CMSC 420 (0201) - Midterm Exam 1

Problem 1. (10 points)

(a) (5 points) Show the final tree that results from performing `splay(5)` to the tree shown below. For assigning partial credit, indicate which nodes are involved in your zig-zig, zig-zag, and zig rotations.

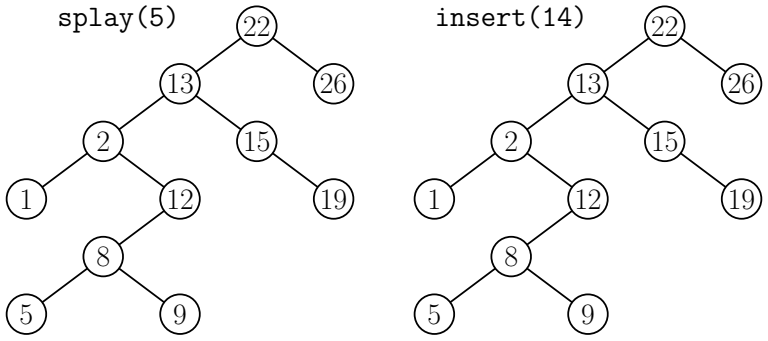


Figure 1: Splay tree operations.

(b) (5 points) Show the final tree that results from performing `insert(14)` to the tree shown below. For assigning partial credit, indicate which nodes are involved in your zig-zig, zig-zag, and zig rotations

Problem 2. (35 points) Short answer questions. Unless requested, explanations are not required, but may always be given to help with partial credit.

(a) (4 points) You have an inorder-threaded binary tree with n nodes. Let u be an arbitrary non-leaf node in this tree. **True or False**: There must be at least one thread that points into u.

(b) (5 points) You perform a preorder traversal of a *full binary tree* with n nodes. You have a counter that is incremented whenever you visit a leaf node and decremented whenever you visit an internal node. As function of n, what are the maximum and minimum values that this counter might achieve at any point in the traversal? (This is taken over all possible full binary trees with n nodes.)

(c) (5 points) You build a union-find data structure for a set of n objects. Initially, each element is in a set by itself. You then perform k union operations, where $k < n$. Each operation merges two different sets. Can the number of union-find trees be determined from k and n alone? If not, answer "It depends". If so, give the number of trees as a function of k and n.

(d) (5 points) Your boss asks you to program a new function for your leftist heap. Given a leftist heap with n entries, the operation returns the *third smallest* item in the heap (without modifying its contents). What is the minimum number of heap entries that you might need to inspect to be certain that the third smallest item is among them?

(e) (4 points) You have just performed an insert into a 2-3 tree of height h. What is the maximum number of split operations that might be needed as a result?

(f) (4 points) You have just inserted n (distinct) keys into a treap. As a function of n, what is the probability that the smallest of the n keys is located at the root of the tree?

(1) 0 (That is, it cannot happen)
(2) Roughly $1/n$ (By "roughly", we mean "up to constant factors")
(3) Roughly $1/(\log n)$
(4) Roughly $1/2^n$
(5) Roughly $1/(n!)$

(g) (4 points) You have a skiplist containing n keys, where n is a very large number. Suppose you perform a find operation. The search algorithm visits one or more nodes at each level of the structure. How many nodes do you expect to visit at level 4 of the search structure?

(1) None of them
(2) $O(1)$
(3) $O(\log n)$
(4) $O(n/(2^4))$
(5) All of them

(h) (4 points) You have just inserted a key into an AA tree having h levels. What is the maximum number of skew operations that might be needed as a result? (Here we are only counting skew operations that have an effect on the structure, in the sense that a rotation is performed.)

Problem 3. (15 points) In this problem, we assume that we are given a tree-based heap structure, which is represented by a binary tree (not necessarily complete nor leftist). Each node `u` stores three things, its priority, `u.key`, and the pointers to its subtrees, `u.left` and `u.right`. The keys are min-heap ordered (that is, a node's key is never smaller than its parent). There are no NPL values.

