Introduction to Algorithms: 6.006
Massachusetts Institute of Technology — September 26, 2017
Instructors: Jason Ku, Muriel Medard, and Silvio Micali — Problem Set 3

Problem Set 3

All parts are due on October 3, 2017 at 11:59PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on alg.csail.mit.edu .

Problem 3-1. [20 points] **Binary Search Tree Practice**

(a) [5 points] **Traversal:** A common operation on a binary search tree is to print its elements in sorted order. This can be done using an *in-order traversal*: first find and print the minimum element of the tree (e.g. with `find_min`), and then repeatedly find and print the next element (e.g. with `find_next`). Because `find_min` and `find_next` each take $O(h)$ time, where h is the height of the tree, one can naively calculate a $O(nh)$ upper bound on the time to print all n elements using in-order traversal. Show that in fact, in-order traversal requires at most $O(n)$ time.

(b) [10 points] **Property Checking:** Given the root node of a binary tree, describe $O(n)$ time algorithms that evaluate whether the tree satisfies each of the following properties. Assume for this problem that the only data stored at a node is the node's key, and pointers to its parent, left child, and right child.

- **BST Property:** the key of every node is greater than or equal to every key in the node's left subtree, and less than or equal to every key in the node's right subtree.
- **AVL Property:** every node's left and right subtrees differ in height by at most one.

(c) [5 points] **Make AVL:** Consider the worst case binary search tree with no branching containing the keys $\{1, 2, 3, 4, 5, 6, 7\}$, with key 7 at the root. Show how to transform it into a binary search tree satisfying the AVL Property by performing a sequence of left and/or right rotations.

Problem 3-2. [30 points] **Consulting**

Briefly describe systems that can help each of the following clients quickly solve their problems. By "quickly", we mean at most linear with respect to the size of outputs and at most logarithmic with respect to other problem parameters. For example, if you need to return k elements from a set of n elements, you should use at most $O(k + \log n)$ time to do so.

(a) [10 points] **Secured Shipping:** Fred runs the FredEx shipping company. Throughout the day he receives shipments to his warehouse, each shipment contained in identical regulation-sized packaging. As packages arrive, scanners read each package's ID, address, and the value of its contents. This data must be quickly entered into an inventory. Fred has hired a trucking company to pick up and deliver the packages. Each day, the trucking company sends two trucks. One is a regular truck that has very large capacity and can easily hold an entire day's inventory. The other truck is smaller, but well armored and secure. It's cargo is shared with other companies, so Fred only knows how many packages it will be able to carry when it arrives. Each day, Fred wants to ship as many of the most valuable packages from his inventory as he can inside the armored truck, and ship the remainder in the regular truck. Help Fred design his inventory so he can quickly determine which items to ship in the armored truck each day.

(b) [10 points] **Cloud Computing:** Bev Jezos runs a cloud computing company that rents server time to customers around the world. She has a fixed number of servers, each with a fixed amount of bandwidth. Many times per second, some customer makes a request to the system for a fixed amount of bandwidth to be distributed evenly across a requested number of servers, for a fixed amount of time. Each server holds a reference to its record in the system and is responsible for telling the system when the server's available bandwidth changes. Help Bev design a system that will quickly allow her to either request bandwidth on available servers, or to respond that there are not enough servers available to fulfill the request.

(c) [10 points] **Holiday Sales:** Holly Mark runs a website that sells cheap gifts targeted at office holiday gift giving. She wants to make it easy for shoppers to stay within budget. Each holiday season, merchants can upload price and product information for new gift items, while customers can ask for listings of a desired number of the most expensive items they can buy, below a certain price. Unfortunately, she currently stores all the product data in an unsorted array. This is great for her merchants as they can upload new products quickly; but her customers are frustrated because they have to wait a long time for product listings to load. Help Holly improve her database to quickly satisfy both merchants and customers.

