Lab 11: Heaps and Treaps Reflection

Lab 11: Heaps and Treaps Group 14: Ju Ho Kim, Sangmin Kim Date: May, 11th, 2025

- After implementing both the MinHeap and Treap data structures, I gained valuable insights into how these structures maintain their properties and when to use each one effectively.
- The heap maintains its order through a process called percolation (or bubbling). When we insert a new element, it's initially placed at the end of the array to maintain the complete tree property. Then, the element percolates up by repeatedly comparing with its parent and swapping if it's smaller (in a min-heap). This upward movement continues until the heap property is satisfied. Similarly, when removing the minimum element, we replace it with the last element and let it percolate down by comparing with children and swapping with the smaller child. This percolation mechanism ensures O(log n) time complexity for insertions and deletions while maintaining the heap property that parents are always smaller than their children.
- Priorities help the treap stay balanced by introducing randomness into the tree structure. While a regular BST can become unbalanced (worst case: a linked list with O(n) operations), the treap uses random priorities to force rotations that maintain balance. When we insert a node, it follows BST rules for the key, but if its randomly assigned priority is higher than its parent's, we perform rotations to maintain the heap property for priorities. This randomization provides expected O(log n) height with high probability, preventing adversarial input patterns from creating unbalanced trees. I would use a heap when I need quick access to the minimum (or maximum) element, such as in priority queues, Dijkstra's algorithm, or heap sort. Treaps are better when I need both efficient searching by key and want to avoid worst-case BST scenarios without complex balancing logic like in AVL or Red-Black trees.