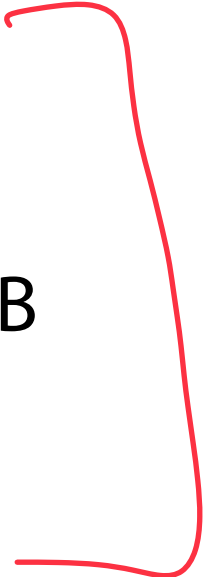


Get distance between two iterators

```
randIter A, B;
```

```
A < B;           // True if A is earlier in container than B
```

```
A - B;         // The distance between A and B
```



Implementing quickselect with RandIter

Random Access Iterator lets you:

Do most things you'd expect an array to be able to do!

The power of the **Interface!**



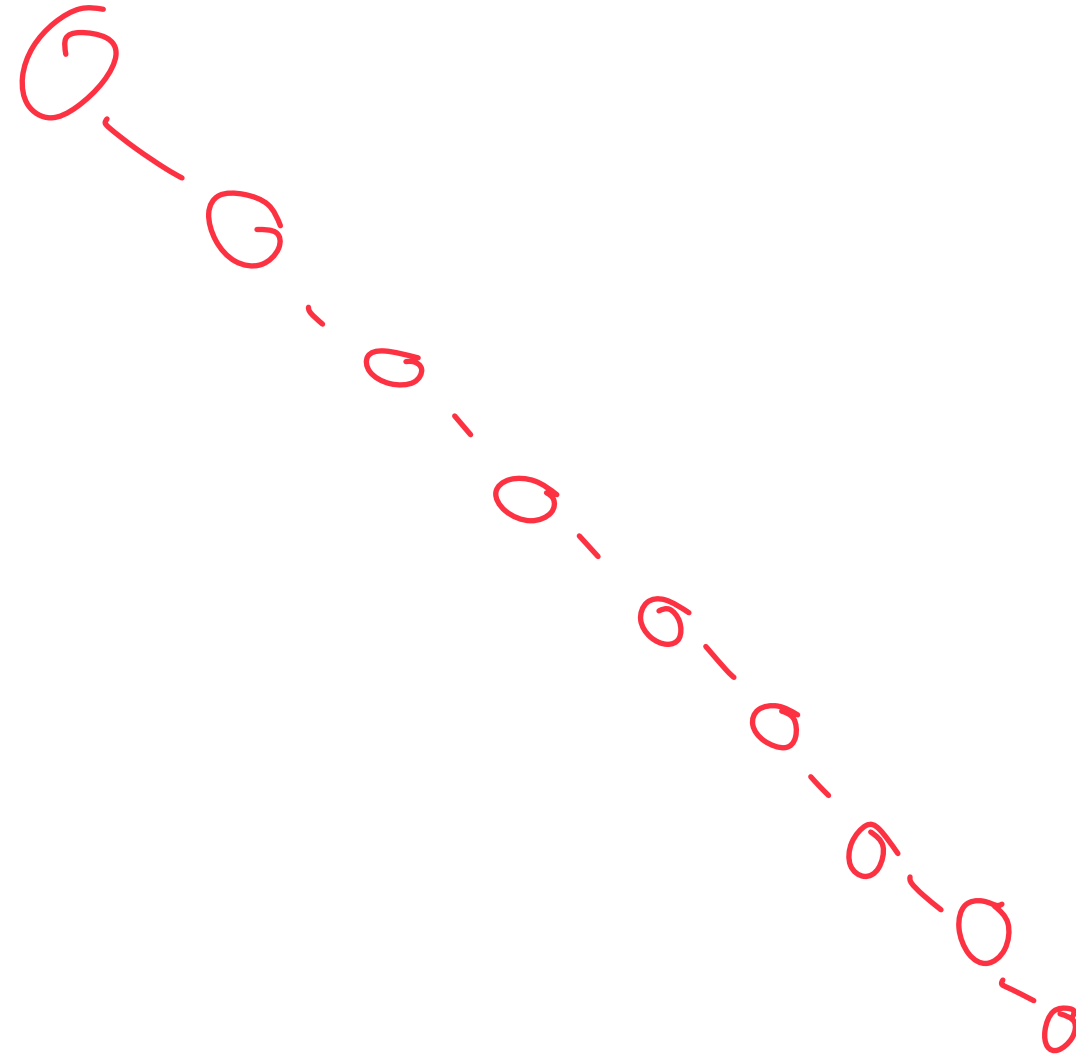
https://en.cppreference.com/w/cpp/iterator/random_access_iterator



Lets go back to core content...

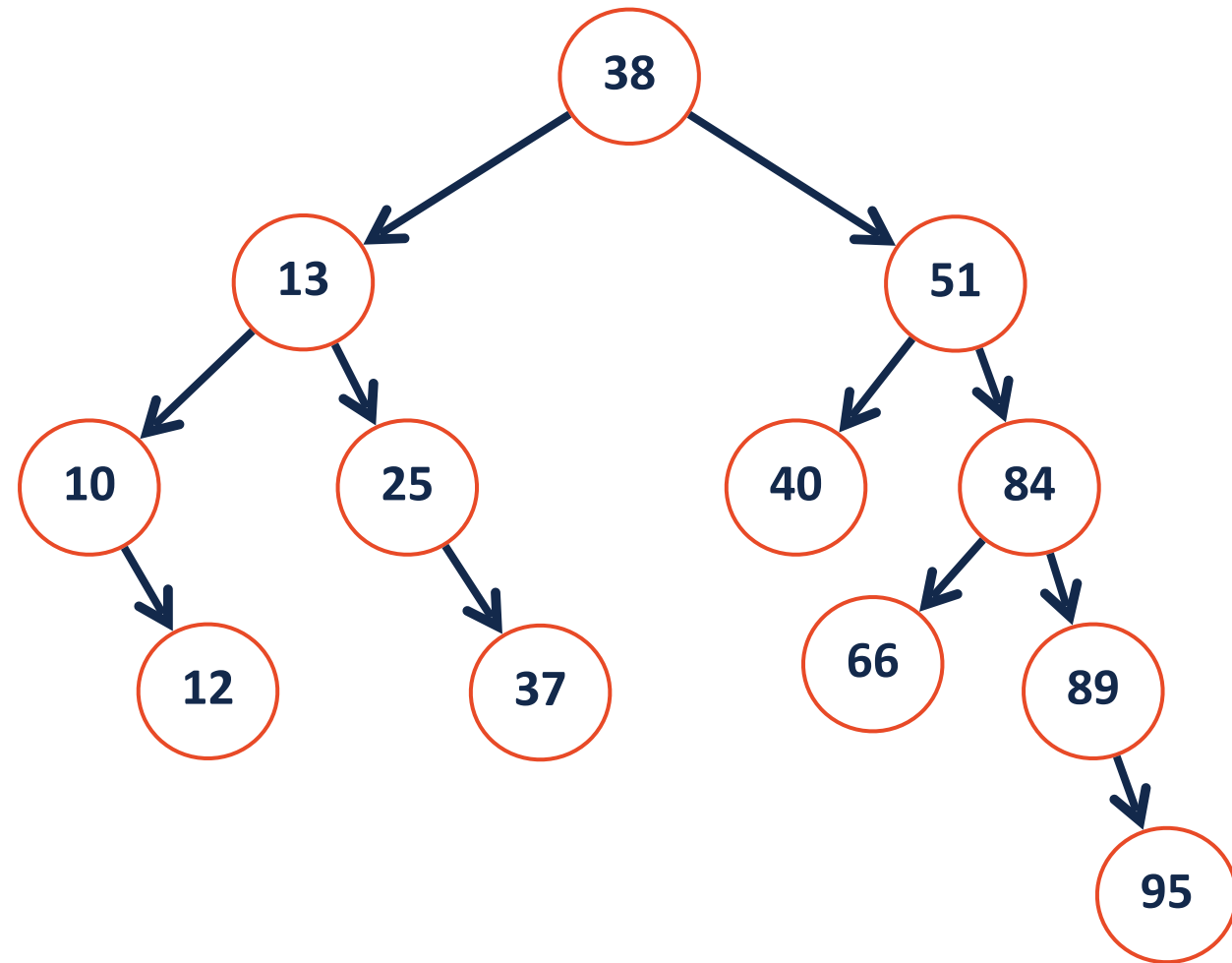
BST Analysis – Running Time

Operation	BST Worst Case
find	$O(h)$
insert	$O(h)$
remove	$O(h)$
traverse	$O(n)$



BST Analysis – Running Time

Operation	BST Worst Case
find	$O(h)$
insert	$O(h)$
remove	$O(h)$
traverse	$O(n)$



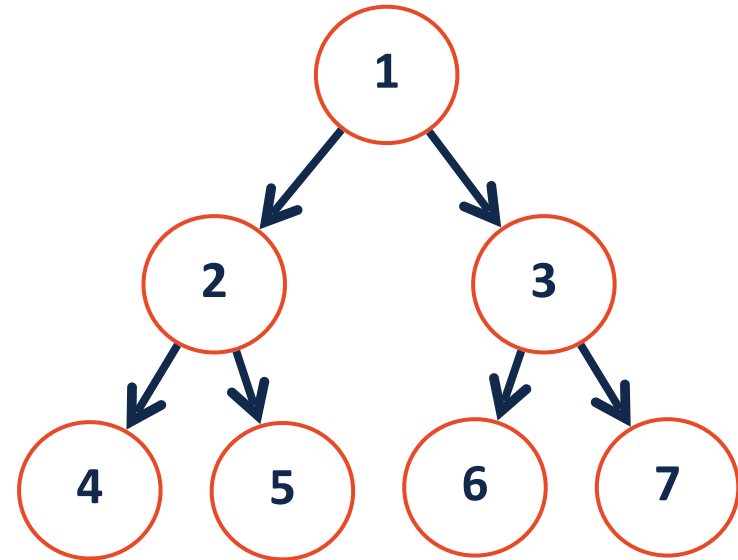
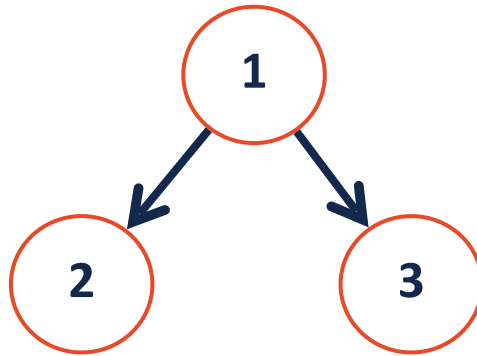
BST Analysis

Every operation on a BST depends on the **height** of the tree.

... how do we relate $O(h)$ to n , the size of our dataset?

