Elizabeth Oyebade

MET CS 526-O2

04/19/2022

Problem 1:

Problem 1: Adding the entry with key = 13.

Step 1: Start by checking whether the value of the key is less or greater than root node of the tree.

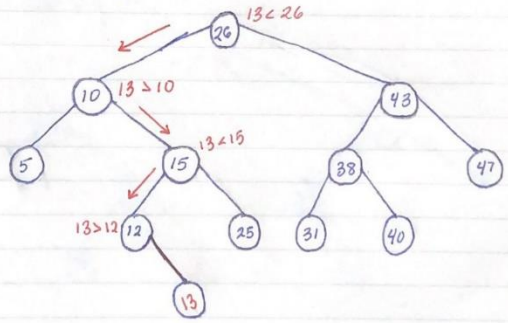
Step 2: Compare the key 13 with the root node 26.
13 < 26, so go to the left subtree.

Step 3: Compare the next key 10 with 13.
So, 13 > 10, go to the right subtree (child of parent key 10)

Step 4: Compare the next key which is 15 with 13.
13 < 15, so go to the left subtree (child of parent key 15).

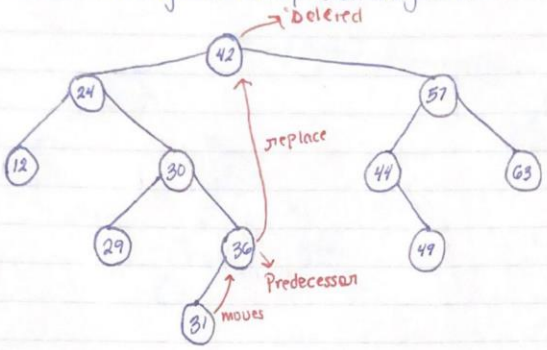
Step 5: Compare the next key which is 12 with 13.
13 > 12, so go to the right subtree. Since key 12 has no right child, insert a node with the key 13 as the child of the key node 12.

The final binary search tree:



Problem 2:

Problem 2: Deleting the entry with key = 42

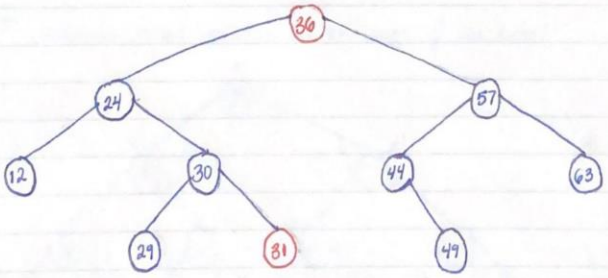


* key 42 have both left child & right child.

Step 1: Replace the key 42 with either its inorder successor or predecessor.

Step 2: Since 36 is the predecessor, replace it with 42 and delete the node key 42.

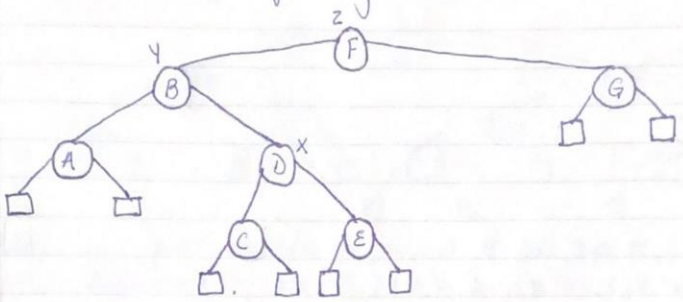
Step 3: Since the node with key 36 has one child (node key 31) before it was moved, node key 31 will become the new right child of node key 30.



* The final binary search tree

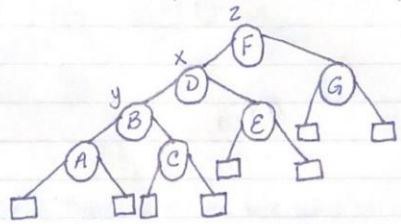
Problem 3:

Problem 3: Given the following unbalanced AVL tree:



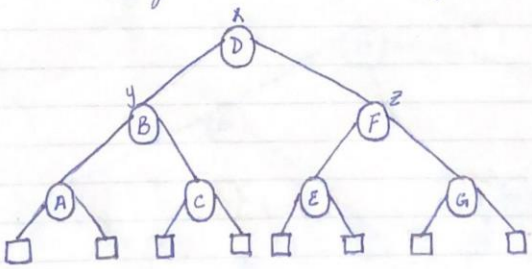
"y" is the left child of "z" and "x" is the right child of "y". Applying a double rotation will result in:

1st rotation: left rotation in the left subtree:



* Applied the Trinode Restructing to bring the tree in balance

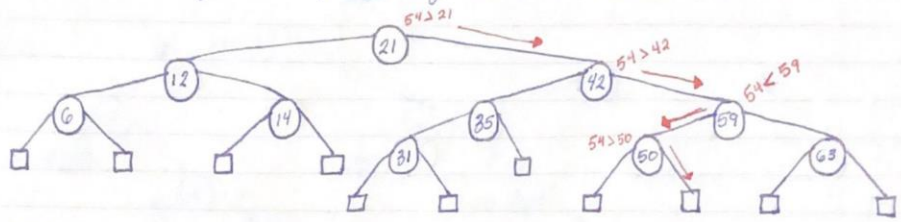
2nd rotation: right rotation at the root of the tree:



