Assignment 6 - Trees & Heaps

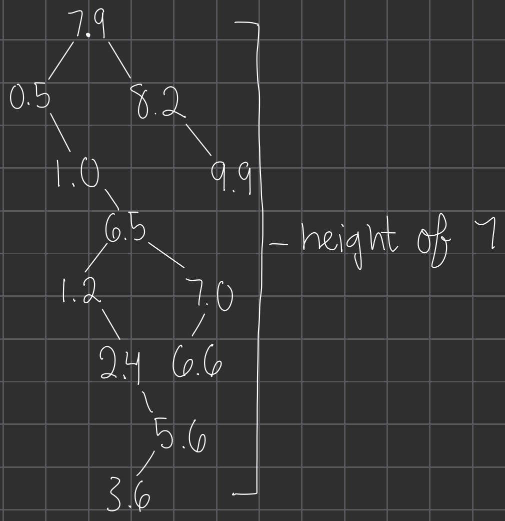
Due April 16 2025

Jayden Cruz

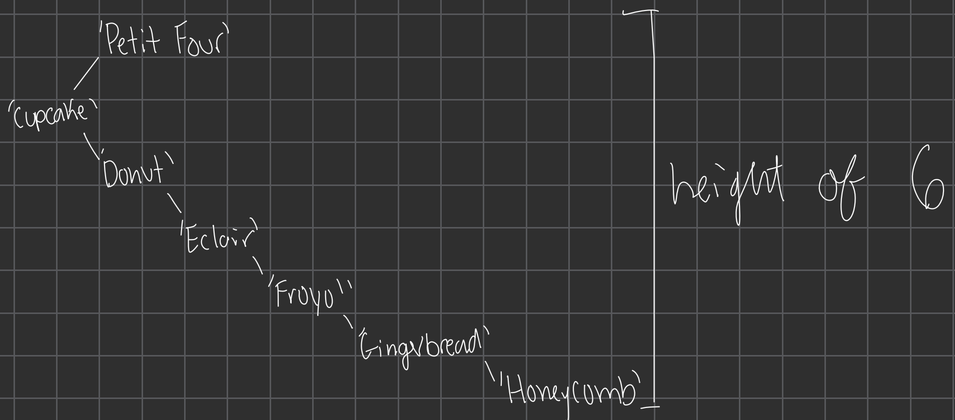
1. (text) Type of Tree [10 points]

R<C<L

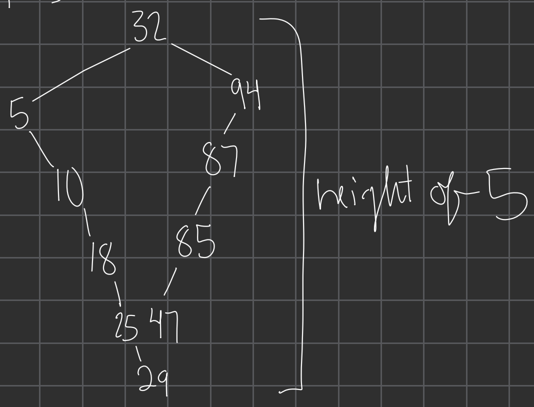
a. [7.9, 0.5, 1.0, 6.5, 8.2, 7.0, 6.6, 9.9, 1.2, 2.4, 5.6, 3.6]



