Segment Trees

Teammates: Diego Perez-Torres, Max Mueller, Hayden St. Germain, Alexander Lang

Introduction:

This task assigned to us as a Final Group Project for the course CSC 212 serves as a chance to learn about and create our own code to implement segment trees.

Methods:

A segment tree is a data structure that allows answering range queries over an array with O(logn) time, while allowing the modification of the array (2014).

They are structurally identical to Binary Search Trees, with the only difference being that instead of filling in the tree from top to bottom, left to right; segment trees are built from bottom to top, right to left after generating a tree the size of two times the amount of values minus 1. The purpose of this will be later explained.

Segment trees can be applied for finding the sum/max/min of the range, computational geometry, geometric information systems, and more. It also allows data to be stored arbitrarily (citation here).

History:

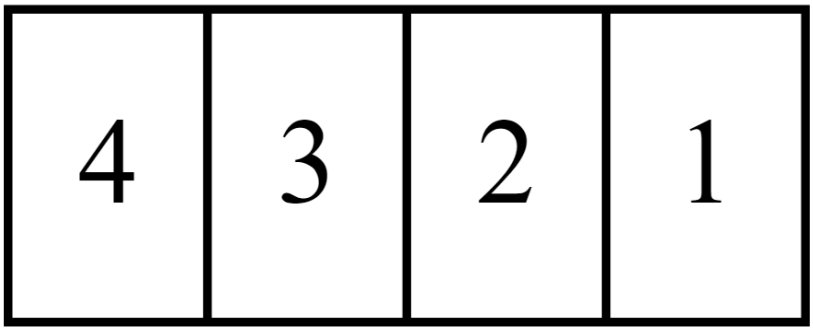
The segmentation tree was first invented by John Bentley in 1977 to solve Klee's Rectangle Problem in a more efficient way. The Klee's Rectangle Problem is a way to determine the union of a plane of rectangles. To solve these problems an algorithm known as the Bentley-Ottmann algorithm is used. The Bentley-Ottmann algorithm uses a sweeping line approach. This means that a vertical line moves across the plane from left to right intersecting the input line segments in sequence as it moves across the plane. Once this line passes through these endpoints that many actions can occur a number can be added to a running total; a min value can be changed or a max value can be changed. Doing this a time complexity can be achieved of O(n log n) which is much faster than a traditional binary search tree of a worst case of O(n).

Implementation:

The underlying structure of our code is simply two vectors. One that stores the range of initial values, and the other imitating the segment tree itself.

The purpose of my segment trees is to either get the sum/max/min of the range of integers.

Using a command line argument to read a txt file filled with comma separated values, the initial vector is stored inside of an object with said values. Let's say I'm looking for the minimum in this tree.
The vector should look like this:



After this, the second vector is created by filling it in with 0s by the size of two times the amount of values minus one. At least that is how it should go. Due to some error or bug in my code, it often outputs the wrong information when I create it by that size. It is more than likely a syntax error on my part, but I did not have the time to fix it. For the sake of simplicity and consistency with the report, I'll continue as if the error was not present.

After the second vector is created, the values from the first should be inserted into the last cells. The order in which the values are placed towards the end does not matter.

The second vector should look like this:

0	0	0	4	3	2	1

Note that there are three 0s. That is because the segment tree has not been completed yet. When building this incomplete segment tree, there will always be n-1 empty spaces. These empty values will store the sum/min/max of the compared values.