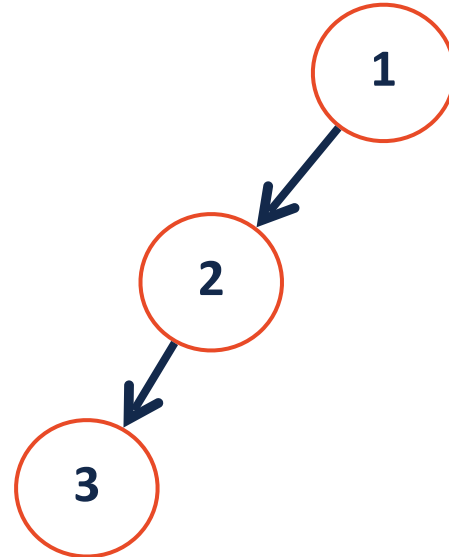
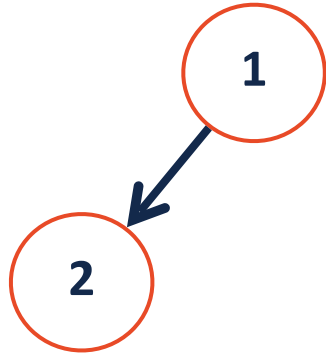
BST Analysis

What is the **max** number of nodes in a tree of height h ?



BST Analysis

What is the **min** number of nodes in a tree of height h ?

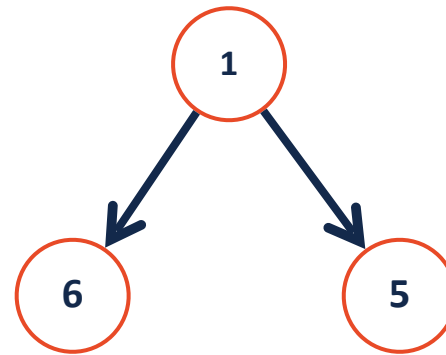


BST Analysis



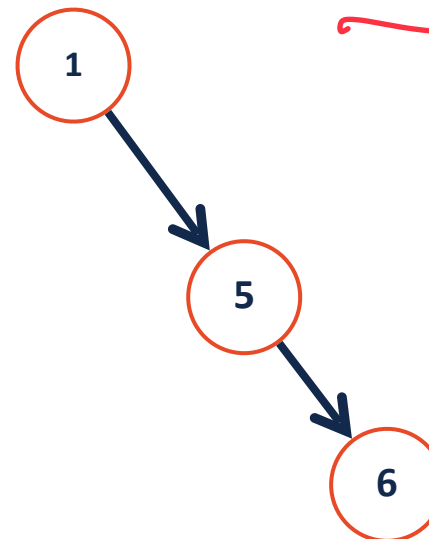
A BST of n nodes has a height between:

Lower-bound: $O(\log n)$



$$n = 2^{\log(h+1)} - 1$$
$$\log n \sim O(h)$$

Upper-bound: $O(n)$

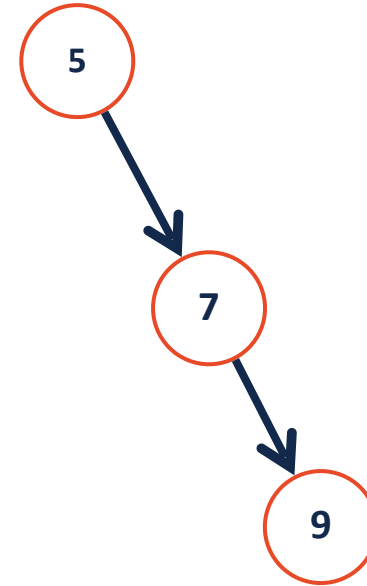
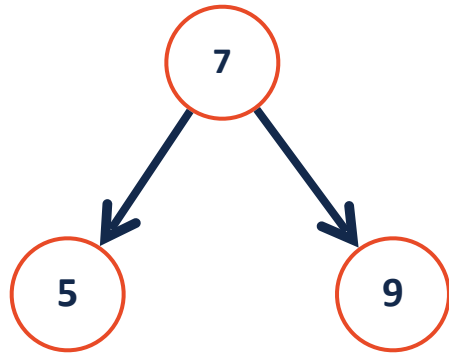


$$h = n$$

More like
TQ's

Height-Balanced Tree

Balanced trees enforce better bounds on height!



Height balance: $b = \text{height}(T_R) - \text{height}(T_L)$

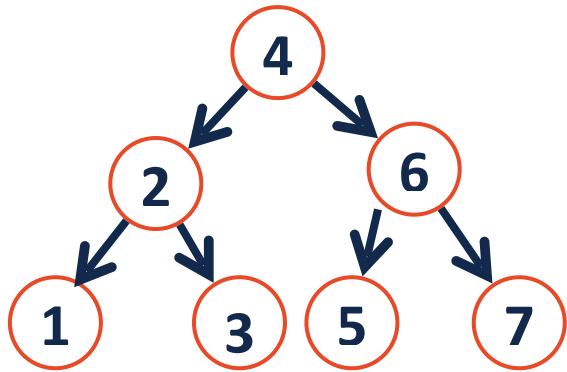
A tree is "balanced" if $|b| \leq 1$

Height-Balanced Tree

Join Code: 225



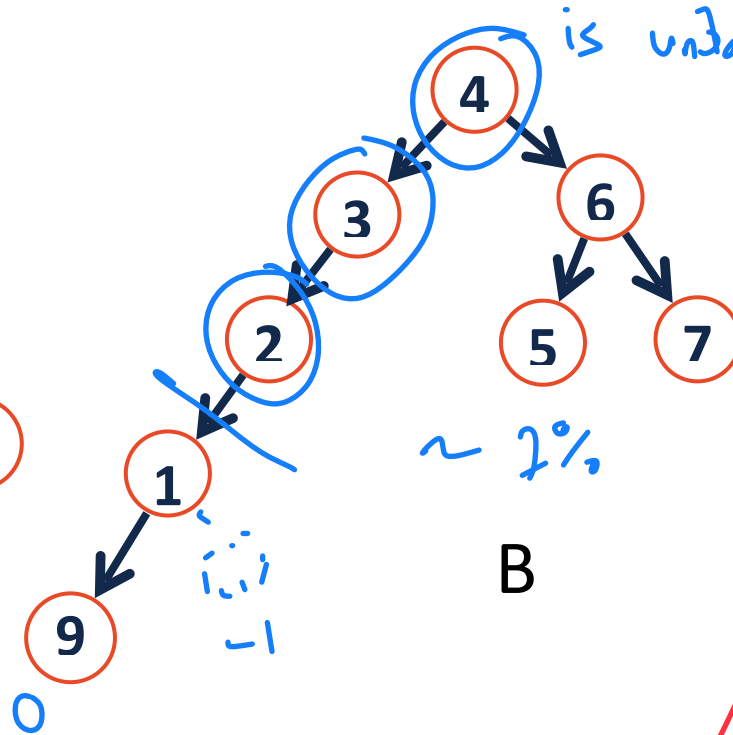
Which of the following binary trees are balanced?



100%

A

is balanced

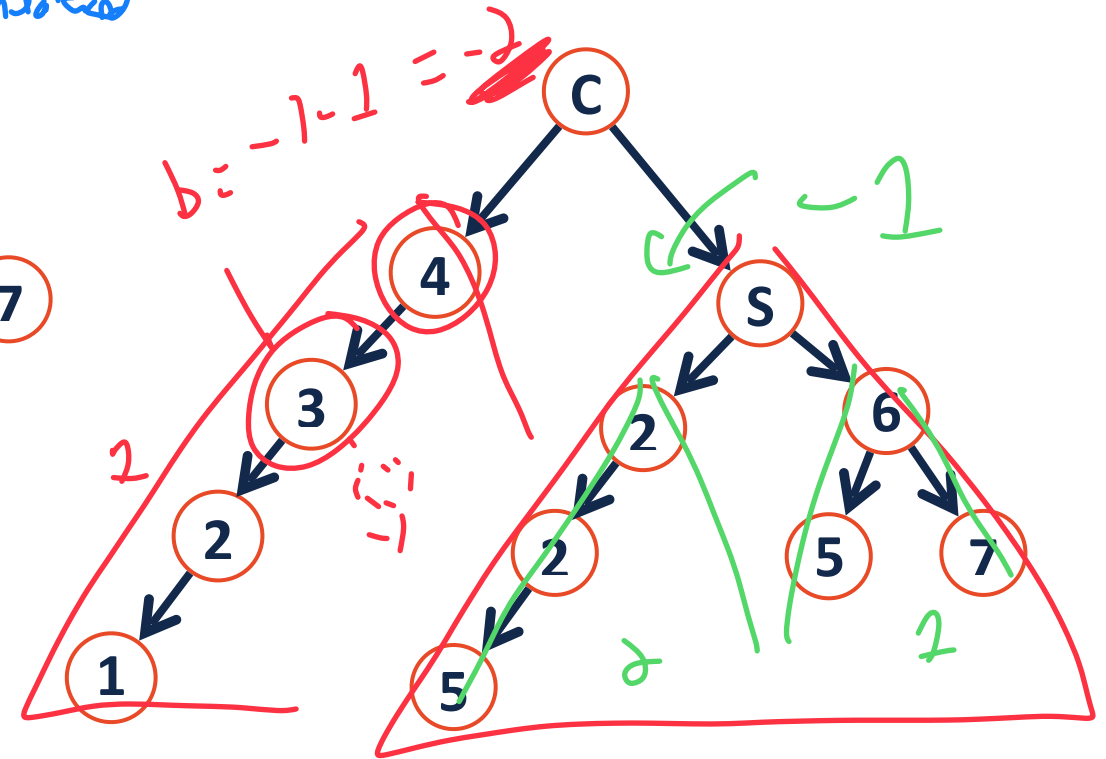


~ 2%

B

$-1 - 0 = -1$

Not balanced



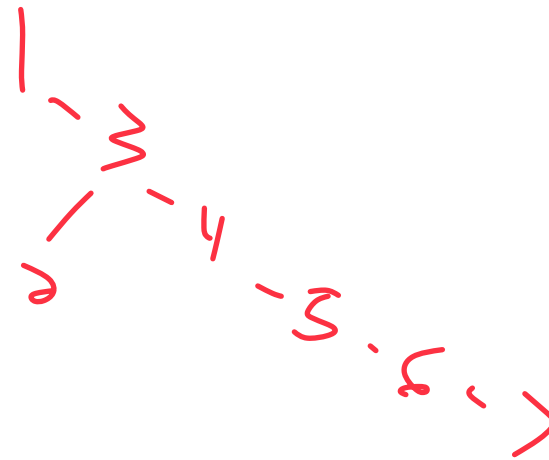
C 60%

Not balanced!