(a) (5 points) Present pseudocode for a function `swapRight(Node u)` which is given a pointer to the root of a tree. It traverses the right chain of this tree and swaps the left and right subtrees of all nodes along this chain (see the figure below). It returns a pointer to the resulting tree. For full credit, your function should run in time proportional to the length of the right chain in the tree.

(b) (10 points) Present pseudocode for a function `swapMerge(Node u, Node v)`, which is given pointers to the roots of two trees. It merges the right chains of these two trees according to min-heap order, and then performs `swapRight` on the resulting tree (see the figure below). It returns a pointer to the resulting tree.

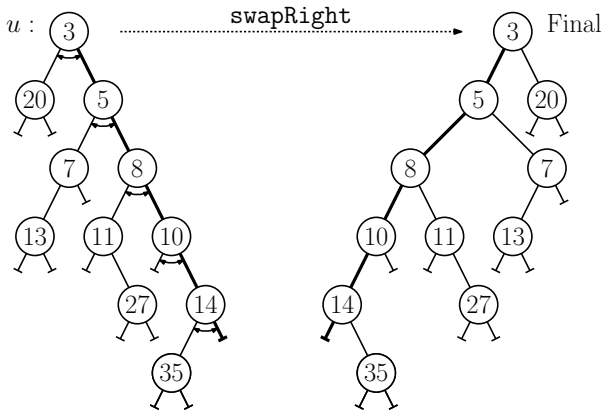


Figure 2: The function `swapRight`.

For full credit, your function should do this in one pass. That is, it is allowed to recurse down and return up, but that is all. For half credit, you can do it in two passes (one pass to merge and one pass to swap). You may use `swapRight` from (a).

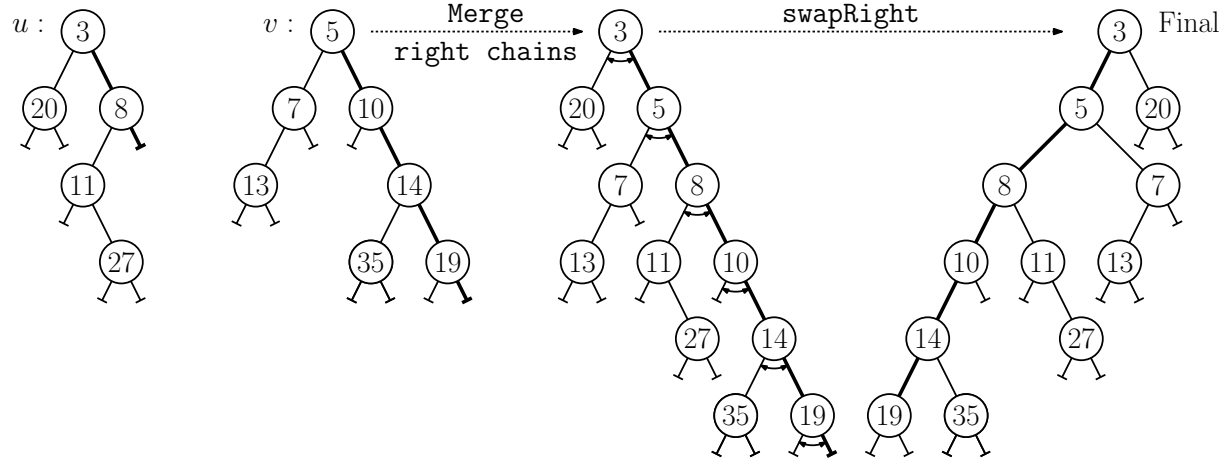
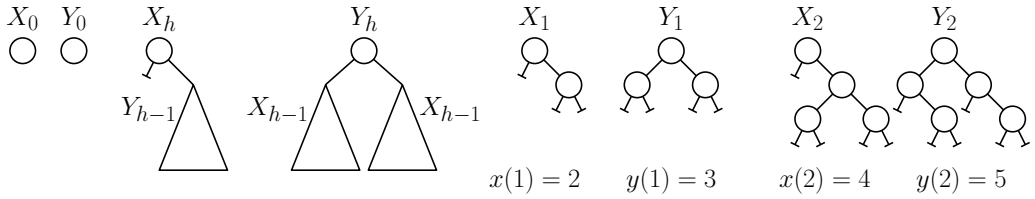


Figure 3: The function `swapMerge`.