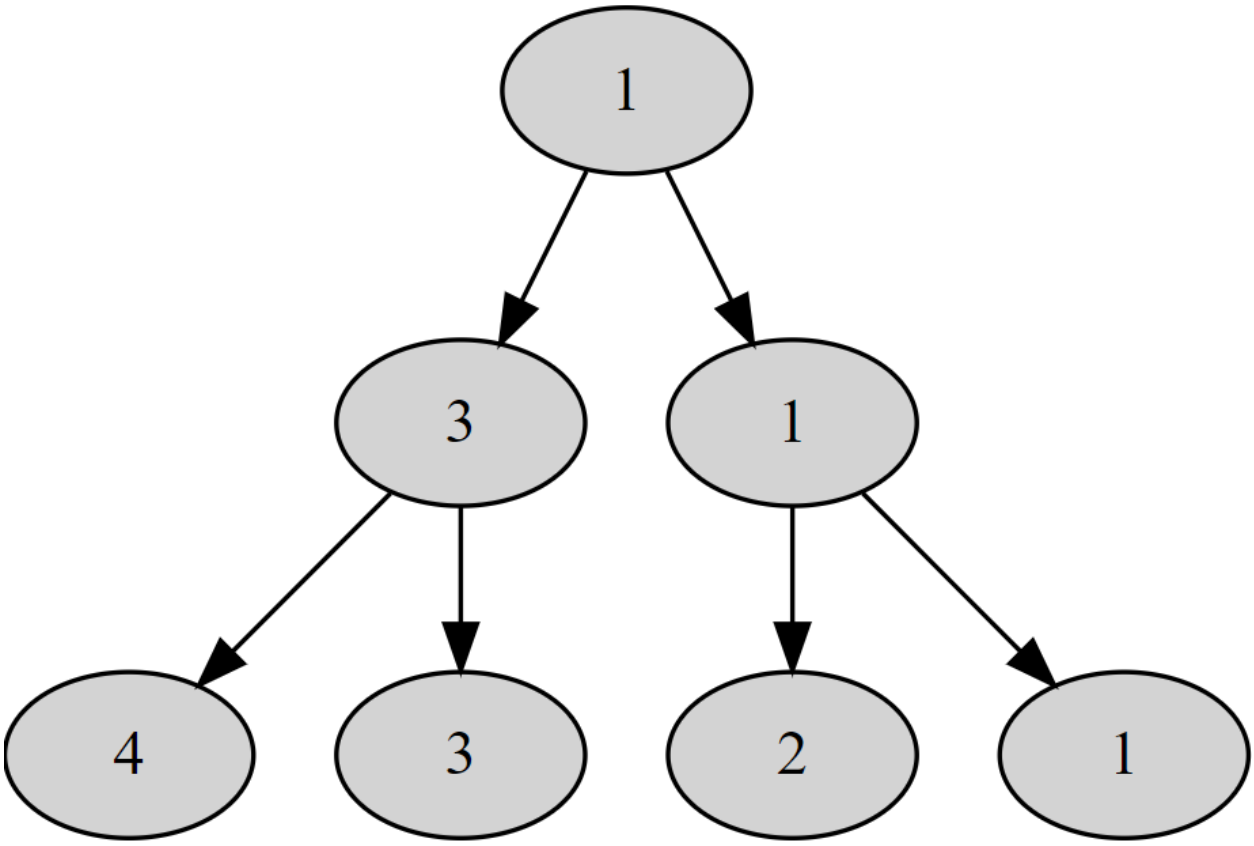
But how would I compare them? This is not a BST, so it cannot call for left nor right pointers.

That is where the position of the indexes come in. The root index is determined by dividing the index by half. Seeing as how integers always round down to the nearest integer, the root of example, index 6, will always be index 3.
At index 3, the minimum between index 5 and 6 will be placed there. This continues until the index which stores the last integer from the original vector is passed by index +=2. We only calculate the root index with even indices.

The vector should now look like this:

1	3	1	4	3	2	1

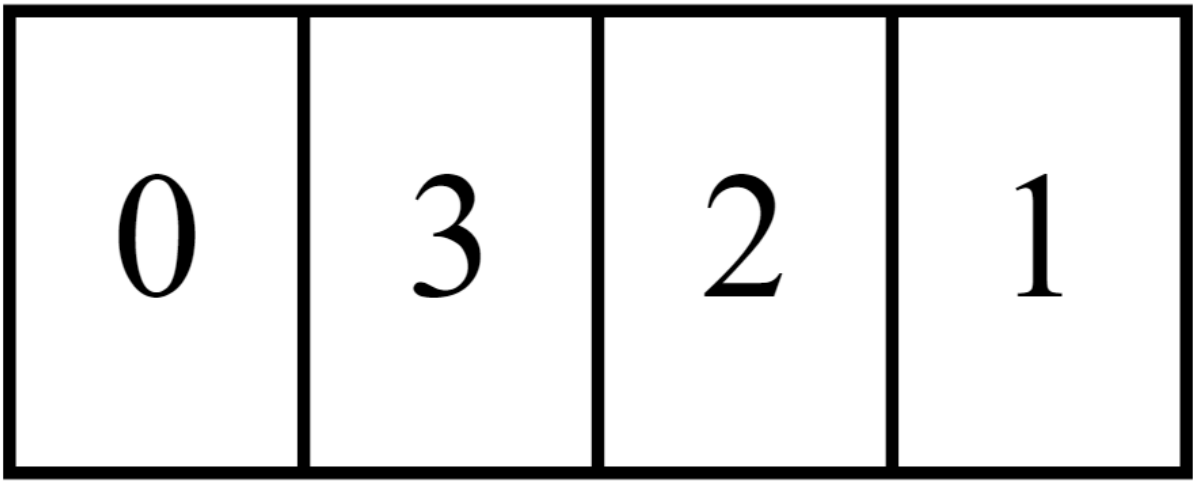
This is now a complete segmentation tree, and just like a normal BST, it is filled in from top to bottom, left to right; with each layer no bigger than 2^n starting at n = 0.



