1. Heap (Priority Queue Implementation)

1.3 What is the time complexity of the insert and extract operations in your implementation?

The insert operation places the new element at the end of the heap. Then it performs a “bubble up” step. In the worst case, this process might move the element from the bottom to the root position. Since the binary heap is a complete binary tree with height proportional to O(log n), the worst case time complexity for insertion is O(log n).

In the extract operation, the root element is removed in O(1) time. To maintain the heap structure after removing the root, the last element in the heap is moved to the root position. Then a “bubble down” step is performed. In the worst case, this element may have to move down the height of the tree, which again leads to a worst case time complexity of O(log n).

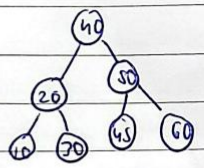
1. 4 Explain how the heap maintains its structure after an insertion or extraction.

Insertion = When a new element is added to the heap, it is placed in the next available position at the bottom to maintain the complete tree structure. However, this can violate the heap property. To fix this, the newly inserted element is compared with its parent. If the heap property is not maintained, the two elements are swapped. This “bubble up” process continues until the heap property is fully restored, ensuring every parent is correctly ordered relative to its children.

Extraction = When the root element is removed, the heap temporarily loses its proper structure. To correct this, the last element in the heap is placed at the root. This ensures the heap remains a complete tree but may violate the heap property. To restore the property, the root element is compared with its children, and swaps are made if necessary. This “bubble down” process continues until the heap property is restored, so that every parent child relationship once again follows the correct order.

2. Binary search BST

2.1 Construct the tree



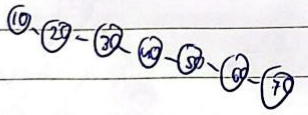
2.2 Perform an in-order traversal. Write down the sequence.

10, 20, 30, 40, 45, 50, 60

2.3 Explain what happens if the sequence is sorted. ([10,20, 30, 40,50 60, 70])

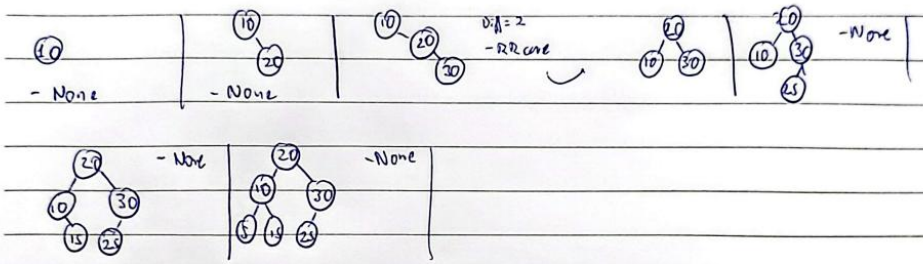
-This will create a not balanced tree, with each element after another.

-Also every time complexity will be O(n) and not O(log n)



3. AVL Tree

3.2 After the sequence simulate the AVL Insertion.



3.3 Explain why balancing ensures O(log n) height for an AVL Tree.

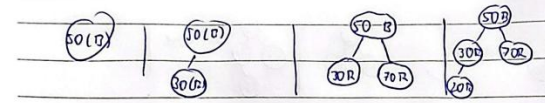
- Because if not balance, that meaning ~~just~~ ordered, then you end up with a linked list, that has a time complex of O(n).

- As In AVL when does happens you can do a rotation (LL, RR, LR, RL) so that the node are redistributed and it keeps being balanced.

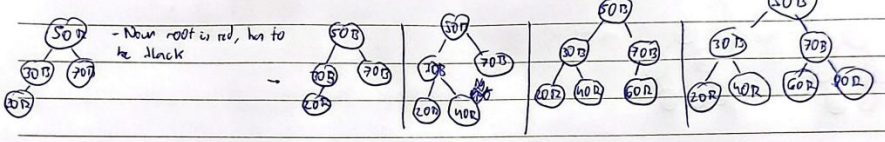
- In the case of 30, the tree became unbalanced (to the right), so that the time complexity was O(n). By performing the Right-Right rotation I rebalance the tree. So that now the time complex is O(log n)

4. Red-Black Tree

4.1 Construct the tree for [50, 30, 70, 20, 40, 60, 80]



- I have to recolor 50 to red and recolor 30 and 70 to black



- Now root is red, has to be black

4.2 Compare to AVL Trees. Which one to choose for frequent insertions.

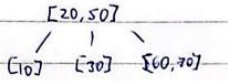
AVL - more balance, each node subtree is at most 1. Faster lookups because of tighter height constrains. However insert requires more rotations to keep balance strict.

Red-Black - slower lookup because have more relaxed balancing rules.

=> Red-Black is my choice - Because it can minimize the overhead of rebalancing operations when inserting. Still maintains O(log n) performance.

5 2-4 Tree

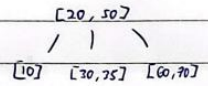
5.1 Simulate the insertion of [30, 35, 40]



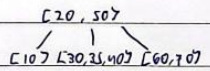
- root node has keys = 2 children
- each children is a leaf node.

Insert 30 → no change, already exists

Insert 35 → goes to the middle, there is space with 30.



Insert 40 → goes to the middle key. A leaf can have up to 3 keys, so no split is needed



5.2 Explain why it maintains O(log n) in search, insert and delete.

- It ensures ~~every~~ the height grows very slowly as more elements are included.
- The operations involve traversing from the root down to a leaf. Since the height grows at most O(log n), these operations also take O(log n) time.

Advantages

1. Balance is easier - It dont requires rotations, it simply splits. ~~and~~
2. Less time to do split - compared to rotation, this takes less manipulation of pointers in code.