Assignment 6 - Trees & Heaps

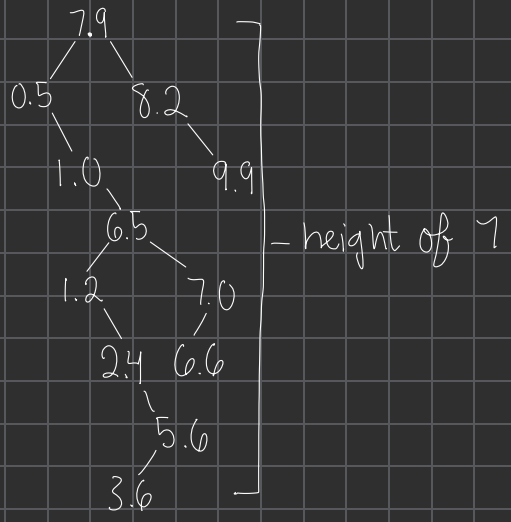
Due April 16 2025

Jayden Cruz

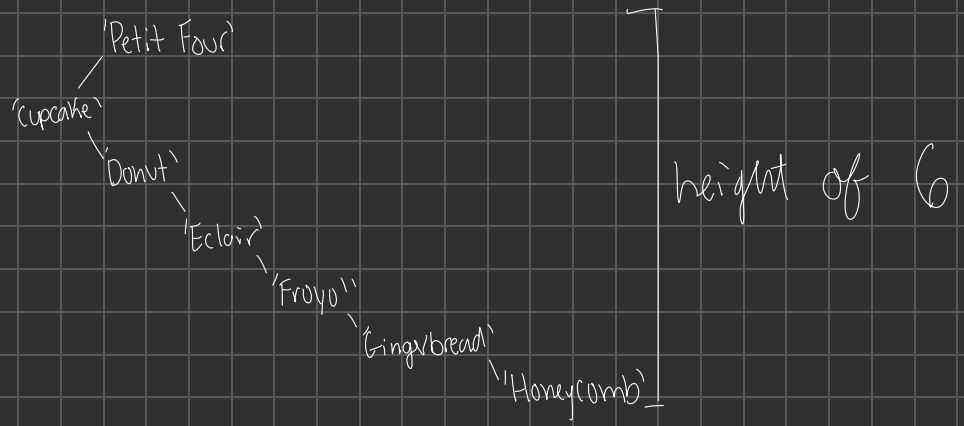
1. (text) Type of Tree [10 points]

R < C < L

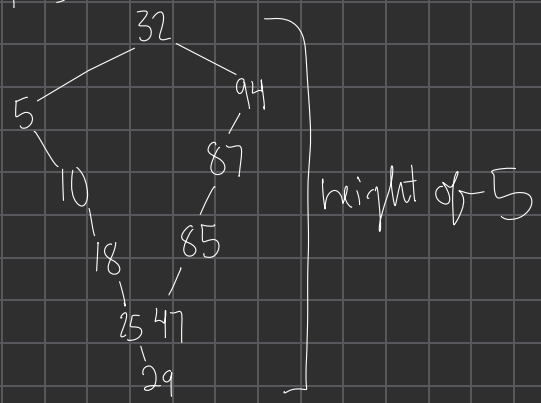
a. [7.9, 0.5, 1.0, 6.5, 8.2, 7.0, 6.6, 9.9, 1.2, 2.4, 5.6, 3.6]



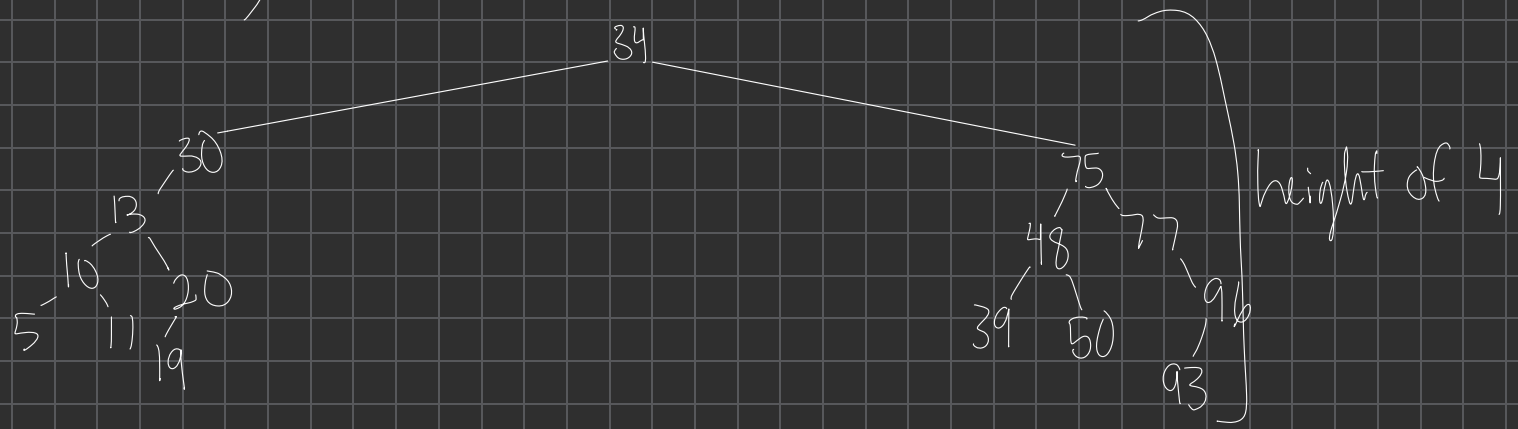
b. ['Petit Four', 'Cupcake', 'Donut', 'Eclair', 'Froyo', 'Gingerbread', 'Honeycomb']



