Second Handson Report

Simone Stanganini mat. 616834

November 24, 2025

- **Problem 1: Min and Max**:

  For this implementation i decided to put the lazy value directly inside the node structure for simplicity without having to manage separate ST. The build phase follows a standard recursive approach: it starts from the root (covering the largest range) and splits down until it reaches the single elements (leaves). The computational cost for this is $\Theta(n)$ because we do at maximum 2n -1 operation.

  For the update function, the logic changes based on the overlap:

  - Total overlap, i only update the lazy value, if it is already a lazy node, we simply take the minimum.
  - Partial overlap, i make recursive calls to the children, before doing that, if i find a lazy value, i push it down to the children.
  - No overlap, i still return the current value of the node to make sure that the parent nodes always have the correct maximum value.

  All of this keeps the complexity at $\Theta(\log n)$.

  The max query works in a similar way:

  - If i reach a leaf, i update the answer directly (since leaves are always in total overlap here).
  - In case of total overlap, we directly take the result, for lazy value as always we propagate.
  - In case of partial overlap, i also propagate the lazy value first (because there was a previous total overlap there), and then i call the function recursively on the children.

  This operation also costs $\Theta(\log n)$.

  Finally, the propagate function simply pushes the lazy value to the children. If it finds a leaf, it updates its value immediately. So the total cost in time is $\Theta(n + m \log n)$ and $\Theta(n)$ space.

- **Problem 2: Is There**:

  The big problem here was how to answer `is_there` in costant time in case of total overlap. So for doing this task i used an HashSet, this way inside each node, we keep all the elements seen in that range and i can answer in costant time.

  For building the tree the cost is $\Theta(n \log n)$ because we iterate the overlaps array for each node, however initialy, there is a linear cost to build the array of overlaps, and this is done using a sweepline algorithm.

  Finally the `is_there` function perform a standard search on a the segment tree:

  - No overlap: return nothing.
  - Partial Overlap: We search recursively among the children.
  - Total Overlap: simply check if we have the answer inside the node (using the HashSet).

  The total cost of this solution is $\Theta((n + m) \log n)$ and $\Theta(n \log n)$ in space because of the hashset in each node.