Question 3

Problem 1: Merge Sort, because it is stable (maintains the original order of duplicate books) and has guaranteed $O(nlog_2(n))$ worst case performance, which makes it fast enough to inform the librarian before end of day.

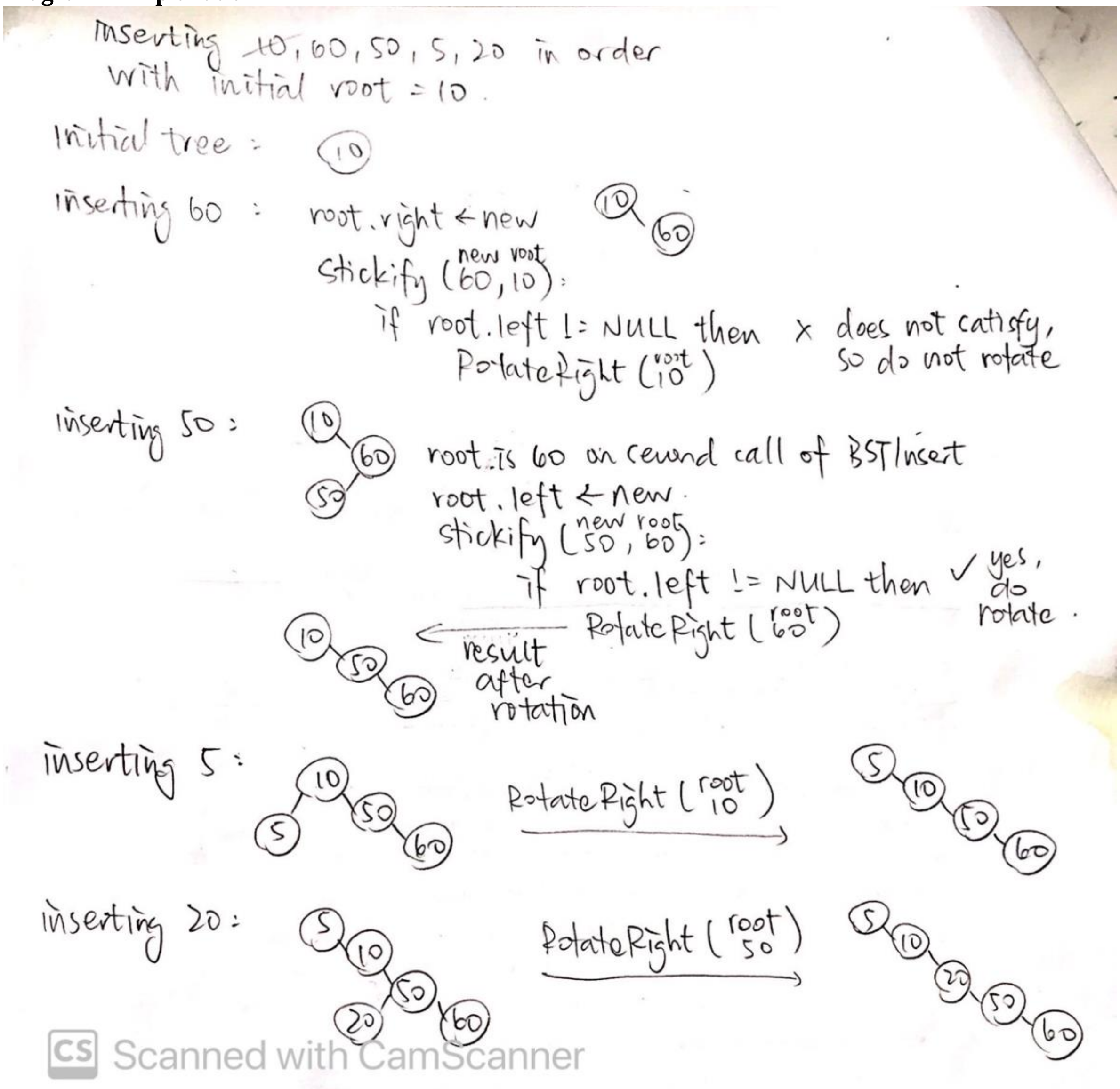
Problem 2: Selection Sort - it is insensitive to input data, best/average/worst case performance are all $O(n^2)$, which is good for sorting arbitrary records. Also, each item is swapped directly into their correct position, hence minimising record swaps by requiring a maximum of n swaps.

Solution 3: Counting Sort, because the input is a set of non-negative integers and the algorithm has a linear runtime O(n+k), with k = 10 in this case, as its worst case performance. As the maximum key value (10) is significantly smaller than the number of values to be sorted, Counting Sort is very space efficient as it only uses an array of space O(10) to store its counts.

Question 4

```
function Stickify(new, root):
        if root.left != NULL then
                RotateRight(root)
```

Diagram + Explanation



Inserting 10, 60, 50, 5, 20 in order with initial root = 10.

Initial tree :

inserting 60 : root.right ← new

Stickify $(\overset{\text{new}}{60}, \overset{\text{root}}{10})$:

if root.left != NULL then × does not catisfy, so do not rotate

RotateRight $(\overset{\text{root}}{10})$

inserting 50 : root is 60 on cecond call of BST Insert

root.left ← new

stickify $(\overset{\text{new}}{50}, \overset{\text{root}}{60})$:

if root.left != NULL then ✓ yes, do rotate.

RotateRight $(\overset{\text{root}}{60})$

result after rotation

inserting 5 : RotateRight $(\overset{\text{root}}{10})$

inserting 20 : RotateRight $(\overset{\text{root}}{50})$