Option A: Correcting bad insert order → lab bst

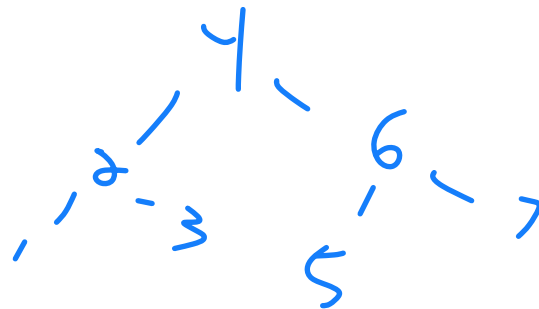
The height of a BST depends on the order in which the data was inserted

Insert Order: [1, 3, 2, 4, 5, 6, 7]



Almost worst
tree

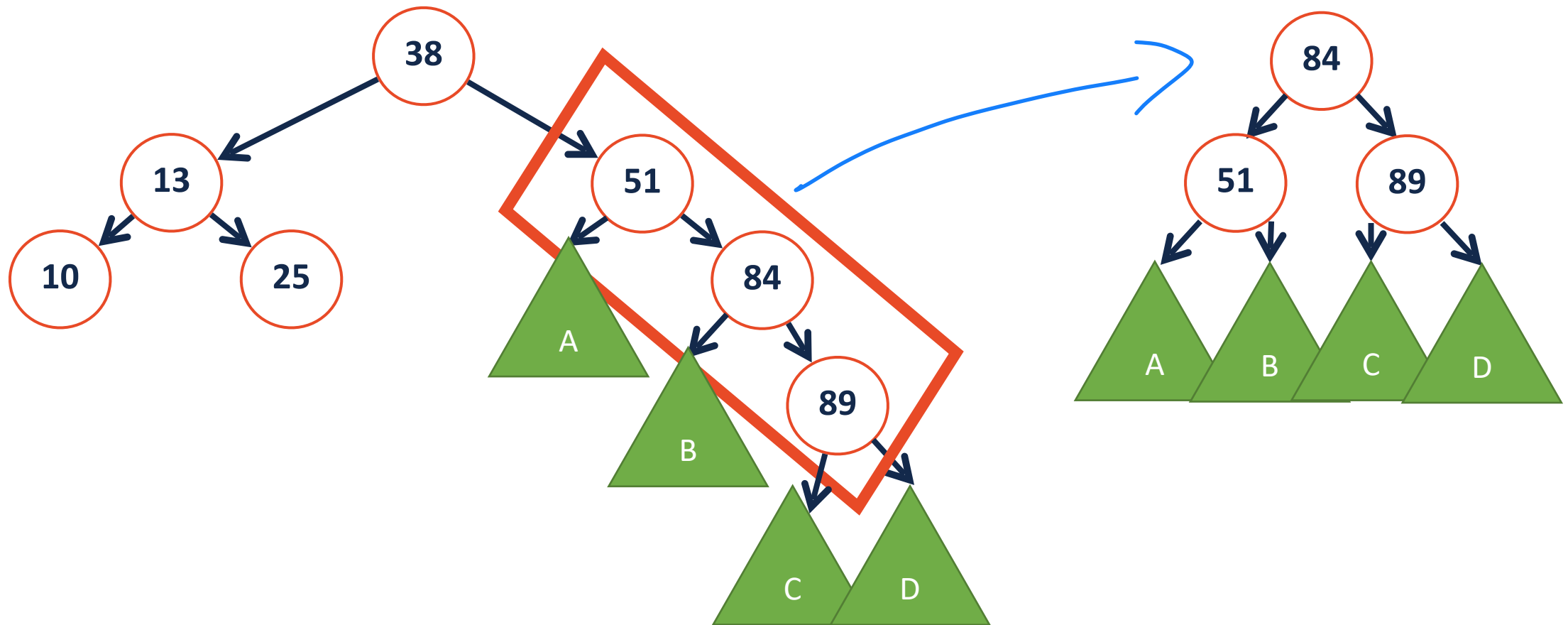
Insert Order: [4, 2, 3, 6, 7, 1, 5]



Best

AVL-Tree: A self-balancing binary search tree

Rather than fixing an insertion order, just correct the tree as needed!



BST Rotations (The AVL Tree)

We can adjust the BST structure by performing **rotations**.



These rotations, when used correctly:

- 1.

- 2.

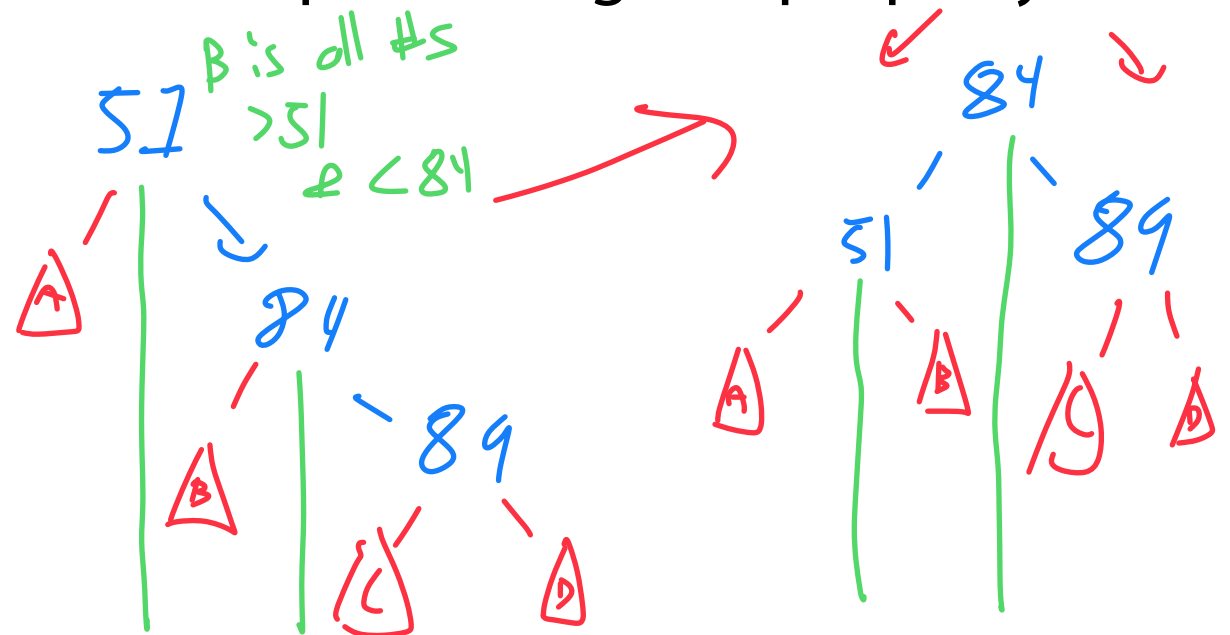
BST Rotations (The AVL Tree)

We can adjust the BST structure by performing **rotations**.

These rotations, when used correctly:

1. Modify the arrangement of nodes while preserving BST property

2.



BST Rotations (The AVL Tree)

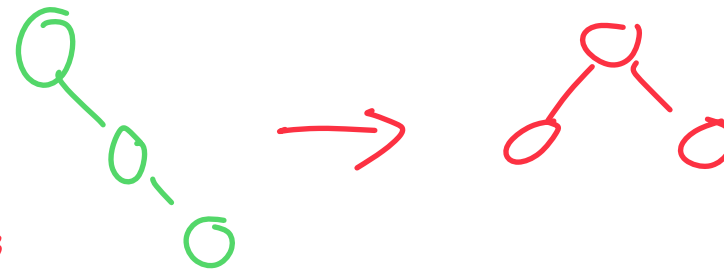
We can adjust the BST structure by performing **rotations**.

These rotations, when used correctly:

