CS2040S Tutorial 6

AY 24/25 Sem 2 — github/omgeta

Q1. (a.) Buckets = `[[42, 0, 7], [36], [], [24, 52, 45], [18, 60, 32], [47], [27]]`

(b.) Lookups are faster with BSTs but Insertion are faster with linked lists. Overall, we prefer BSTs when we lookup more than insert.

(c.) Key needs to be stored in case of hash collision where multiple keys have the same hashed key. We need to store the original key to differentiate the values during search.

Q2. Counting Sort with List to store nodes found: $O(n + t \log t) = O(n)$ since t is small

Q3. Solution:

```
class RandomizedSet {
    private Map<Integer, Integer> map; // val -> index
    private List<Integer> list;
    private Random rand;

    public RandomizedSet() {
        map = new HashMap<>();
        list = new ArrayList<>();
        rand = new Random();
    }

    public void insert(int val) {
        if (map.containsKey(val)) return;
        map.put(val, list.size());
        list.add(val);
    }

    public void remove(int val) {
        if (!map.containsKey(val)) return;
        int idx = map.get(val);
        int lastVal = list.get(list.size() - 1);
        list.set(idx, lastVal);
        map.put(lastVal, idx);
        list.remove(list.size() - 1);
        map.remove(val);
    }

    public int getRandom() {
        return list.get(rand.nextInt(list.size()));
    }
}
```

Q4. Solution:

```
class Custom {
    private AVLTree<Node> t;
    private HashMap<Key, Node> m;

    public void insert(Key k) {
      // Create Node and insert into AVL Tree normally
      // Store successor and predecessor at time of Insertion
      // Add pointer to AVL Node with key to Hash Map
    }

    public void remove(Key k) {
      // Remove Node from AVL Tree normally and adjust links
      // Remove Node from Hash Map
    }

    public boolean lookup(Key k) {
      // Check if Node in Hash Map
    }

    public Node successor/predecessor(Key k) {
      // Lookup Node and return successor or predecessor Node
    }
}
```

Q5. Store count of each elements in HashMap, then from 1 to n, return first key with value 0. For in-place use midterm solution.