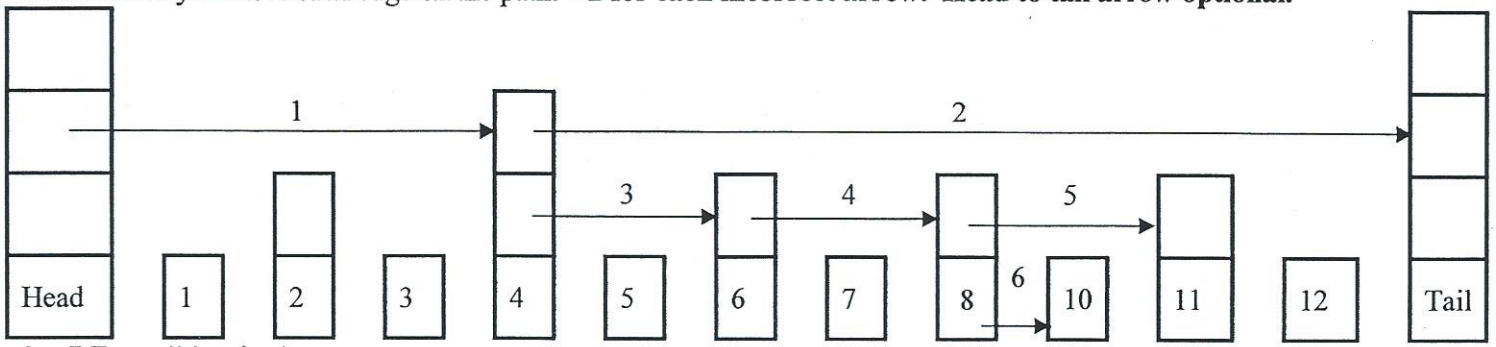
ECS 60 **Midterm #1 Key** **Winter 2015**

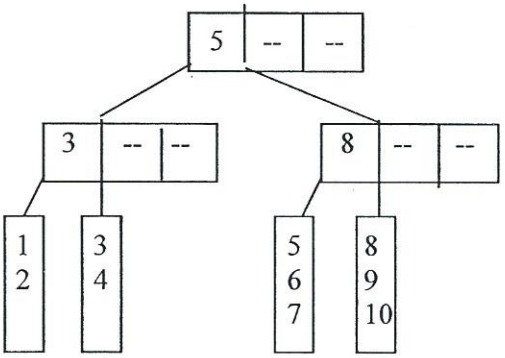
1. Skip Lists (10 points)

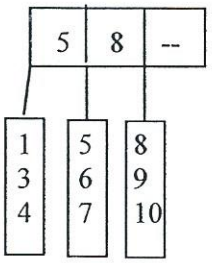
On the diagram below (not the one you drew for part a), show the path searched when a find(10) is executed. Consecutively number each edge on the path. **–2 for each incorrect arrow. Head to tail arrow optional.**



2. BTrees (20 points)

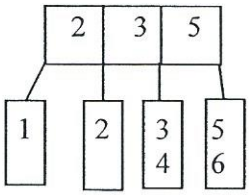
a) (10 points) Given the following 4-ary BTree with L = 3, draw the BTree that would exist after a delete(2). Sean's Rules for delete: look left, look right, try to merge left, try to merge right.

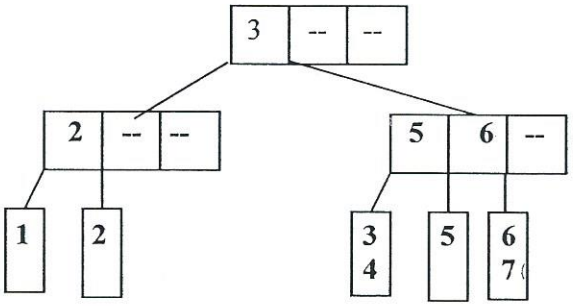




1, 3, 4 leaf = 3 points. One internal node = 4 points. Root 5, 8, and "—" = 1 point each.

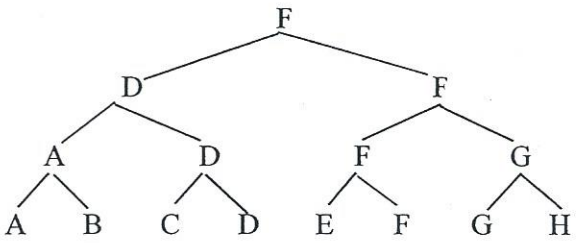
b) (10 points) Given the following 4-ary BTree with L =2, draw the BTree that would exist after an insert of 7. . Sean's rules for insert: look left, then look right, then split. When splitting, the new node will contain the larger values, and contain at least as many as the old node.