1. Modify the arrangement of nodes while preserving BST property

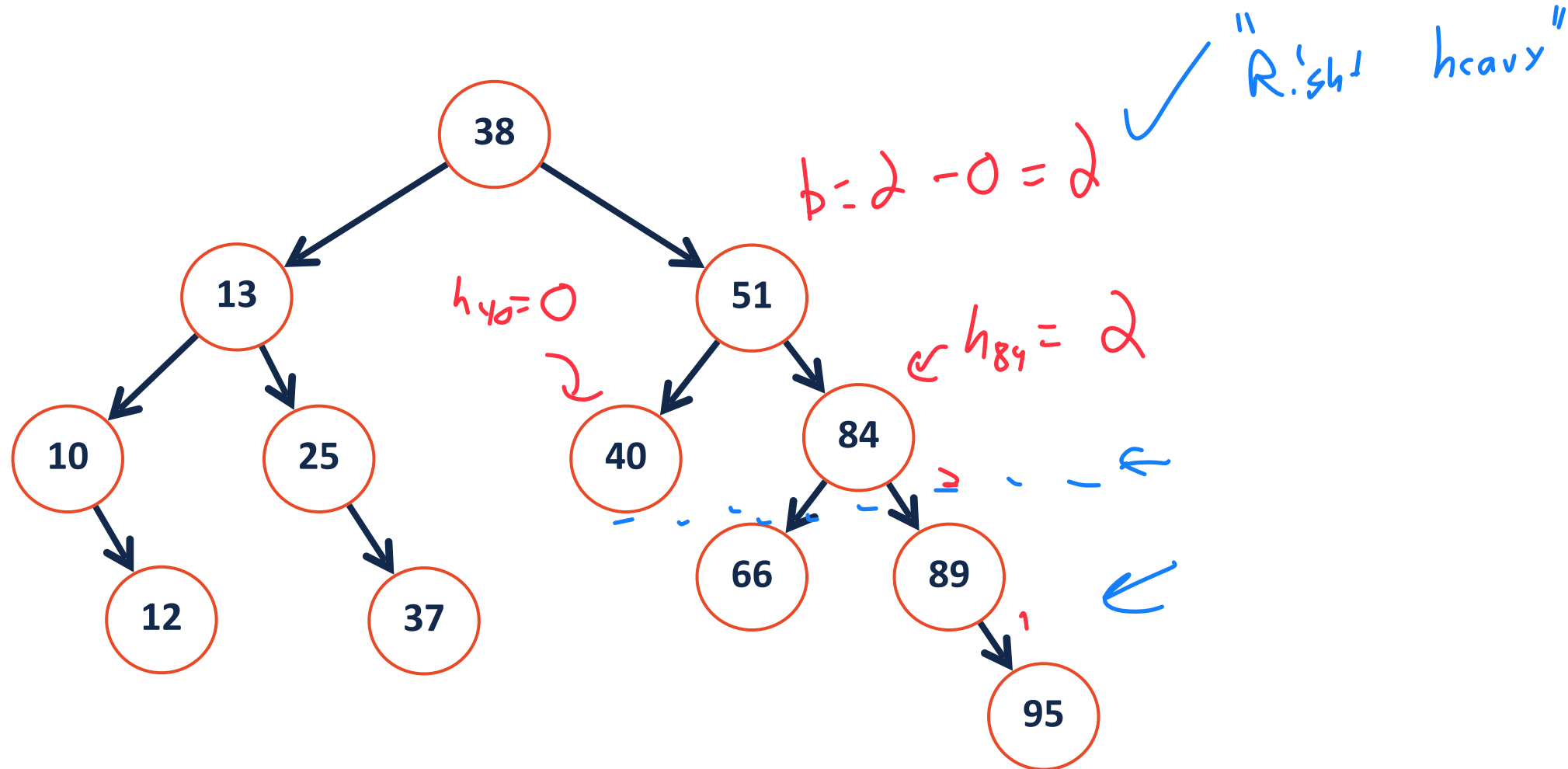
2. Reduce tree height by one

↳ This is a theory thing

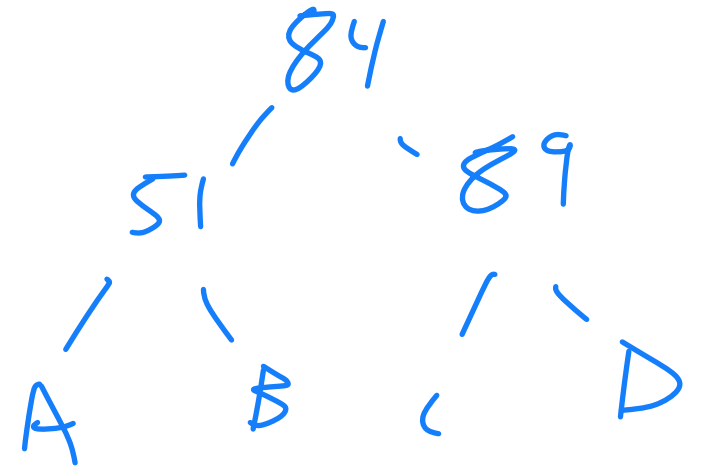
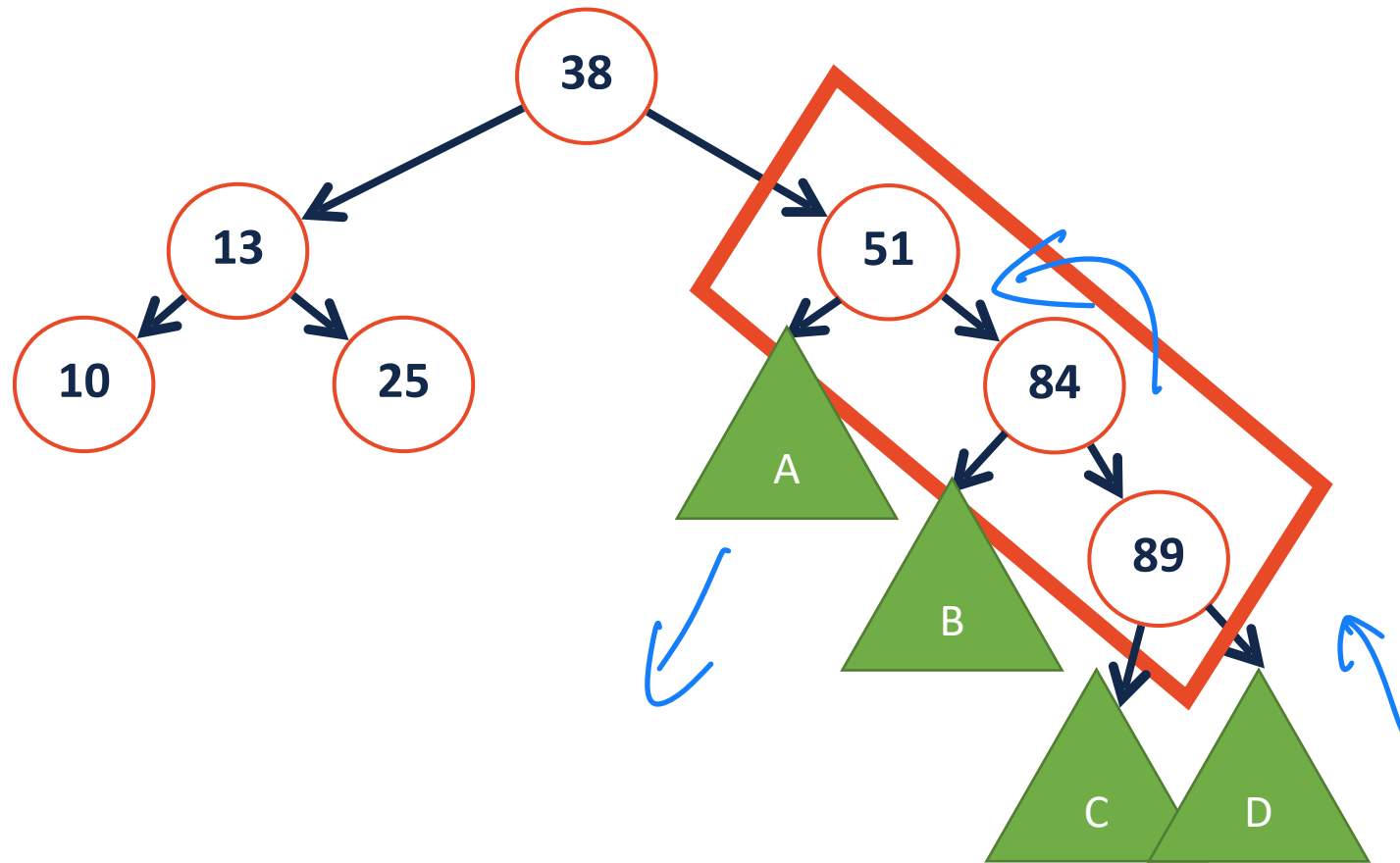


BST Rotations (The AVL Tree)

To begin, let's find the imbalance in the following tree:

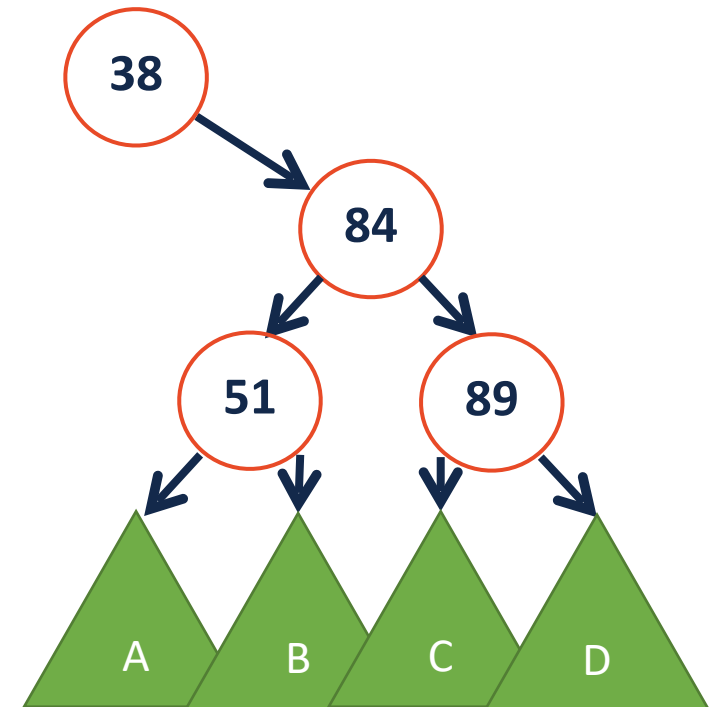
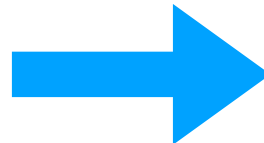
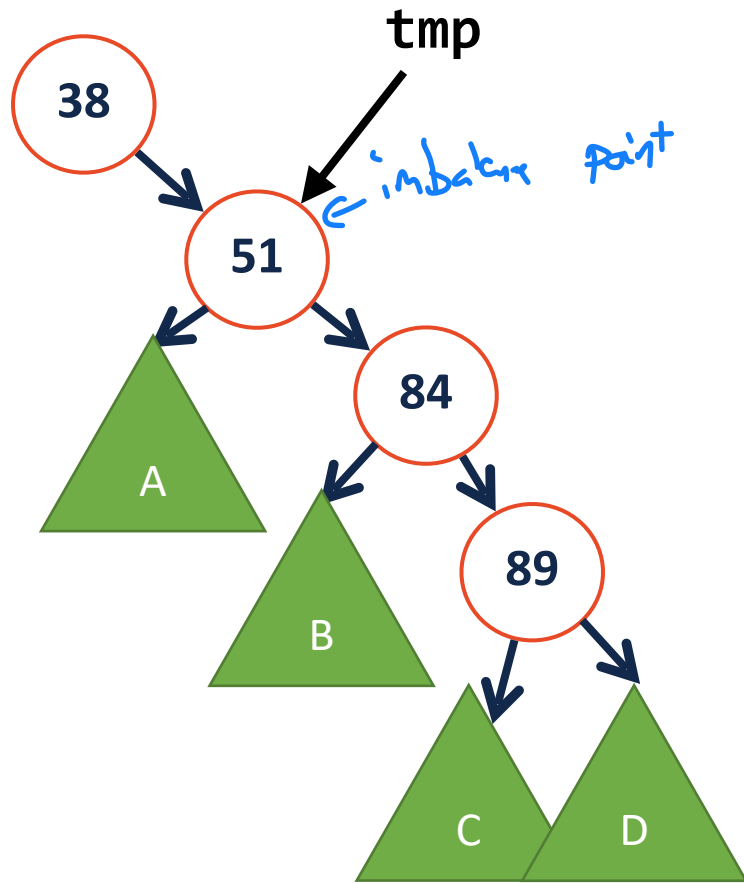