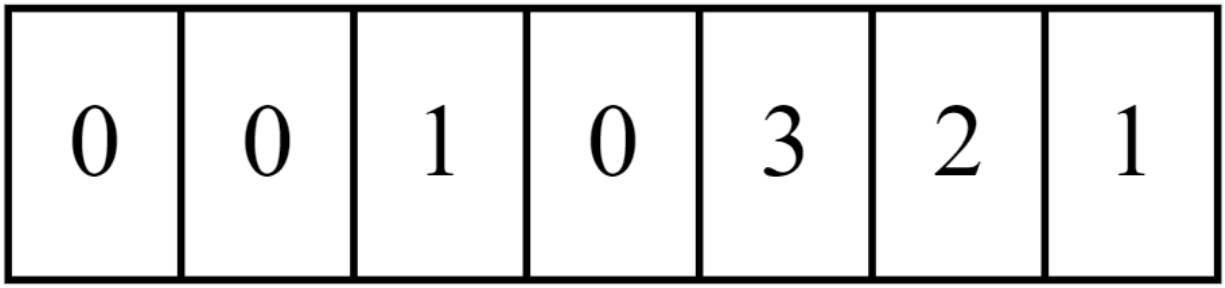
Let us say we wanted to update a value in the original vector to possibly find a new minimum. Because this is a vector, changing the value at any index you desire is simple.

data[index] = newVal;

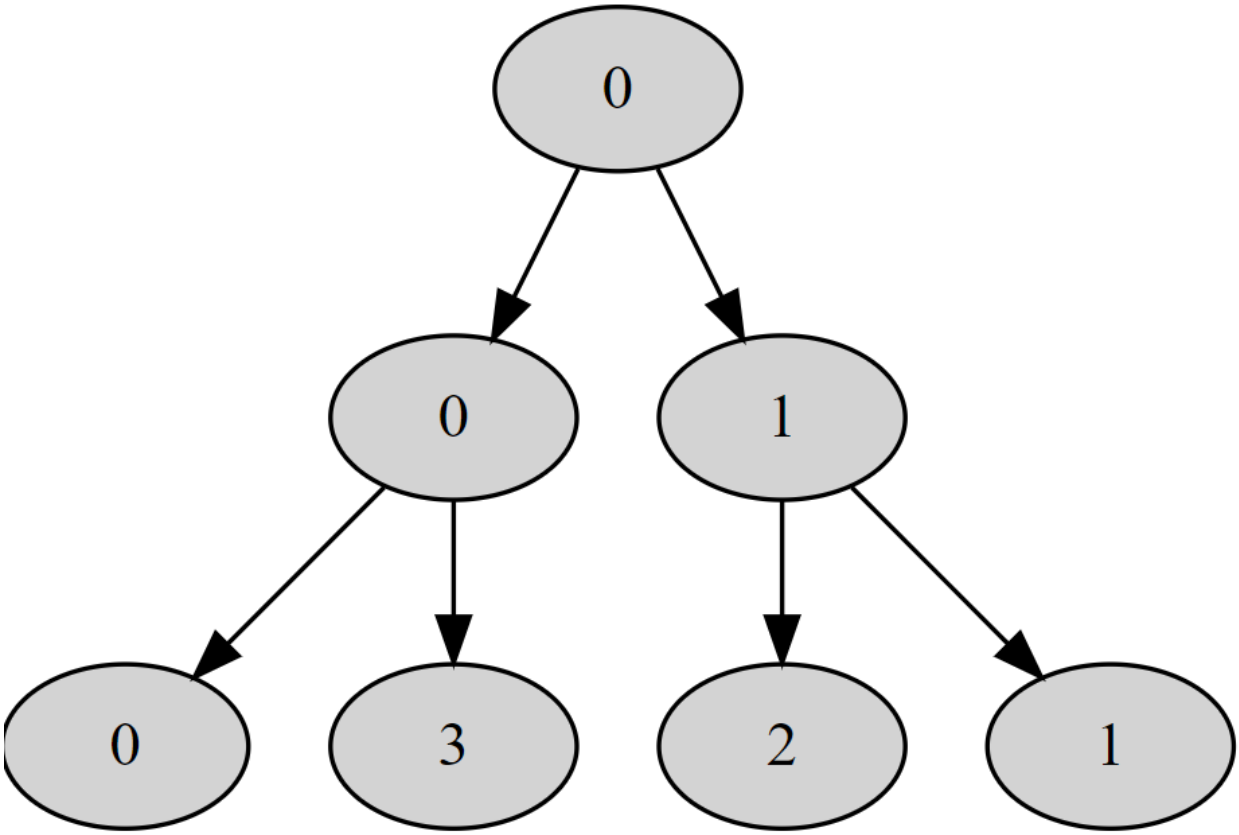
Let's say we wanted to update the first value with 0:



The second vector would update to this using the same steps as previously:



And thus the segment tree would look like this:

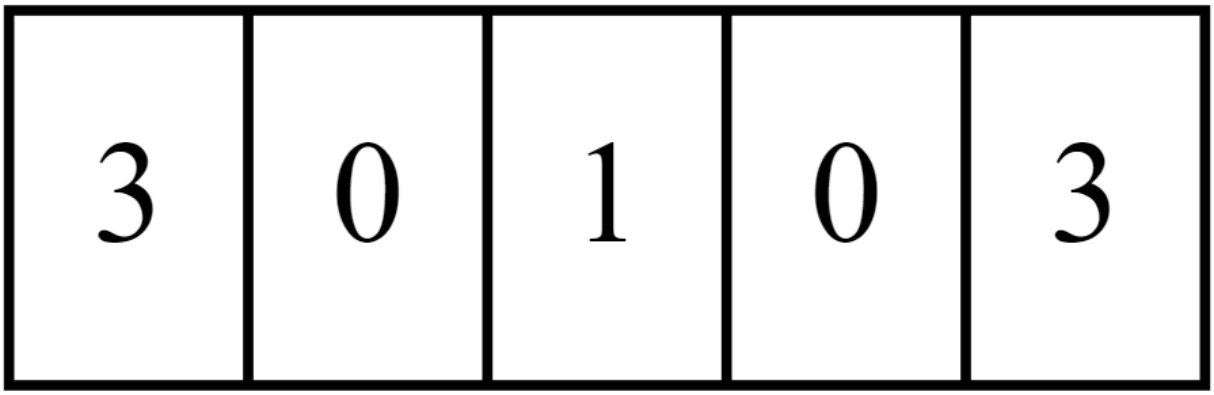


That is how the segment tree would update if a value in the range was updated.

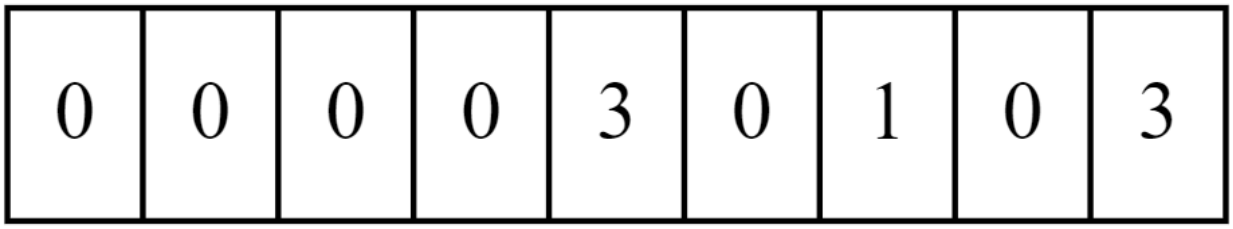
Now let us say that we want to insert a new value into the vector, once again at the beginning with the int 3. Because this is still a vector, that is simple enough to do.

original.insert(original.begin() + insIndex, insValue);

The first vector should look like this:

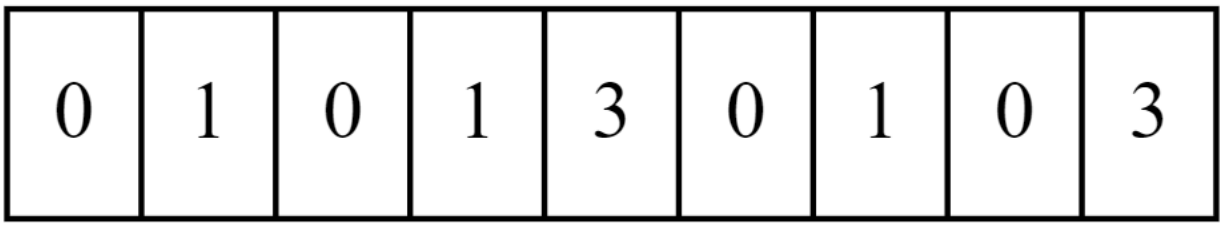


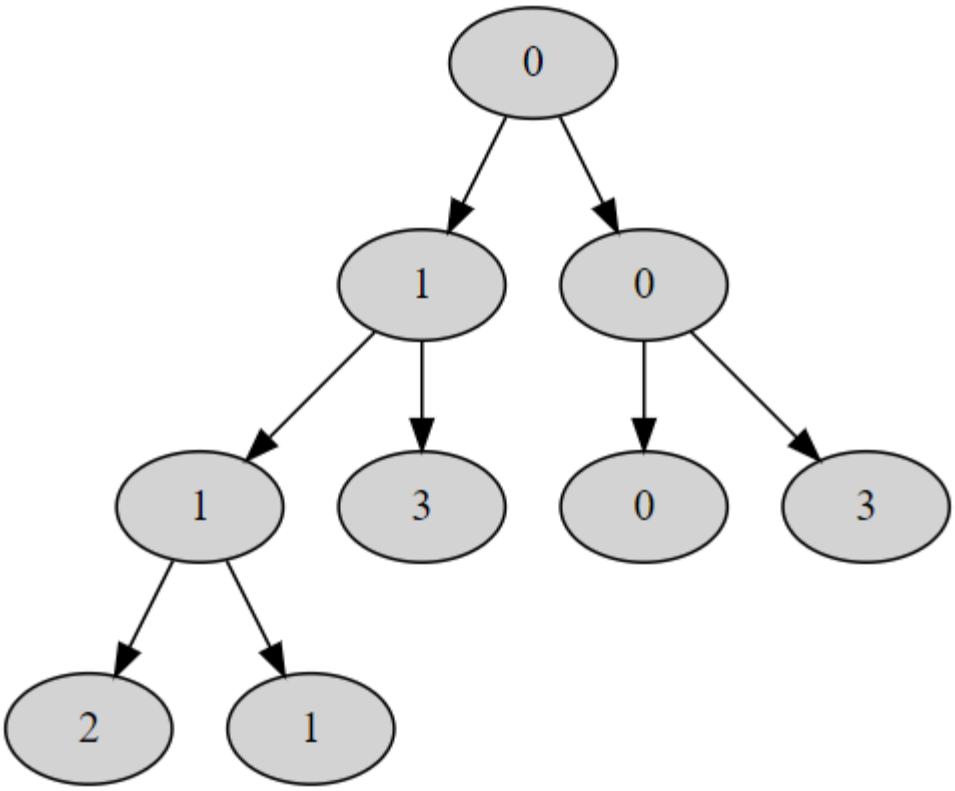
And the second vector like this with once again the same steps taken earlier:



Notice how after the inserted 3, the number of 0's has increased. That is because of earlier established n-1 being the amount of empty spaces meant to be filled with minimums.

Complete segment tree:





If you wanted to search through the completed vector for a specific value, a simple for loop would suffice.

The same goes for actually viewing the entire segment tree and what index has what value. To make a dot file like the one above however, another for loop is required.

The second for loop is initialized with a variable named current to 2, where it outputs into a text file a variable with a label of its stored value like so:

C1 [label = 0]

Then the variable is connected to current and current +1 like so:

C1 -> {C[current = 2], C[current +1 = 3]}, until current is greater than 2(size of vector)-1

I then copy the contents of the text file and paste it into this online dot file viewer: https://dreampuf.github.io/GraphvizOnline/

Contributions:

Presentation description, real-life application and history: Max Mueller, Hayden St. Germain, Alex Lang
Presentation explanation of code: Diego Perez-Torres
Report methods: Max Mueller, Hayden St. Germain, Alex Lang
Presentation implementation: Diego Perez-Torres

Citations:

"Segment tree," *Segment Tree - Algorithms for Competitive Programming*, 2014. [Online]. Available: https://cp-algorithms.com/data_structures/segment_tree.html. [Accessed: 24-Apr-2022].

Baeldung. "Segment Tree and Its Applications." *Baeldung on Computer Science*, 22 Oct. 2021, https://www.baeldung.com/cs/segment-trees.

[Accessed: 24-Apr-2022].

“Klee's measure problem,” *Wikipedia*, 02-Dec-2020. [Online]. Available: https://en.wikipedia.org/wiki/Klee%27s_measure_problem. [Accessed: 26-Apr-2022].

“Bentley–ottmann algorithm,” *Wikipedia*, 02-Apr-2022. [Online]. Available: https://en.wikipedia.org/wiki/Bentley%E2%80%93Ottmann_algorithm. [Accessed: 26-Apr-2022].