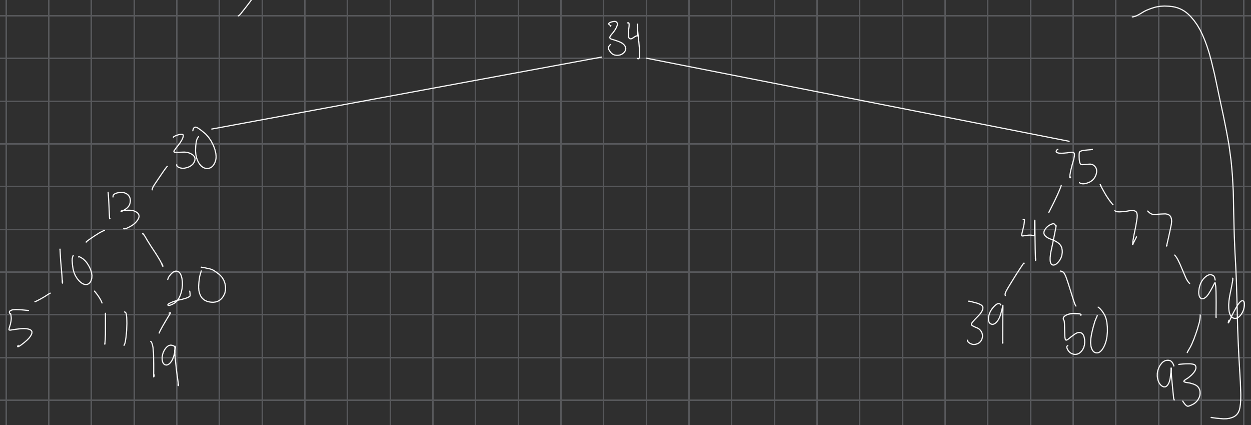
b. ['Petit Four', 'Cupcake', 'Donut', 'Eclair, Froyo', 'Gingerbread', 'Honeycomb']



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]



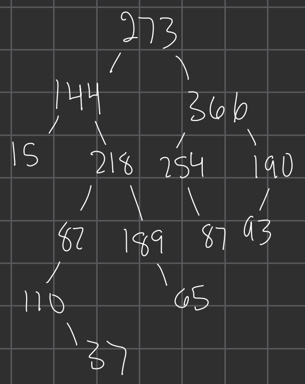
height of 4

2. (text) BST Traversal [10 points]

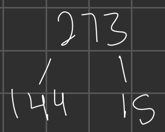
a.

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

(68+21+(92×2))=273
21+15+(54×2)=144
15+0+0 = 15
54+46+(59×2)=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



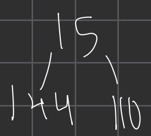
PreOrder: [273, 144, 15, 218, 82, 110, 37, 189, 65, 366, 254, 87, 190, 93]



X not a BST, since not BST cant be AVL

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

a



X Same

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

c. no, not bst so cant be AVL bc AVL is BST + balancing property.

5. (text) Algorithm Analysis [5 points]

⟵ time

⟵ space

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 it loops through the provided list of candidates once, adding each one to a HashMap with an initial vote count of zero. Each insertion into the map is a constant-time operation, but the loop makes the total complexity linear in the number of candidates.

castVote

The castVote method has a time complexity of O(1) and a space complexity of O(1). This method simply updates the vote count for a specific candidate in the HashMap. Since accessing and updating values in a HashMap are constant-time operations, it performs efficiently regardless of the number of candidates or votes.

castRandomVote

The castRandomVote method has an average-case time complexity of O(1) and a space complexity of O(n). The method randomly selects a candidate to vote for by converting the key set of the HashMap to a list and selecting a random index. Although the selection and vote casting are constant-time operations, creating the list of candidates introduces a linear-time and linear-space operation relative to the number of candidates.

rigElection

The rigElection method has a time complexity of O(n) and a space complexity of O(1). It assigns a majority of votes to the specified candidate and resets all others to zero initially. Then, it redistributes any remaining votes by giving one vote to other candidates until all votes are used. Both of these steps require iterating over all candidates, but no additional data structures are created beyond the existing map, keeping the space usage constant.

getTopKCandidates

The getTopKCandidates method has a worst-case time complexity of O(n log n) and a space complexity of O(n). It adds all candidates and their vote counts into a max-heap (priority queue) to sort them by the number of votes. Building the heap takes linear time, and retrieving the top k elements involves k extractions from the heap, each of which takes O(log n) time. The method also requires temporary storage for the heap, proportional to the number of candidates.

auditElection

The auditElection method has a time complexity of O(n log n) and a space complexity of O(n). It creates a list from the entries of the candidate map and sorts it in descending order based on vote counts. Sorting the list takes O(n log n) time, and storing the list of entries uses space proportional to the number of candidates.