Left Rotation



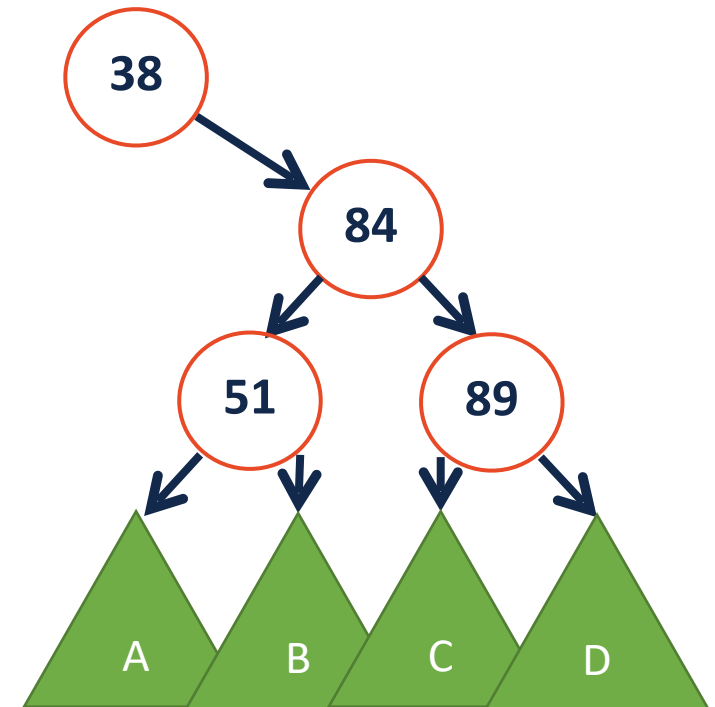
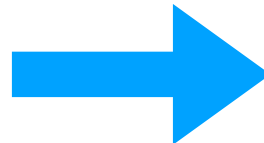
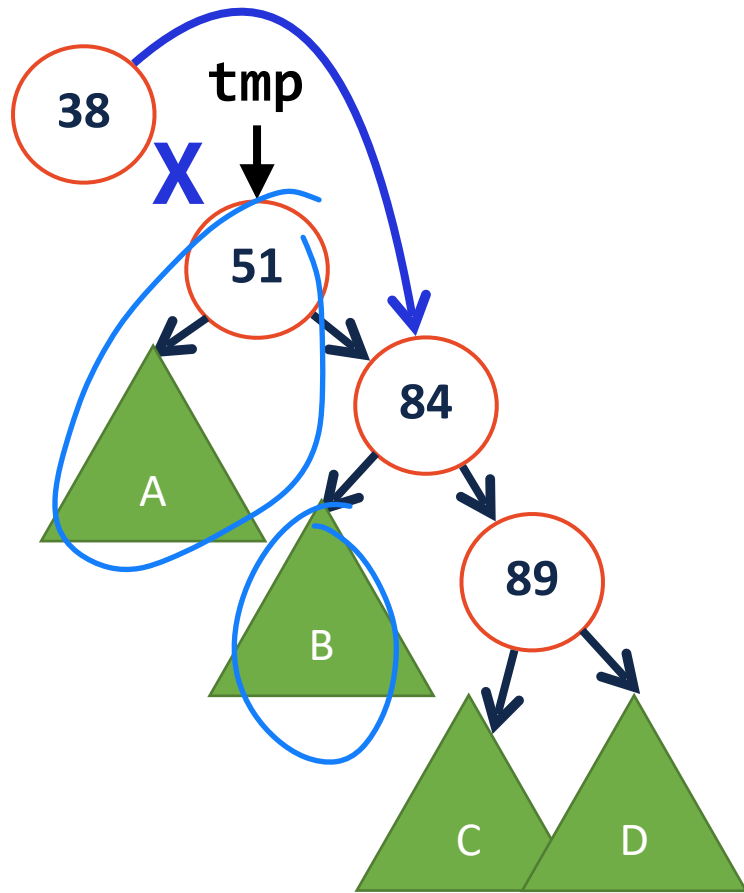
Left Rotation

1) Create a tmp pointer to root



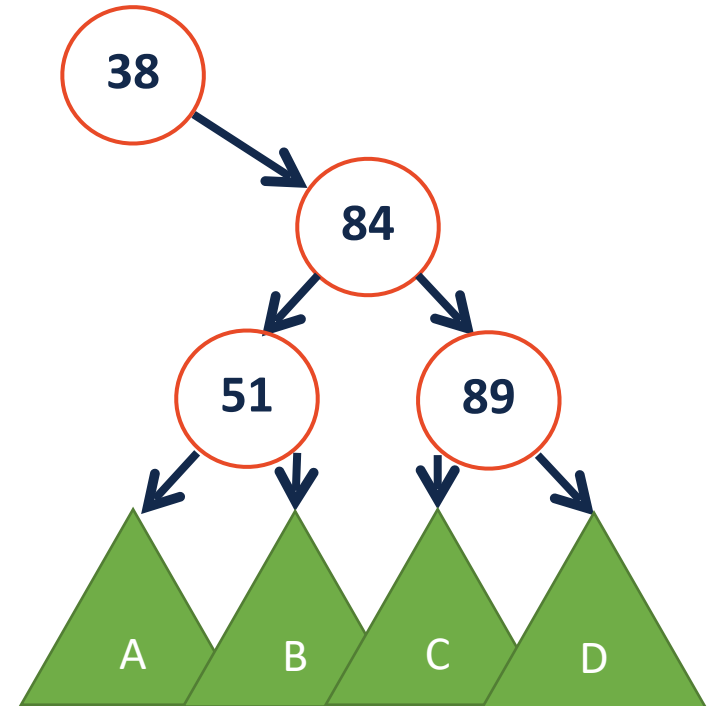
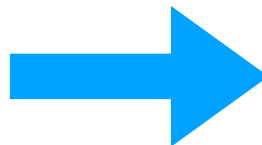
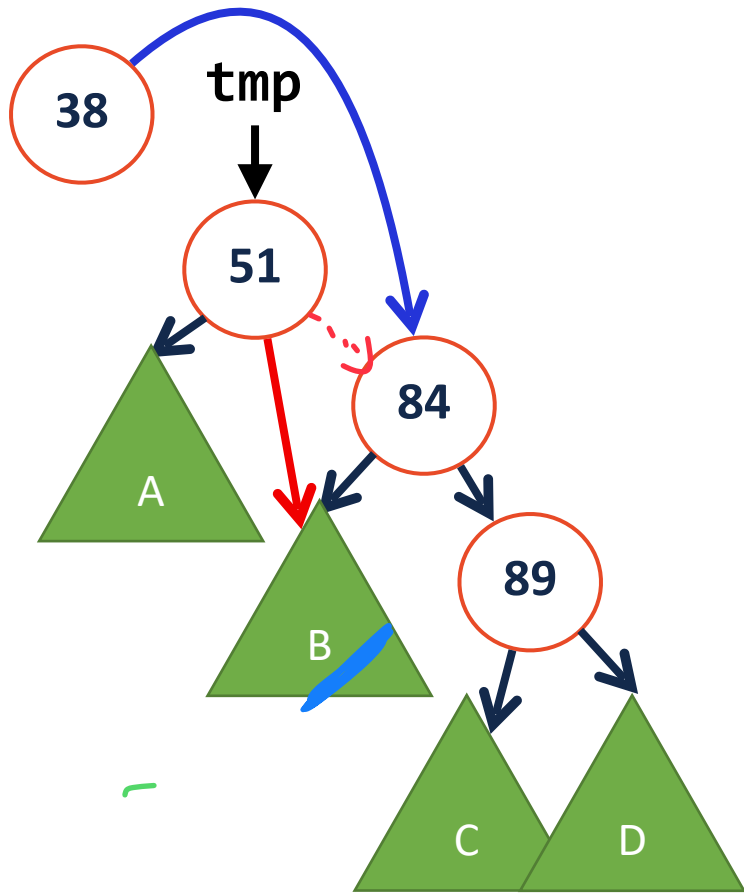
Left Rotation

- 1) Create a tmp pointer to root
- 2) Update root to point to mid



Left Rotation

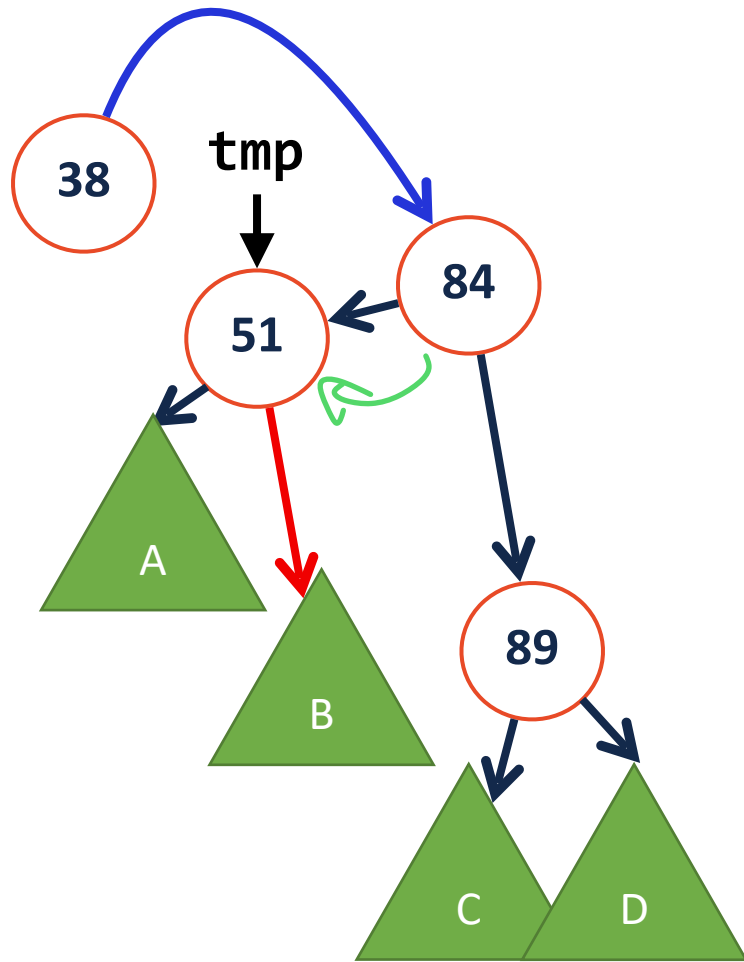
- 1) Create a tmp pointer to root
- 2) Update root to point to mid
- 3) tmp->right = root->left



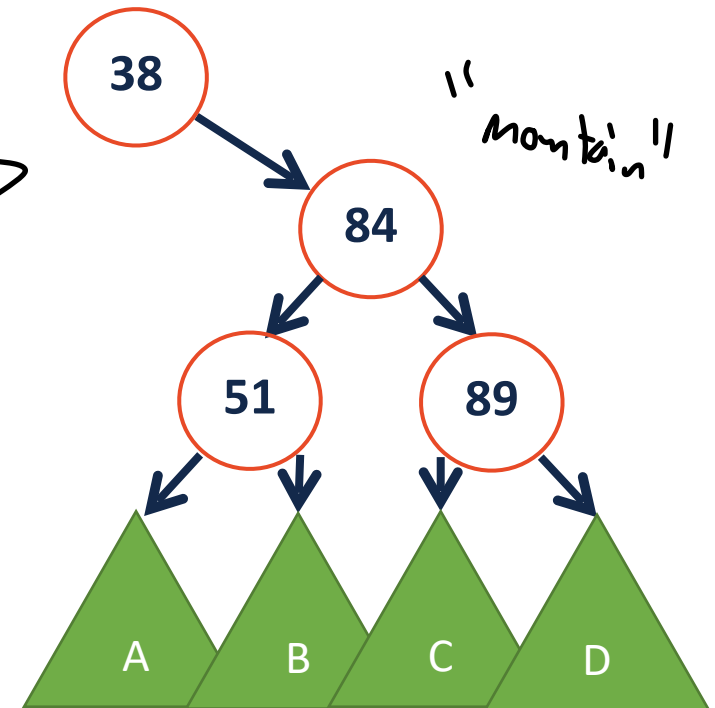
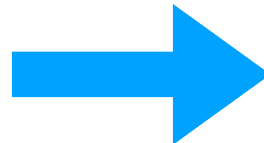
Left Rotation