Problem 3-3. [50 points] **Transaction Log:** Barren Wuffett owns a small investment firm that trades a single volatile mutual fund. Hundreds of times a second, brokers buy and sell shares of the fund at different prices as the price dramatically fluctuates throughout the day. At any given time, Barren wants to know how many transactions were traded that day within a given price range. Help Barren design his transaction log; maybe he will share his profits with you! The transaction log must support three operations:

- `add_transaction`: adds a single transaction into the log containing the the number of shares traded at a given price. A negative price means a broker sold shares, and a zero or positive price means shares were purchased.
- `sold_in_range`: for a given inclusive price range, return the total number of shares that brokers sold at a price within the given range.
- `bought_in_range`: for a given inclusive price range, return the total number of shares that brokers bought at a price within the given range.

You want to support all of these operations in $O(\log n)$ time for a transaction log containing n transactions, so you decide to use a **single** modified AVL tree to store all the transaction information. Code implementing an AVL tree is provided in the problem set template. When describing algorithms, you may describe in terms of any of the BST or AVL operations discussed in class.

(a) [8 points] **Augmentation:** The keys in our modified AVL tree will be `Transaction` objects, not integers, containing additional information that might be useful. For example, BST nodes are often augmented to store properties of the subtree rooted at the node, such as: subtree height or skew (already in AVL); number or sum of keys in the subtree; subtree min, median, mean, max, etc. In addition to the price and number of shares for each transaction, describe any additional information you might want to store at a node. Would you need to update this information if the tree changes shape? You may want to think about the other parts of this problem before answering.

(b) [3 points] **Comparison:** In order to store `Transactions` in a tree satisfing the BST Property, we need to know how to compare them. Describe how you want your modified AVL tree to order the `Transactions` it contains.

(c) [7 points] **BST Range:** Let p be a node of a binary search tree containing key k_p, and let $[a, b]$ be an inclusive query range. If p's key is outside the query range ($k_p < a$ or $b < k_p$), it is not difficult to show that at most one of it's left or right subtrees can contain keys in the query range. If p's key is inside the query range ($a \leq k_p \leq b$), let d be a descendant of p whose key k_d is also inside the query range; and let ℓ and r be the left and right child of d respectively. Of the subtrees rooted at ℓ and r, show that at most one of them can contain keys both inside and outside the query range.

(d) [7 points] **Summing Shares:** Describe an algorithm to return the number of shares sold within a given inclusive price range using an augmented AVL tree. If the number of shares output by the algorithm is k, there are many algorithms that can output a correct result in $O(k + \log n)$. We are looking for a $O(\log n)$ algorithm, that does not depend on k. **Hint:** use the result from part (c) to help your analysis.

(e) [25 points] **Implement:** Complete the `Transaction` and `TransactionLog` Python class templates provided to implement the transaction log you described above. Please submit code for this problem in a single file to alg.csail.mit.edu. ALG will import only your class implementations of `Transaction` and `TransactionLog`; do not include classes `BST` or `AVL`.

```
from AVL import AVL

class Transaction():
    def __init__(self, shares, price):
        self.shares = shares
        self.price = price
        ###############################################
        # Part (a): Add any additional storage here #
        ###############################################

    def __lt__(self, other):
        "Evaluate comparison between two transactions: (self < other)"
        ##########################
        # Part (b): Implement me #
        ##########################
        return True               # return a boolean

    def __str__(self):
        return str(self.shares) + '@' + str(self.price)

class TransactionLog(AVL):
    def add_transaction(self, shares, price):
        "Adds a transaction to the transaction log"
        super().insert(Transaction(shares, price))

    def update(self):
        "Augments AVL update() to fix any properties calculated from children"
        super().update()
        ##################################################
        # Part (a): Add any additional maintenence here #
        ##################################################

    def sold_in_range(self, range_min, range_max):
        "Returns the number of shares sold within an inclusive price range"
        if self.key is None:
            return 0
        count = 0
        ##########################
        # Part (d): Implement me #
        ##########################
        return count

    def bought_in_range(self, range_min, range_max):
        "Returns the number of shares bought within an inclusive price range"
        if self.key is None:
            return 0
        count = 0
        ##########################
        # Part (d): Implement me #
        ##########################
        return count
```