c. [32, 5, 94, 87, 10, 18, 85, 47, 25, 29]



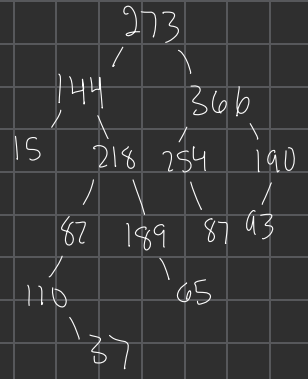
d. [34, 30, 75, 77, 96, 48, 39, 50, 93, 13, 10, 5, 11, 20, 19]



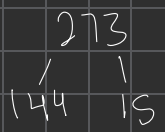
2. (text) BST Traversal [10 points]

c. on original tree [68, 21, 15, 54, 46, 36, 37, 59, 65, 92, 80, 87, 97, 93]

(68+21+(92x2)) = 273
21+15+(54x2) = 144
15+0+0 = 15
54+46+(59x2) = 218
46+36+0 = 82
36+0+74 = 110
37+0+0 = 37
59+0+130 = 189
65+0+0 = 65
92+80+194 = 366
80+0+174 = 254
87+0+0 = 87
97+93+0 = 190
93+0+0 = 93

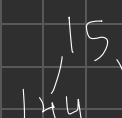


PreOrder: [273, 144, 15, 218, 82, 110, 37, 189, 65, 366, 254, 87, 190, 93]



X not a BST, since not BST can't be AVL

InOrder: [15, 144, 110, 37, 82, 218, 189, 65, 273, 254, 87, 366, 93, 190]



X same

b. no bst -> left subtree < node < right subtree

c. no, not bst so can't be AVL bc AVL is BST + balancing property.

5. (text) Algorithm Analysis (5 points)

initializeCandidates
The initializeCandidates method has a time complexity of O(n) and a space complexity of O(n), where n is the number of candidates. This is because the method iterates through the provided list once, adding each candidate to a HashMap with an initial vote count of zero. Although inserting into a HashMap takes constant time per operation, performing it n times results in linear complexity. The space used grows with the number of entries in the map, which is equal to the number of candidates.

castVote
The castVote method has a time complexity of O(1) and a space complexity of O(1). This method directly accesses and updates the vote count for a given candidate in the HashMap. Since HashMap operations like get and put are constant time on average and no additional data structures are used, both time and space usage remain constant regardless of n, the number of candidates.

castRandomVote
The castRandomVote method has an average-case time complexity of O(1) and a space complexity of O(n), where n is the number of candidates. The method randomly selects a candidate by first converting the key set of the HashMap into a list of size n, which introduces O(n) time and space for that conversion. However, selecting a random index and casting the vote are constant-time operations, making the overall average time complexity close to O(1), with O(n) space due to the temporary list.

rigElection
The rigElection method has a time complexity of O(n) and a space complexity of O(1), where n is the number of candidates. The method first assigns the majority of votes to one specified candidate, setting all others to zero. It then redistributes any remaining votes by iterating through all other candidates again. These two passes through the candidate list each take O(n) time, but no new data structures are created, so space usage remains constant.

getTopKCandidates
The getTopKCandidates method has a worst-case time complexity of O(n log n) and a space complexity of O(n), where n is the number of candidates. This is because it adds all n candidates to a max-heap (priority queue), which is an O(n) operation, and then extracts the top k elements, each extraction taking O(log n) time. The temporary storage for the heap also grows linearly with the number of candidates.

auditElection
The auditElection method has a time complexity of O(n log n) and a space complexity of O(n), where n is the number of candidates. It creates a list of all candidate entries and sorts them in descending order based on vote counts. Sorting takes O(n log n) time, and the space needed for the list is proportional to the number of candidates.