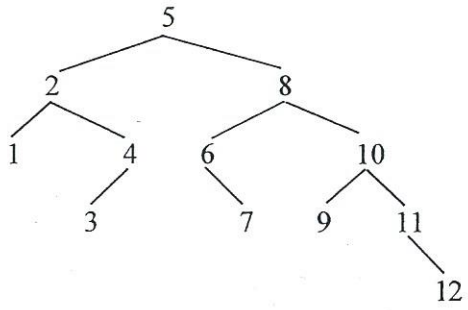
Split leaf, split internal node, new root = 2 points each. Internal 2, 3, 5, 6 = 1 point each.

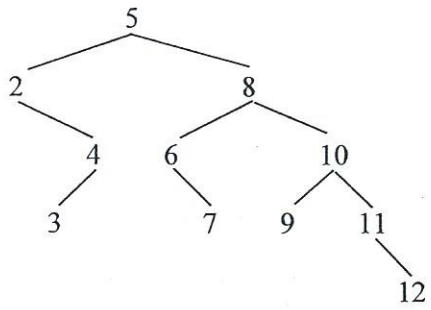
3. (15 points) Given a binary tree that contains the results of a 64-team single elimination tournament, I want to find the first team of the six teams that F beat. Note the diagram below only shows the upper parts of the much larger tree. My code only follows the path of F's victories down from the root, so it is not a true tree traversal. Nonetheless, which tree traversal code would my code most closely parallel and why? Only one or two sentences are needed with 8 points for the correct traversal, and seven points for an explanation of your choice.

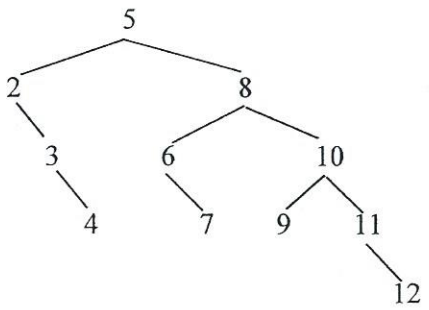


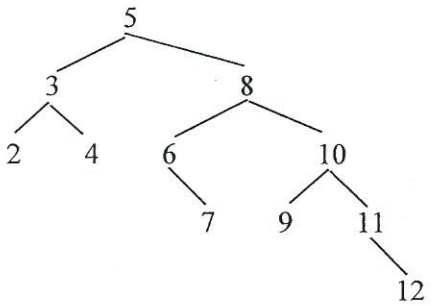
It would be similar to a post order traversal because it would have to look at both children first to find the loser and determine the correct path to take.

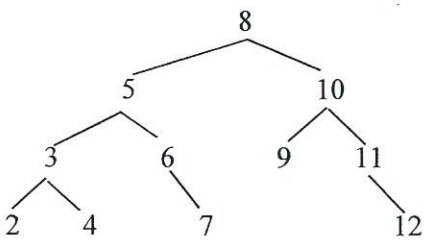
4. (20 points) Given the following AVL tree, draw a representation of the tree after a delete(1) operation has been executed. (You may wish to write intermediate tree(s) to ensure partial credit. Rather than copying parts of the tree that remain the same from the previous intermediate tree, you may simply use dashed lines from the appropriate node to indicate that either the ancestors or descendants remain the same from the previous intermediate tree. However, your final tree must be complete, and not use any dashed lines.) 5 **points for each intermediate tree.**







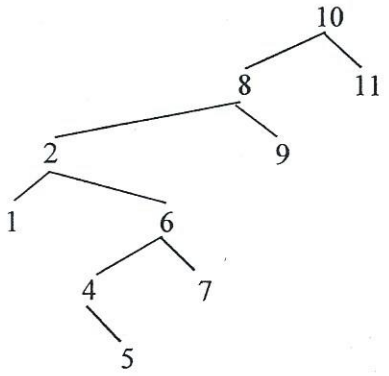


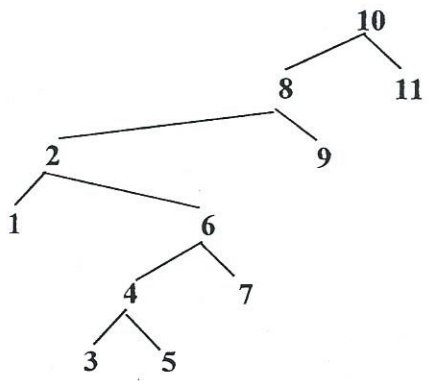


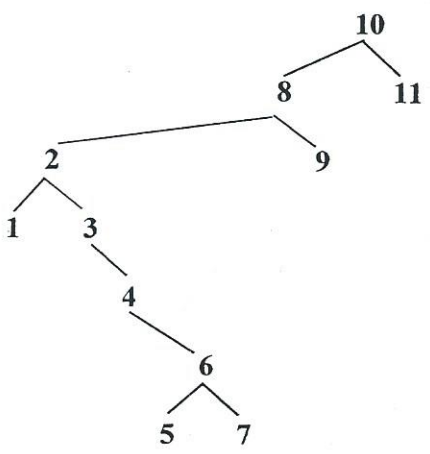
5. (15 points, All or nothing) We expect quite short answers to these questions, i.e., no more than three sentences each. In the BTree program, how dida node know it was the root node?