Problem 4. (25 points) This problem involves the analysis of two new tree structures, called the *X-tree* and *Y-tree*, which are defined recursively in terms of each other. Let X_h and Y_h denote the X-tree and Y-tree of height h, respectively. X_0 and Y_0 are both defined to be a single node. For $h \geq 1$, X_h consists of a root node whose left child is `null` and whose right child is Y_{h-1}. The tree Y_h consists of a root node whose left and right children are both X_{h-1} (see the figure below).



(a) (3 points) Draw a picture of X_3 and a picture of Y_3. (You don't need to draw the null pointers.)

(b) (8 points) Let $x(h)$ and $y(h)$ denote the number of nodes in X_h and Y_h, respectively (see the figure above). Clearly $x(0) = y(0) = 1$. Assuming $h \geq 1$, give a formula that expresses $x(h)$ as a function of $y(h-1)$, and a formula that expresses $y(h)$ in terms of $x(h-1)$. **Hint:** The formulas are simple and do not involve any summations.

(c) (4 points) Assuming $h \geq 1$, give a formula that expresses $x(h)$ as a function of $x(h-2)$. **Hint:** The formula is simple and does not involve any summations.

(d) (10 points) Prove that if h is even, $x(h) = 3 \cdot 2^{h/2} - 2$.

Problem 5. (15 points) This is an extension of the Homework 1 problem on the *dual stack*, which stores two stacks in a single array. Recall that we are given an array `A` of length m, one of the stacks starts at index 0 and grows upwards and the other starts at index $m-1$ and grows downwards. Initially, the array has space for two entries ($m = 2$), and both stacks are empty.

Assuming we have space, each single operation has an actual cost of +1 unit. Whenever we run out of space, we expand the array as follows. Let n_1 and n_2 denote the numbers of elements in the two stacks. We allocate a new array of size $m' = w \cdot \max(n_1, n_2)$, for some integer constant w. (In Homework 1, $w = 3$, but here it will be a parameter that we can adjust). The actual cost of the expansion is equal to the total number of elements copied, that is, $n_1 + n_2$ (see Fig. 4).

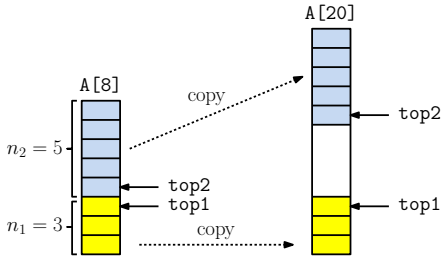


Figure 4: Expanding dual stack, where $w = 4$. When we run out of space, we allocate a new array of size $w \cdot \max(n_1, n_2) = w \cdot 5 = 20$. The actual cost is +8 for the copy and +1 for the final push.

(a) (3 points) Suppose that we have just performed a reallocation. Our current array is of size $n_1 + n_2 = m$, and our new array is of size $m' = w \cdot \max(n_1, n_2)$. What is the minimum number of stack operations until the new array needs to expand again? Express your answer as a function of some combination of m, m', and/or w. Briefly explain.

(b) (8 points) Derive the smallest constant c such that the amortized cost of our expanding dual stack is at most c and prove the correctness of your answer. (The value of c will depend on w.)

(c) (4 points) How small can w be before the amortized cost is no longer bounded by a constant? Briefly explain. (Remember that we required that w is an integer.)