Comp 160: Algorithms **Recitation 6**

1. We want to modify a BST to allow having repeated items (say, we want to store TAs by their grade). We modify the main invariant as follows: for any item, all items in the left subtree are less than **or equal** to it, and all items in its right subtree are strictly greater than it.

   (a) As a warm-up, insert the following TAs into the modified tree: (4,EJ), (42,Bruce), (15,Carl), (15,Naoki), (16,Meguna), (8,Anthony), (4,William), (42,Kevin), (16,Radhika), (23,Ashley), (42,Cole), (23,Maria), (4,Brian), (15,Josh), (8,Robby), (8,Mert), (16,Rosa), and (23,Bruce). Draw the resulting tree

   (b) Devise an algorithm for retrieving **all** copies in the tree whose grade is g. Runtime must be proportional to the depth of the tree. First, describe the steps of your algorithm.

   (c) Justify briefly (1-3 sentences) the correctness of your algorithm.

   (d) What is the runtime of your algorithm? Justify briefly (1-3 sentences) the runtime of your algorithm.

   (e) Is it possible to now use AVL rules (or a variation) to maintain the tree balanced? Why or why not?

   (f) Say that we consider a different invariant: instead of storing copies with the same key always to the left, we flip a coin and store it to the left with 50% probability (otherwise we store it in the right). Say that now we insert n objects in the tree.

   Devise an algorithm for retrieving **all** copies of a specified item under the modified tree. What is the runtime?

   (g) Can you design a better way to handle duplicate keys? What is the takeaway?

2. Let $S = \{s_1, \ldots, s_n\}$ be an set containing real numbers.

   (a) The *min gap* is defined as $\min_{i \neq j} |s_i - s_j|$ (in other words, the two numbers that are closest to each other). Give a very simple algorithm to compute the min gap in $\Theta(n^2)$ time. For simplicity, you can assume that numbers are stored in an array, so $S[1] = s_1$, and so on.

   (b) Can you use some observations to reduce the computation time to $O(n \log n)$? Look for some property that s_i and s_j would have.

   (c) After you spent $\Theta(n \log n)$ time, we insert a new element into S. Can we update the min gap efficiently? What if we inserted several numbers one after each other?

   (d) Let's try solving this problem using an augmented tree. We will store all numbers in a balanced tree (such as AVL). In order to determine what to augment the tree with, we need to characterize the solution. How can a solution look like? In other words, could it be that both s_i and s_j are to the right of the root? can both be to the left of the root? Are there other possible cases?

(e) Using the previous answer let's choose how to augment the tree. Imagine you are at the root. You have access to the element of S that is stored there. Also, you can ask questions to your left and right children (for example, you could ask *how many elements are in your subtree?*), but you cannot look further down in the tree. What questions do you need ask to your children to compute min gap? Can we use augment the tree and store that information?

(f) Draw a balanced tree with 5 nodes, for each node show the augmented attributes.

(g) Say that we have a tree augmented the tree with that information. Describe how you would handle an insertion. Make sure to describe how to update any augmented attribute.

(h) Can your structure handle deletions easily? If yes, explain. If not, can you handle deletions by further extending your data structure?

Additional practice questions:

3. **Challenge:** say you want to solve the min gap problem by augmenting the tree, but you are only allowed to store the subtree size at every node. How would you do so? Would this structure work for deletions?

4. Given an array of distinct numbers, define a "flip" to be a pair of numbers such that the smaller one is to the right of the larger one. For example, in the array $[3, 2, 1]$, there are three flips (3 is flipped with both 1 and 2, and 2 is flipped with 1). Give an algorithm to count the number of flips in a given array in $O(n \log n)$ time.

   Note: there are (at least) three different strategies that can be used to solving the problem. One of them uses a technique learned in the first block of the semester. Another technique uses tools explained this week, and the third one we will discuss in the future. Gotta catch them all!

5. (binary tree recursion practice) **Leetcode question 101**: Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center). Leetcode gives bonus points if you could solve it both recursively and iteratively. We give bonus points for a good algorithm structure, and proof of correctness.

6. **Challenge**: Say that you have an AVL tree with n elements. What is the possible range of ranks that the root can have? Express your solution as a function of n (that is, can the root of an AVL tree be the smallest element, regardless of the value of n? Can it have rank $n/2$? what about other ranks?)