Lab 6 (10 points) **Due date:** November 4, 11:59 PM

Note: To receive ANY credit for these questions, you must:

1. Show all your work (all calculations, all swaps, all rotations, all insertions), circle the final answer (if applicable)
2. Assign each question to a page number on Gradescope. (if you do not assign a page number to each question, the question will receive a zero score)
3. Verify you are answering the questions asked in each problem. Start your answers to each question in a new page (even if there is space left in the previous page) and indicate at the bottom of the page if your answer continues in a second page. All vocabulary follows the descriptions and conventions described in lectures.

Question 1 [2 pts]: Answer the following questions for each of the two trees shown below:

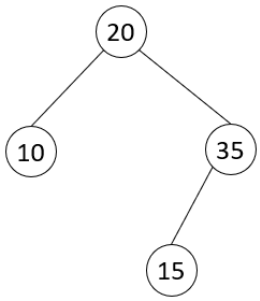
a) What is the height of the tree?

b) What is the maximum number of nodes that could be added to the tree without increasing its height?

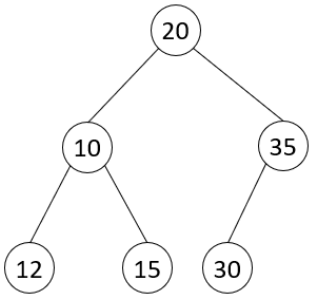
c) Is it a full tree? Is it a complete tree?

d) List the preorder, inorder and postorder traversal of the tree

e) Is it a binary search tree? If your answer is no, rearrange the nodes to make it a binary search tree (draw the new tree)



Tree #1



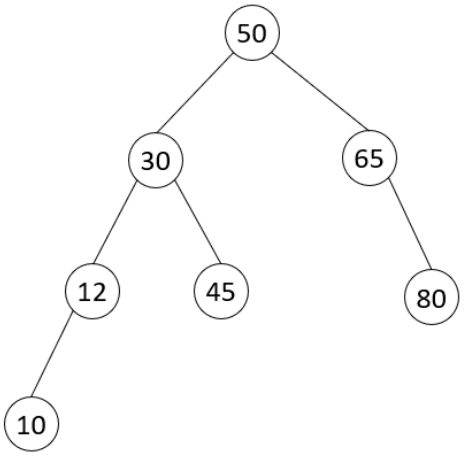
Tree #2

Question 2 [1 pt]: Answer the following questions for the tree shown on the right:

a) Is the tree an AVL tree? (Briefly describe why)

b) If we delete node 45 is the result an AVL tree? (show the tree after deletion and briefly describe why the tree is (or not) and AVL tree)

c) If we delete node 50 is the result an AVL tree? (show the tree after deletion and briefly describe why the tree is (or not) and AVL tree) ** The delete operations are cumulative (45 is no longer in the tree)



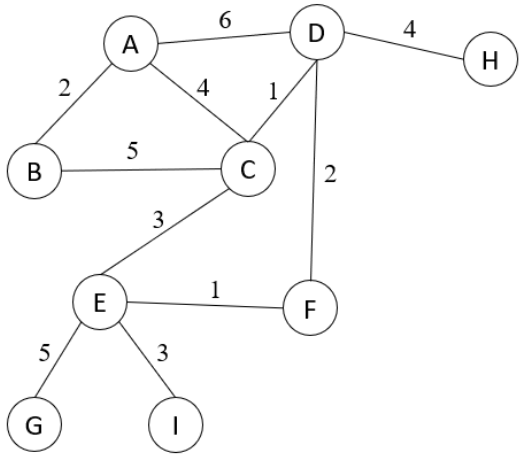
Question 3 [1.25 pts]: Draw the AVL tree obtained when inserting 12, 15, 16, 18, 19 and 17 in that order into an empty AVL tree. If rotations are needed, you must indicate the case of imbalance (as described in lectures) and show the tree before and after each rotation (they are part of the grade).

Question 4 [1 pt]: Draw the min heap that results from inserting 17, 15, 16, 13, 18 and 11 into an empty heap. Then draw the heap that results from performing two calls to deleteMin() in the resulting heap. Don't forget to show your work!

Question 5 [0.75 pts/0.25 each]: Assume you have directed graph with k nodes, where each node corresponds to a number (1, 2, ..., k) and there is an edge from node u to node v if and only if $u < v$. Draw the graph assuming $k = 4$, then answer the following questions:

a) Write the adjacency matrix representation of the graph
b) How many edges are in the graph?
c) If any, how many correct results for topological sort does this graph have?

For questions 6, 7 and 8, use the following undirected weighted graph:



Question 6 [1 pt]: Draw the BFS tree (vertices and tree edges) that results when performing a BFS traversal starting at node C, also draw the DFS tree that results when performing a DFS traversal starting at node G. Include with each tree the traversal path: BFS traversal for BFS tree, and preorder traversal for DFS tree. Follow the alphabetical convention discussed in the video lectures

Question 7 [1 pt]. Run Dijkstra's algorithm in the graph and complete the status of the table when the algorithm is done calculating the shortest path from node *A* to every other node in the graph. Based on that table, what is the shortest path and its cost from node *A* to node *I*? If two nodes have the same shortest known distance, choose a node based on alphabetical order.

Question 8 [1 pt]: Find and draw the minimum spanning tree for the graph given below using both Prim's and Kruskal's algorithms. You must include i) a list with the order in which the edges are added into the MST for each algorithm and ii) the cost of each MST. For Prim's algorithm, use *H* as the starting node.

Question 9 [1 pt]: Assume you have a hash table of size 7. What is the final status of the hash table after inserting the keys 10, 36, 17, 19 and 24 (in that order) assuming collisions are handled by open addressing using:

a) linear probing (+1 probe) and the hash function H(*i*) = (*i*) % *table_size*
b) quadratic probing and the hash function H(*i*) = (*i*) % *table_size*
c) double hashing and using the hash functions H1(*i*) = (*i*) % *table_size* and H2(*i*) = 5- (*i* %5).