- 1) Create a tmp pointer to root
- 2) Update root to point to mid
- 3) $\text{tmp} \rightarrow \text{right} = \text{root} \rightarrow \text{left}$
- 4) $\text{root} \rightarrow \text{left} = \text{tmp}$

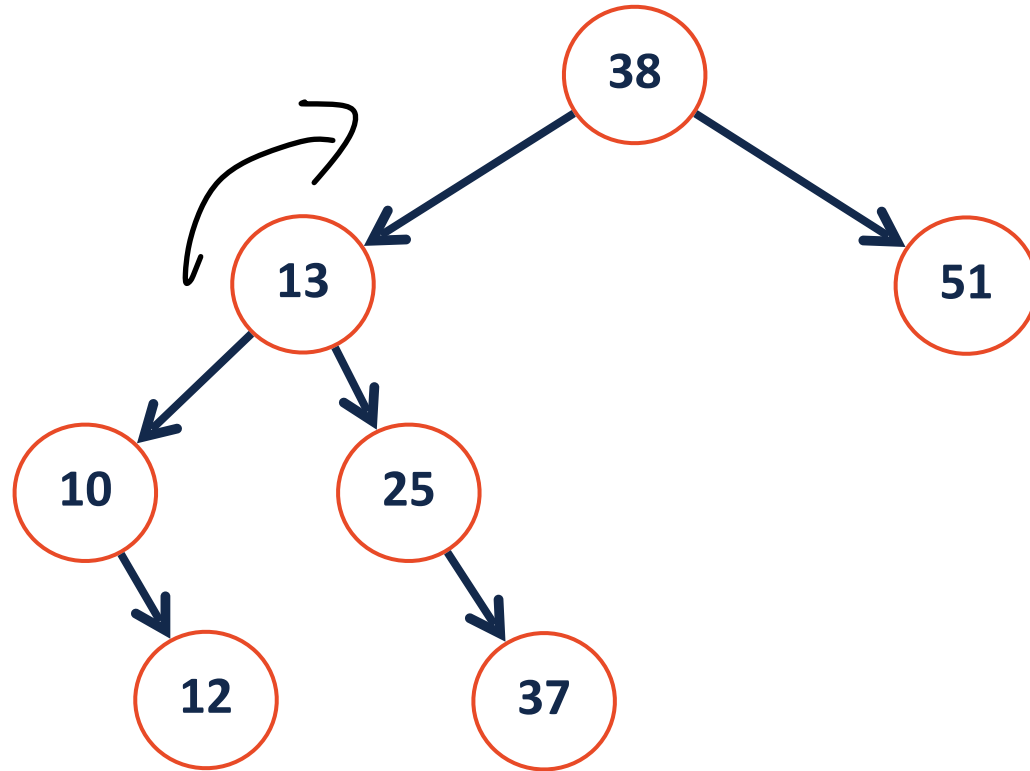


Q
Q
Q
"St !k"

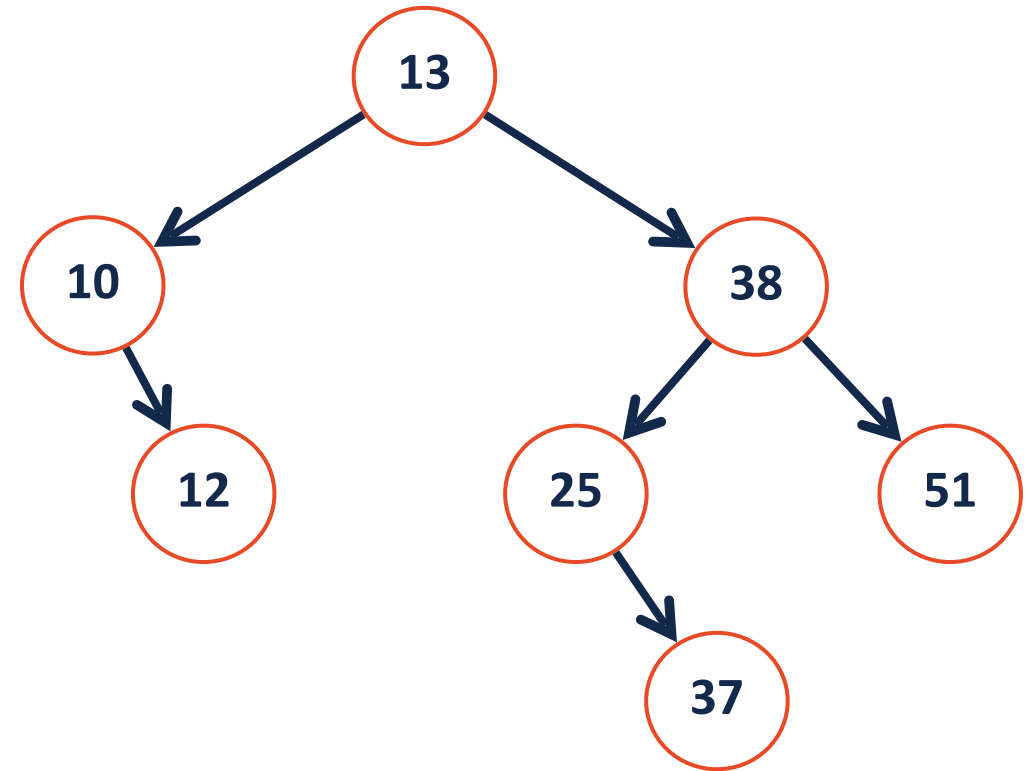
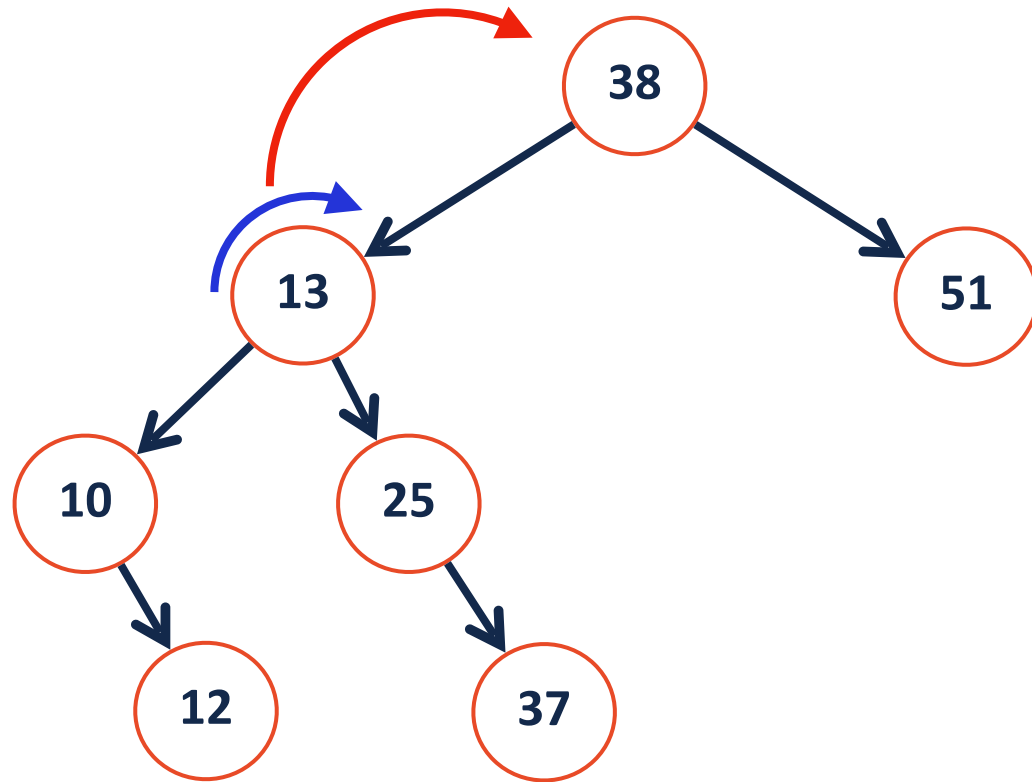


Right Rotation

Stopped here!



Right Rotation



Coding AVL Rotations

Two ways of visualizing:

1) Think of an arrow 'rotating' around the center

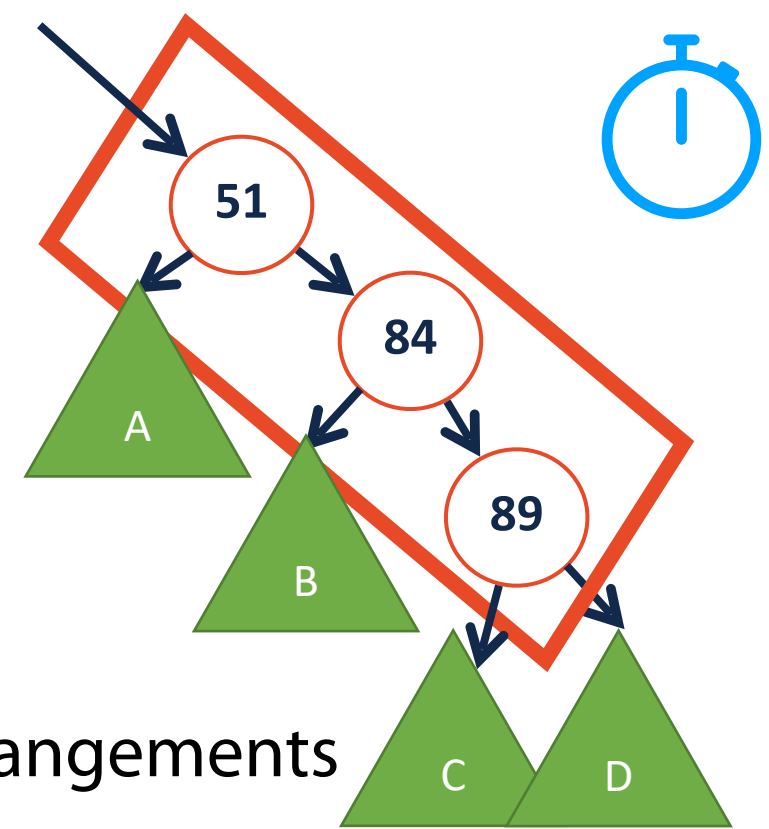
2) Recognize that there's a concrete order for rearrangements

Ex: Unbalanced at current (root) node and need to *rotateLeft*?

