Problem Set 3

Problem 3-1

(a) Answer: Min- and max-heaps enable constant-time retrieval of their min and max elements, respectively, but finding the maximum element in a min-heap or the minimum element in a max heap is $O(n)$. BSTs allow for $O(log_2 n)$ retrieval of both the minimum and maximum element, but only if they are balanced. In the case of extremely unbalanced trees, these operations are $O(n)$. AVL trees are balanced BSTs that enable retrieval of both minimum and maximum elements in $O(log_2 n)$ time, so the answer is AVL.

(b) Answer: Insertion into an AVL tree takes $O(log_2 n)$ time.

(c) Answer: Retrieving the minimum key an AVL tree takes $O(log_2 n)$ time.

(d) Answer: Retrieving the maximum key an AVL tree takes $O(log_2 n)$ time.

(e) Answer: $Rank(h) - Rank(l) + 1$. Just imagine an array $1, 2, 3, 4, 5$. Then $Count(3, 5) = 3$, $Rank(5) = 5$ and $Rank(3) = 3$, so $Count(3, 5) = Rank(5) - Rank(3) + 1$.

(f) Answer: $Rank(h) - Rank(l)$. If we remove 3 from the previous array, we have $1, 2, 4, 5$. Now $Count(3, 5) = 2$, $Rank(5) = 4$ and $Rank(3) = 2$, so $Count(3, 5) = Rank(5) - Rank(3)$.

(g) Answer: $Rank(h) - Rank(l)$. If we remove 5 from the original array, we have $1, 2, 3, 4$. Now $Count(3, 5) = 2$, $Rank(5) = 4$ and $Rank(3) = 3$, so $Count(3, 5) = Rank(5) - Rank(3) + 1$.

(h) Answer. $Rank(h) - Rank(l)$**.** If we remove both 3 and 5 from the original array, we end up with $1, 2, 4$. Now $Count(3, 5) = 1$, $Rank(5) = 3$ and 2, so $Count(3, 5) = Rank(5) - Rank(3)$.

(i) Answer. Choice 4, the number of nodes in the subtree rooted at $node$. This information can be updated in constant time as $node.left.\gamma + node.right.\gamma + 1$. In addition, this information can be used to calculate rank of $node$. To do so, initialize $rank = node.left.\gamma$ if $node.left$ exists, otherwise initialize $rank = 1$. Then travel up the three from $node$. Anytime you go left up the tree to a parent node $parent_L$, increment $rank$ by $parent.left.\gamma + 1$. Stop when you reach the root. This take time $O(log_2 n)$.

(j) Answer: You need $O(log_2 n)$ bits to store a number between 0 and n.

(k) Answer: A leaf node as $\gamma = 1$.

(l) Answer: $N_3.\gamma = 3$.

(m) Answer: $N_2.\gamma = 6$.

(n) Answer: $N_1.\gamma = 10$.

(o) Answer: $Insert$ and $Delete$ only needs to satisfy the BST property, so they do not need to be modified. But once a node is inserted or deleted, separate procedures, namely $Rebalance$, $Left - Rotate$, and $Right - Rotate$, are called to maintain the AVL property. These procedures can modify the tree in ways that change the value of γ, and so must be modified to maintain both the AVL property and the γ rep invariant.

(p) Answer: $Rank$ is similar to $Insert$ or $Delete$. It travels down the AVL tree until if finds the location in the tree that corresponds to a given value. Like those procedures, its running time is bounded by the height of the tree, which is $O(log_2 N)$**.**

(q) Answer: The procedure LCA takes in a tree, a lower value l, and a higher value h. It travels down the tree until it reaches a node whose key is between l and h . It then returns this node. If no such node exists, it returns null. If you visualize this process, you'll see that this finds the lowest common ancestor in the tree.

(r) Answer: At most, LCA has to traverse down the entire tree. This takes $O(log_2 n)$ in a balanced AVL tree.

(s) Anwer: $Node-List$ is called for each of the L nodes added. Running time is $O(1)+O(L)$.

(t) Answer: $List$ makes one call to LCA and one call to $Node - List$. As show in parts (r) and (s), these take $O(log_2 N) + O(L)$ in total.

(u) Answer: First observe that LCA is guaranteed to terminate in $O(log_2 N)$ steps, because it descends one level in the AVL tree at every step. Now assume that a non-empty set of nodes exist whose key values are between l and h. The loop condition guarantees that the node returned is the highest node in this set. Call this node n_R. This implies that nodes with values l and h are located in left and right subtrees of the subtree rooted at n_R. Thefefore, any lower node in the tree cannot be a common ancestor of nodes with values l and h. So, n_R is in fact the $lowest$ common ancestor of nodes with values l and h.

Now assume there are no nodes with key values between l and h. If no node in the tree satistfies the condition in the until block, LCA will traverse left down the tree until it reaches a leaf, at which point it returns $null$.

Problem 3-2

(a) Answer: intersects has the highest CPU usage.

(b) Answer: intersects is called 187590314 times.

(c) Answer: 1 and 2. Left and right endpoints of horizontal wires are points of interest.

(d) Answer: 1. Wires are added to the range index when the sweep line intersects their left endpoint.

(e) Answer: 2. Wires are removed from the range index when the sweep line reaches their right endpoint.

(f) Answer: 4. Nothing happens.

(g) Answer: 3. A range index query is performed.

(h) Answer: 2. Horizontal wires are added to the range index when the sweep line hits their left endpoing, and removed when the sweep line hits their right endpoint. Given that the sweep line moves from left to right, a horizontal line is in the range index for if and only if the sweep line is currently intersecting the horizontal line.

(i) Answer: The y coordinate y_1 is added as the key in the KeyWirePairs.

(j) Answer: count is now users the most CPU.

(k) Answer: count is called 20,000 times.

(l) Answer: see https://github.com/claytonm/6006/tree/master/ps3/circuit2/circuit2.py