Order Statistics
Problem
Given a set of integers, give the $k$’th element in the sorted order.

The $k$’th smallest number is called the $k$’th order statistic.

The standard AVL tree can give us the minimum and the maximum but to get elements by position, we need to store more information in the nodes.

This is called augmenting the AVL tree.
Augmentation
There are multiple types of augmentation.

- Store size of the subtree at the node
- Store the minimum of the subtree
- Store the maximum of the subtree

(For the programming assignment, you need to do the second and third)
A standard AVL tree:
After augmenting by size of the subtree:

```
10, 8
  /   \
5, 5   15, 2
  /     /
2, 3   7, 1 20, 1
  /     /
1, 1   3, 1
```
Now to find the $k$’th element, with $k = 5$. 

```plaintext
10, 8
5, 5 15, 2
2, 3 7, 1 20, 1
1, 1 3, 1
```
The left subtree has \( \geq 5 \) elements, which means that the 5th element in the left subtree.
The left subtree has 3 elements, which means that the 5’th element in this subtree is the same as the $5 - 3 - 1 = 1$’st element in the right subtree.
Algorithm 1 OrderStatistic

function OrderStatistic(Node v, Int k)
    if v.left.size < k-1 then
        return OrderStatistic(v.right, k-v.left.size-1)
    elseif v.left.size = k-1 then
        return v.key
    else
        return OrderStatistic(v.left, k)
    end if
end function
Dynamic array interface
A normal array allows for the following operations:

- Find/Change the value at \( k \)’th position.

We will use an augmented AVL tree to extend the array interface to support:

- Insert/Delete element at \( k \)’th position

Note: The array need not be sorted.
We will use the size augmentation and do insert based on positions. Suppose our starting AVL tree is:

5, 1
Do an insert(10, 1): insert 10 at position 1.
This no longer is a search tree but it is still an AVL tree.
Insert(12, 1):
which balances to:

```
    10, 3
   /   \
  5,1   12,1
```
Insert(7, 3):
Insert(50, 4):
which balances to the following, using a left-right rotate at the angle 12-7-50:

```
10, 5
  /
5,1  50,3
     /  /
  7,1 12,1
```

Now we can use the previous algorithm OrderStatistics to get and set the $k$’th element.
Lazy Propagation
Problem
Given a sorted array of keys, \([a_1, \ldots, a_n]\), perform two types of operations on the array:

- Add an integer \(v\) to the values of all keys which are between \(x, y\)
- Find the value of the element at position \(k\)

Assume that the values associated with each key is 0 at the start.

The naive algorithm does the first operation in \(O(n)\) and the second operation in \(O(1)\).
Can we do better?
We will use an AVL tree, along with the minimum, maximum element augmentation.
At each node store:

- minimum key in subtree of node
- maximum key in subtree of node
- a carry integer for storing the increments.
- AVL tree information:
  1. key
  2. value
  3. height
  4. left child
  5. right child
  6. ...
(1) We first make an AVL tree of the sorted array: For the sake of simplicity assume we have the following AVL tree
If we look at only the augmented information, of the form \(\text{min, max}/\text{value}\), ignoring the actual keys at each node.
Add 10 to all values between 13-150.

- find all top level nodes whose min-max range is fully inside the intended update range and add 10 to their carry (blue nodes).
- find all top level nodes whose min-max range intersects with the intended update range in a non-trivial fashion and add 10 to their value (green nodes)
To query the value of any node, we add up the carry values along the path from the node to the root, plus the value currently at the node. For a more comprehensive example, look at the attachment on the class website.
Algorithm 2 Range Update

function Range Update(Node v, Int x, Int y, Int k)
  # Add k to the range [x, y]
  if [v.min, v.max] ⊆ [x, y] then
    v.carry += k
  elseif [v.min, v.max] ∩ [x, y] = ∅ then
    # They do not intersect
    return
  else
    # We intersect a small amount, in which case
    # make two recursive calls
    # and also appropriately add to v.value
    if v.key ∈ [x, y] then
      v.value += k
    end if
    Range Update(v.left, x, y, k);
    Range Update(v.right, x, y, k);
  end if
end function
Observation:

- Whenever two recursive calls are made, only one of them is going to make further recursive calls.
  (Except for the first branching point).
Observation:

- Whenever two recursive calls are made, only one of them is going to make further recursive calls.
  (Except for the first branching point).

Exercise: Try to give a proof of the observation.
Hint: Try to see the conditions under which more recursive calls are made.
Time complexity of range-update:
After the first branching, only one recursive call from each branch is going to reach a leaf.
Which gives us that the total time taken is $O(\log n)$. 
Some subtleties to note:

- When you do an insert, on a range which had a value added to it, you need to be careful in clearing the path to the inserted node.
- While doing rotate left/right need to be careful in managing the carries

This is possible by using a pushdown helper function
Algorithm 3 Push Down

function PushDown(Node v)
    v.value += v.carry
    v.left.carry += v.carry
    v.right.carry += v.carry
    v.carry = 0
end function
First programming assignment:
Implement range-update and point query.