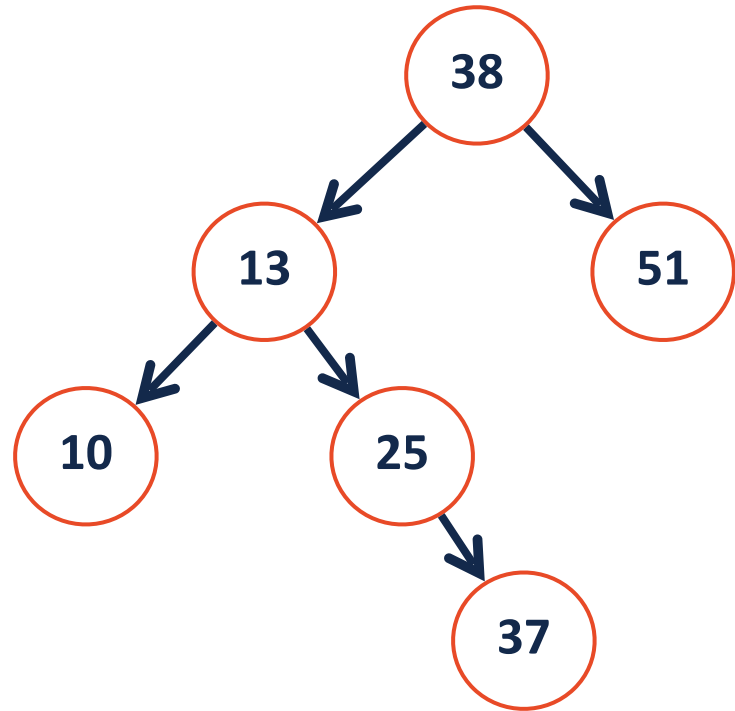
Replace current (root) node with its right child.

Set the right child's left child to be the current node's right

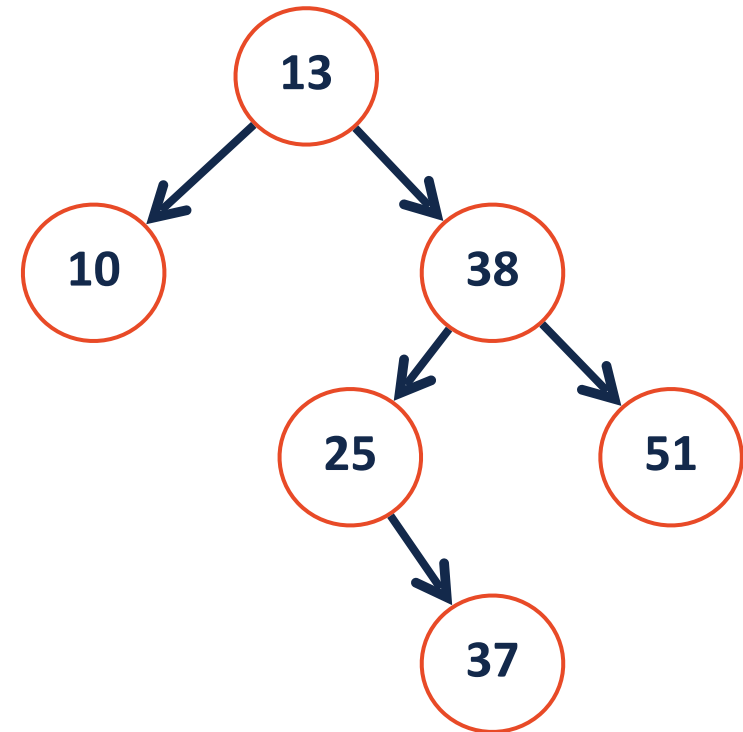
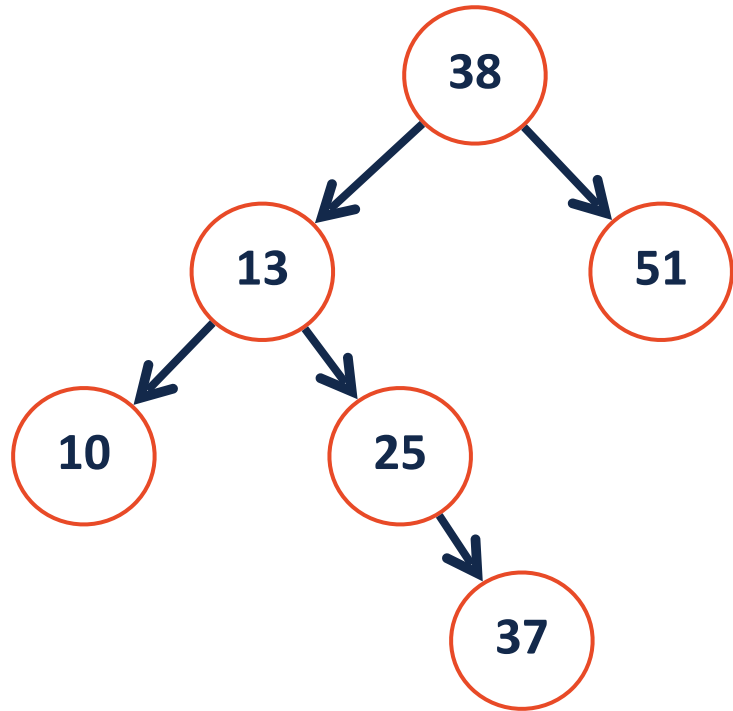
Make the current node the right child's left child



AVL Rotation Practice

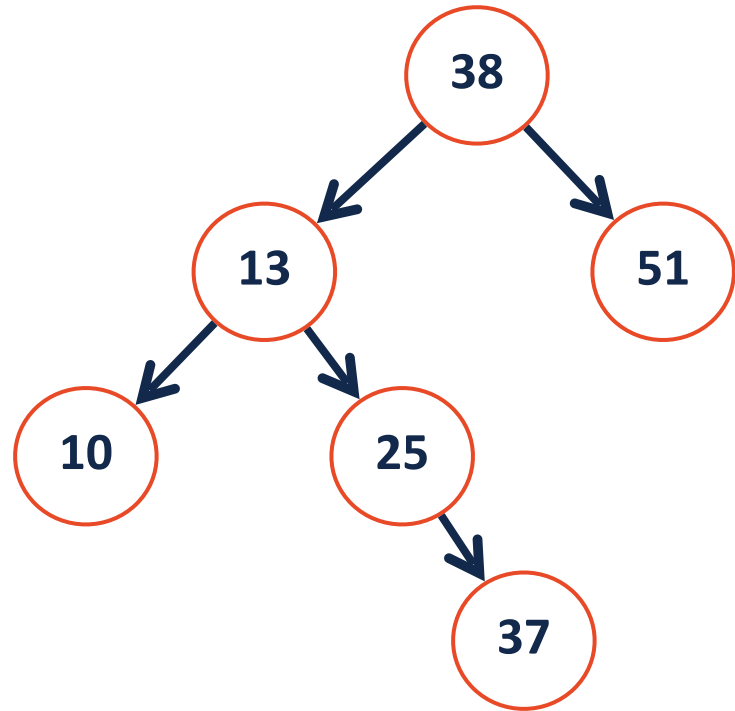


AVL Rotation Practice

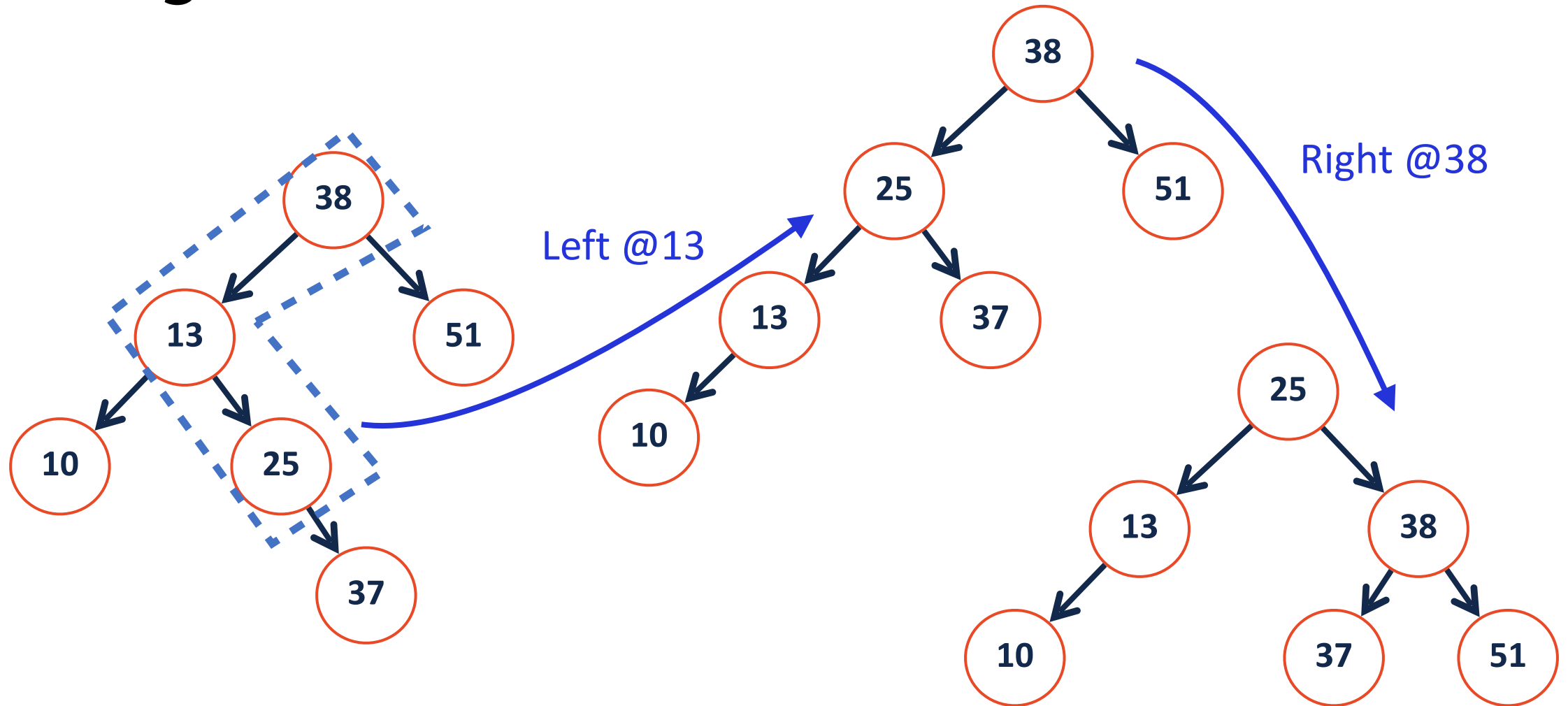


Somethings not quite right...