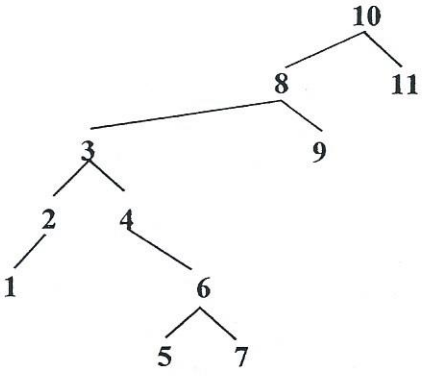
When both its siblings were NULL, or its parent was NULL.

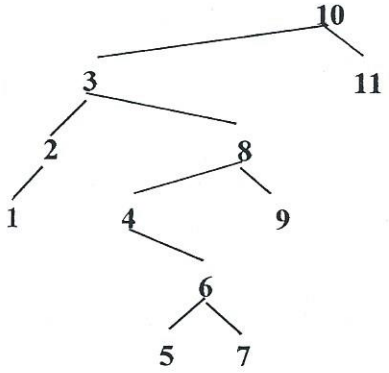
6. (20 points) Given the following splay tree, what would the tree look like after inserting 3. (You may wish to write intermediate tree(s) to ensure partial credit. Rather than copying parts of the tree that remain the same from the previous intermediate tree, you may simply use dashed lines from the appropriate node to indicate that either the ancestors or descendants remain the same from the previous intermediate tree. However, your final tree must be complete, and not use any dashed lines.) **4 points for each correct intermediate tree.**

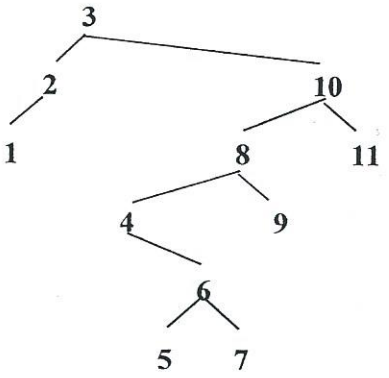












7. In my large corporation, every order has a customer name, a unique invoice number, and as much as 100 bytes of data associated with it. About 20,000 (**D**) invoice numbers are assigned sequentially throughout each day, though the invoices are not necessarily completed in sequential order. Clearly a day's information will fit in RAM, but each invoice is also stored in a database on a hard disk. There are about 400,000 (**M**) invoices a month. My corporation has 100,000 (**C**) customers that buy from us every month. Every day I need to list out the information from the day's invoices in numerical order. Once a month I need to print out statements for each customer with a list of their order information for that month based on information on the disk, and sorted by invoice number. To ensure coherence, I want all the information about an order to be stored in only one place on a disk. Describe the data structure(s) you would use to satisfy these specifications most time efficiently. In particular, you should address the insertion, daily printing, and monthly printout operations. Remember to indicate what keys you are using for sorting. Justify you choices, using big-O notation where appropriate. You should use **C, D,** and **M** in your big-O's, and NOT **n.**

Since inserting and printing invoices are done every day we want to make sure they are fast. I would have a quadratic probing hash table that would permit hashing a customer's name to a unique ID that would be its position within the array. Assuming a load factor a little less than half, then finding a customer's ID would be O(1). I chose a hash table because it has the fastest accesses of all the data structures. Since the daily data fits in RAM we can use a splay tree based on invoice numbers for the daily functions. I cannot use a hash table because the orders need to be listed in order, and a hash table cannot do that. Though I could use a skip list or an AVL tree, we would expect that the invoices would be inserted in close to ascending order. Though we have amortized O(log D) for insertions in a splay tree, we would really expect O(1). The skip list and AVL tree would not take advantage of the temporal locality of the invoice insertions. The daily printing would use a simple inorder traversal O(D).

Besides the splay tree, and hash table, we would have a BTree to store the invoice information on the disk. We use a BTree because it minimizes the number of disk accesses for the operations better than any other data structure. The keys will be a combination the customer ID's as the primary sort, and the year and month as the secondary sort criteria. This means that all of a customer's invoices would be in contiguous leaves sorted by their date. The monthly printing is quite simple. We traverse the hash table to find the current customer IDs, and then combine each with the month and year to form a key for the BTree. We do a form of a ranged find by going to the leaf that should have the month, and then traverse across the leaves until we reach the records for a different customer. Note that a customer's invoices will not necessarily be stored in the correct invoice order, but since there are only four invoices per customer per month on average, sorting is trivial. This process will minimize disk accesses for printing the monthly invoices, and be O(M).