* Final AVL tree result

Problem 4:

Problem 4: Inserting an entry with key = 54 on the balanced tree

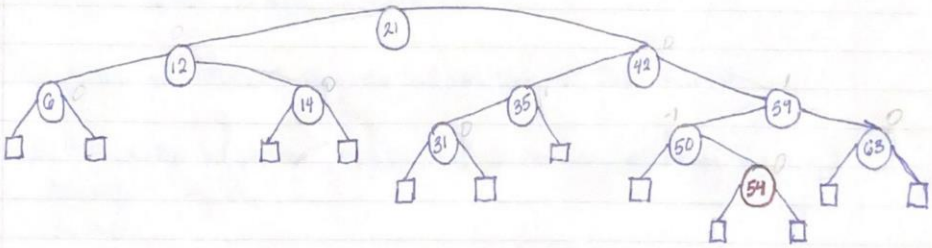


Step 1: Start at the root with the key = 21. 54 > 21, go to the right subtree

Step 2: Compare key 42 with 54. 54 > 42, so go to the right subtree (child of parent key 42).

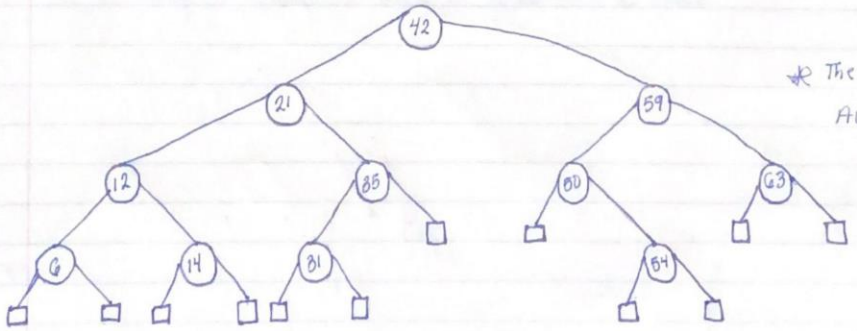
Step 3: Compare key 59 with 54. 54 < 59, so go to the left subtree (child of parent key 59).

Step 4: Compare key 50 with 54. 54 > 50, so go to the right subtree. Since key 50 has no right child, insert a node with key 54 as the child of key node 50.



Step 5: Rebalance the AVL tree since the height of the left subtree from the node key 21 is 3 and from the right subtree is 5. which result in -2.

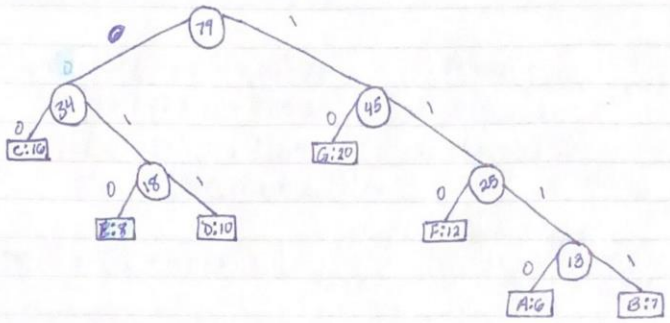
Step 6: Applying a One rotation at the root node.



* The final AVL tree

Problem 5:

Problem 5: Given the following Huffman tree:



1.) Encode the string "BDEGC"

B: 1111, D: 011, E: 010, G: 10, C: 00

Bit Pattern for "BDEGC" using the Huffman tree: 1111011010000

2.) Decode the bit pattern "010111010011" to the original string

010 => E

1110 => A

10 => G

011 => D

Original string for "010111010011" using the Huffman tree is EAGD

Problem 6:

Problem 6:

$P(i,j) = (P(i+1,j) + P(i,j-1))/2$

$P(4,1) = (P(3,1) + P(4,0))/2 \quad \Rightarrow P(4,0) = 0$

$P(3,1) = (P(2,1) + P(3,0))/2 \quad \Rightarrow P(3,0) = 0$

$P(2,1) = (P(1,1) + P(2,0))/2 \quad \Rightarrow P(2,0) = 0$

$P(1,1) = (P(0,1) + P(1,0))/2$

$P(4,1) = (\frac{1}{8} + 0)/2 = \frac{1}{16}$

$P(3,1) = (\frac{1}{4} + 0)/2 = \frac{1}{8}$

$P(2,1) = (\frac{1}{2} + 0)/2 = \frac{1}{4}$

$P(1,1) = (1 + 0)/2 = \frac{1}{2}$

$P(2,4) = (P(1,4) + P(2,3))/2$

$P(1,4) = 1 - \frac{1}{16} = \frac{15}{16}$

$P(2,3) = (P(1,3) + P(2,2))/2$

$P(1,3) = 1 - P(3,1)$

$= 1 - \frac{1}{8} = \frac{7}{8}$

$P(2,2) = (P(1,2) + P(2,1))/2$

$= 1/2$

$P(2,3) = (\frac{7}{8} + \frac{1}{2})/2$

$= 11/16$

$P(2,4) = (\frac{15}{16} + \frac{11}{16})/2$

$= 13/16$

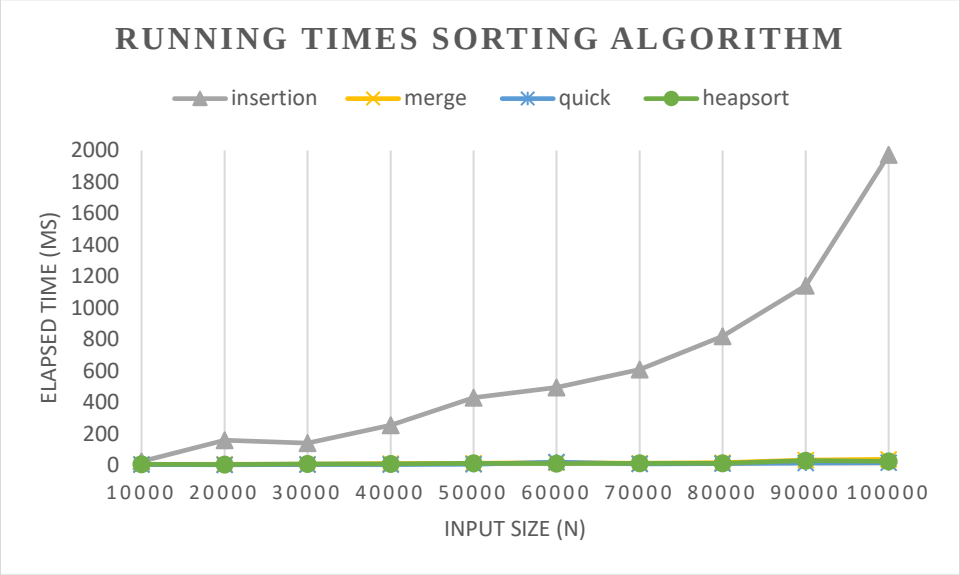
The Probabilities of $P(4,1) = (P(3,1) + P(4,0))/2 = (\frac{1}{8} + 0)/2 = 1/16$

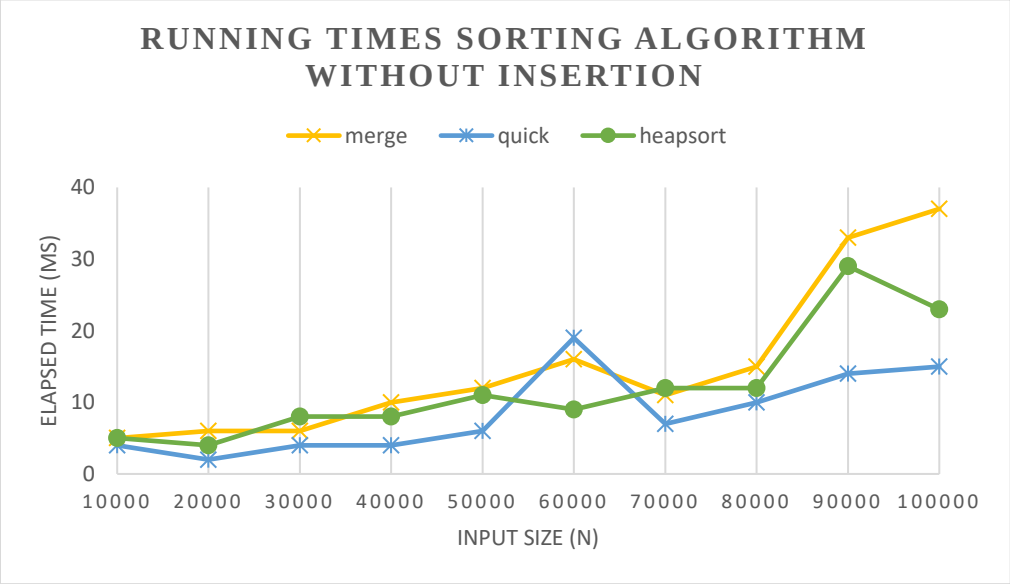
$P(2,4) = (P(1,4) + P(2,3))/2 = (\frac{15}{16} + \frac{11}{16})/2 = 13/16$

Problem 7:

Sorting Algorithms table result:

n *Algorithm*	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
insertion	23	159	139	253	428	494	608	819	1140	1972
merge	5	6	6	10	12	16	11	15	33	37
quick	4	2	4	4	6	19	7	10	14	15
heapsort	5	4	8	8	11	9	12	12	29	23





What I observed while running this program is that the Insertion Sort algorithm takes the longest time, and it increases the most and that is because of the way it's implemented. The insertion sort

running time is resulting in a O(n^2) because all integers to the left of the current integer have be swapped until it can be inserted into the correct place. This process is then being repeated for the next current integer. The table shows an increase in execution time for the insertion sort algorithm which is greater than the linear or O(n log n). The second diagram is showing that the quick sort is faster than merge and heap sort.