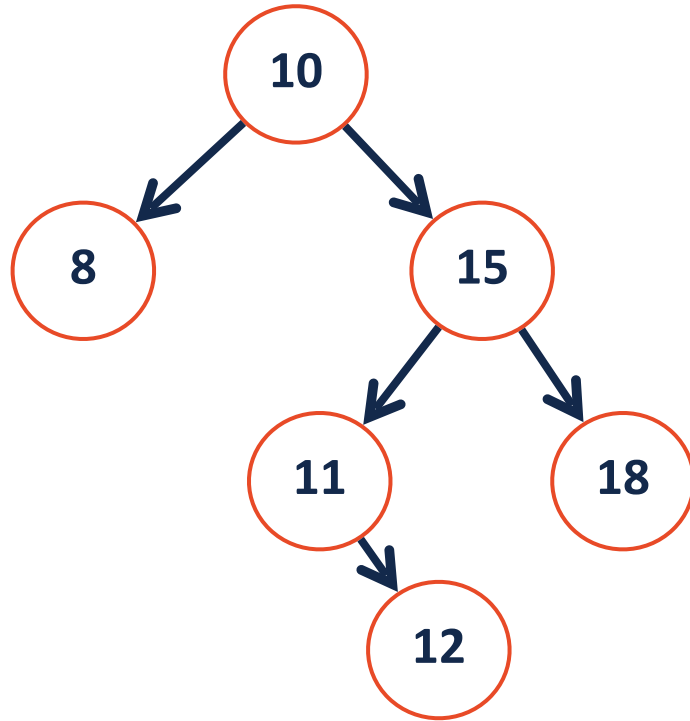
LeftRight Rotation



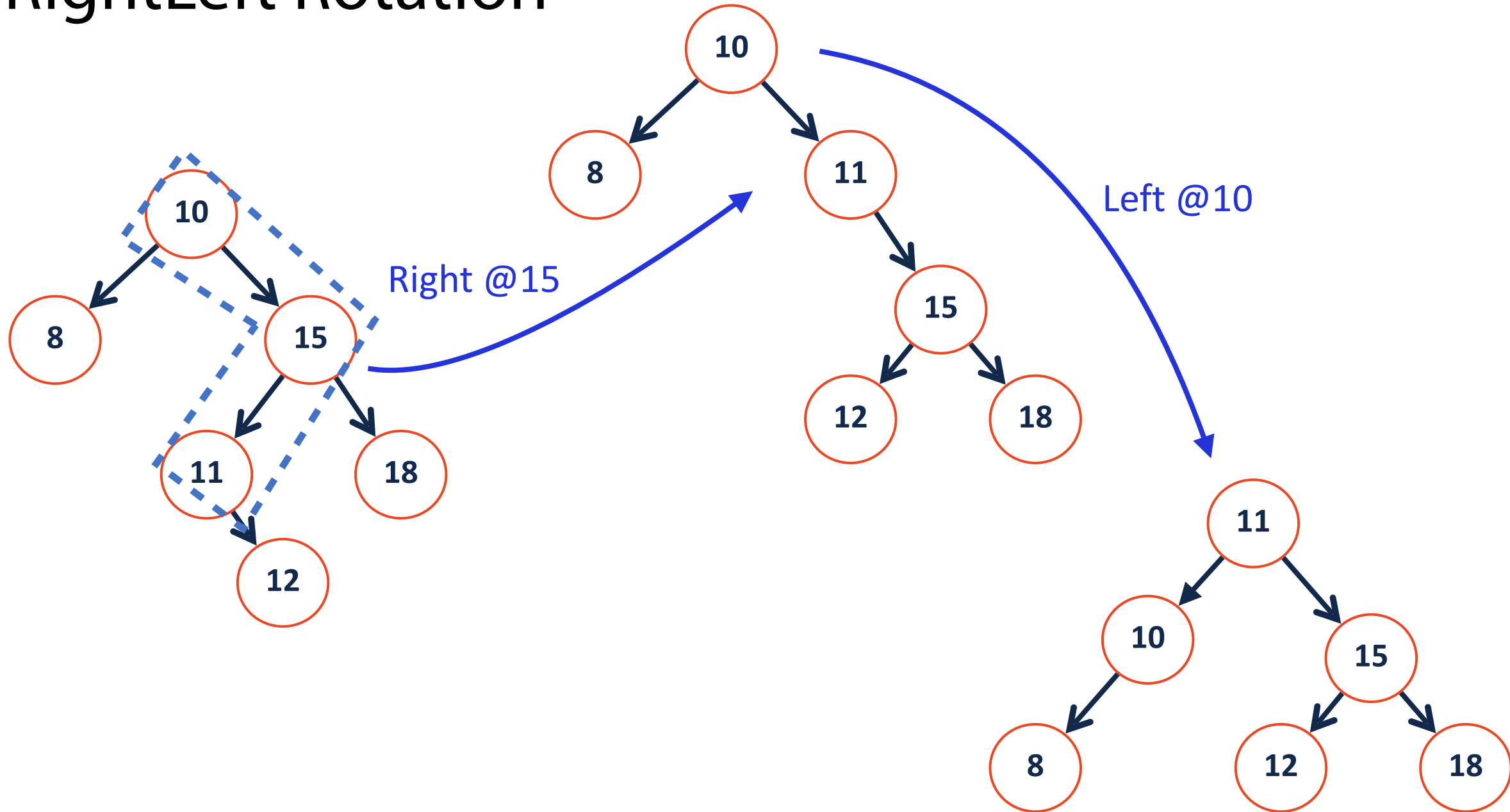
LeftRight Rotation



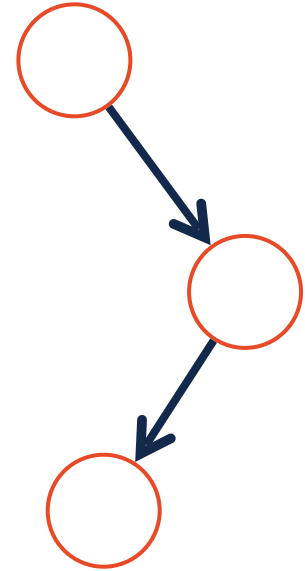
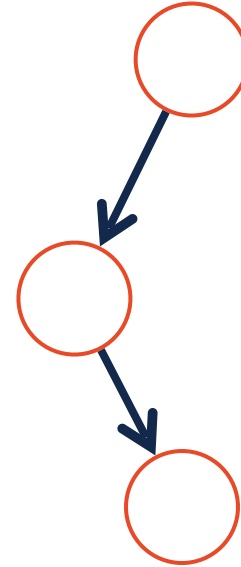
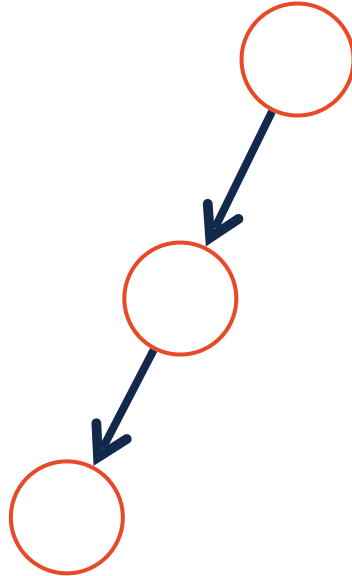
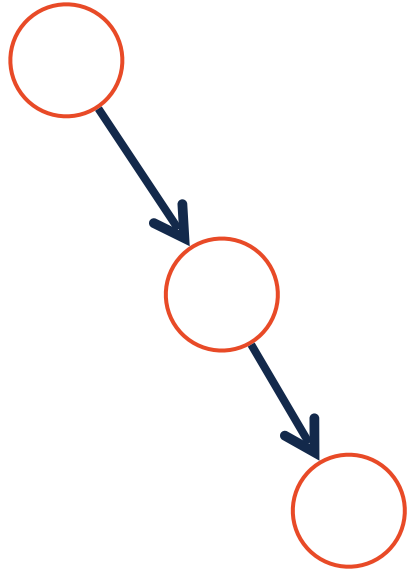
RightLeft Rotation



RightLeft Rotation



AVL Rotations





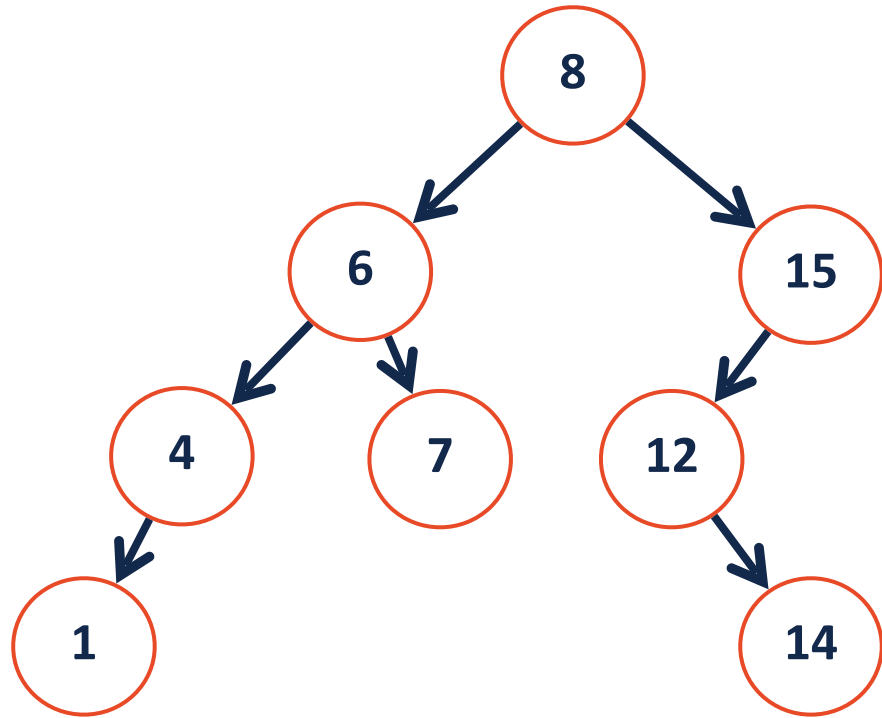
AVL Rotations

Four kinds of rotations: (L, R, LR, RL)

1. All rotations are local (subtrees are not impacted)
2. The running time of rotations are constant
3. The rotations maintain BST property

Goal:

AVL Rotation Practice



AVL vs BST ADT

The AVL tree is a modified binary search tree that rotates **when necessary**

How does the constraint on balance affect the core functions?

Find

Insert

Remove