Handson 2 - Anna Francesca Montagnoli

For the implementation of the segment trees I consulted this site: https://cp-algorithms.com/data_structures/segment_tree.html

Exercise 1

build(a, v, tl, tr): function that builds the segment tree.

- The function uses recursion to construct the segment tree in a top-down manner.If the left and right endpoints are equal (tl == tr), it means the current node represents a leaf in the segment tree, and its value is set to the corresponding element from the given array a.

- If the left and right endpoints are different, so the function recursively builds the left child node and the right child node. The value of the current node v is set to the maximum of its left and right children.

update(v, tl, tr, i, j, val): function that replaces every value of the tree inside the range [i,j] with the minimum of the current node and the value val.
I used the lazy propagation so the function updates just the necessary nodes and uses the lazy tree to keep track of the nodes that still need to be updated.

- The function uses recursion to navigate the segment tree and update the necessary nodes.

- If the given range [i, j] is invalid (i > j), the function simply returns without performing any updates.

- If the current node represents the exact range that needs to be updated (i == tl && j == tr), the function updates the node value as the minimum of the current node value and the value val.

- If the current node represents a range larger than the one to be updated, the function first pushes any pending lazy updates to the children. Then, it recursively updates the left and right child nodes for the appropriate subranges.

The ***push*** function propagates lazy updates from a parent node to its children in the segment tree.

max(v, tl, tr, i, j): computes the maximum in the given range.

- If the range is invalid (i > j), it returns 0.

- If the current node exactly covers the query range [i, j], it returns the stored value in that node.

- Before proceeding with the query, it propagates any pending lazy updates to the current node using the push function.

- The function then recursively queries the left and right children of the current node and returns the maximum value obtained from these recursive calls.

The overall time complexity is O((n+m)logn)
The time for building the segment tree is nlogn where n is the size of the input array. This is because the function traverses each level of the segment tree (logn levels) and performs *O*(n) work at each level.
The time for one query is O(logn): at each level of the recursion, the function performs constant-time operations, and the total number of recursive calls is logarithmic.

The space complexity is O(n^2) because for representing the segment tree and the lazy tree I use two arrays of 4*n positions.

Exercise 2

From the given segments I used the sweep line technique for building an array which stores in every position the number of segments that intersect that position.

The ***build*** function builds the segment tree from the array mentioned before. The segment tree is the same as the first exercise, in fact every node stores the maximum value between its children.

The recursive function ***is_there*** checks if there exists a position within the range [i,j] of the original array where the value is equal to a given value k.

- If the range is invalid (i > j), it returns 0.
- If the current node exactly covers the query range [i, j], it checks if the current node's value equals k. If it does, it returns 1; otherwise, it recursively checks the children if the current node is not a leaf.
- If the query range is not fully covered by the current node, the function recursively checks the children.
- The result of the recursive calls is the maximum value obtained from checking the left and right children.

The overall time complexity is O((n+m)logn)
The time for building the segment tree is nlogn where n is the size of the input array. This is because the function traverses each level of the segment tree (logn levels) and performs *O*(n) work at each level.
The time for one query is O(logn): at each level of the recursion, the function performs constant-time operations, and the total number of recursive calls is logarithmic.

The space complexity is O(n) because for representing the segment tree I use an arrays of 4*n positions.