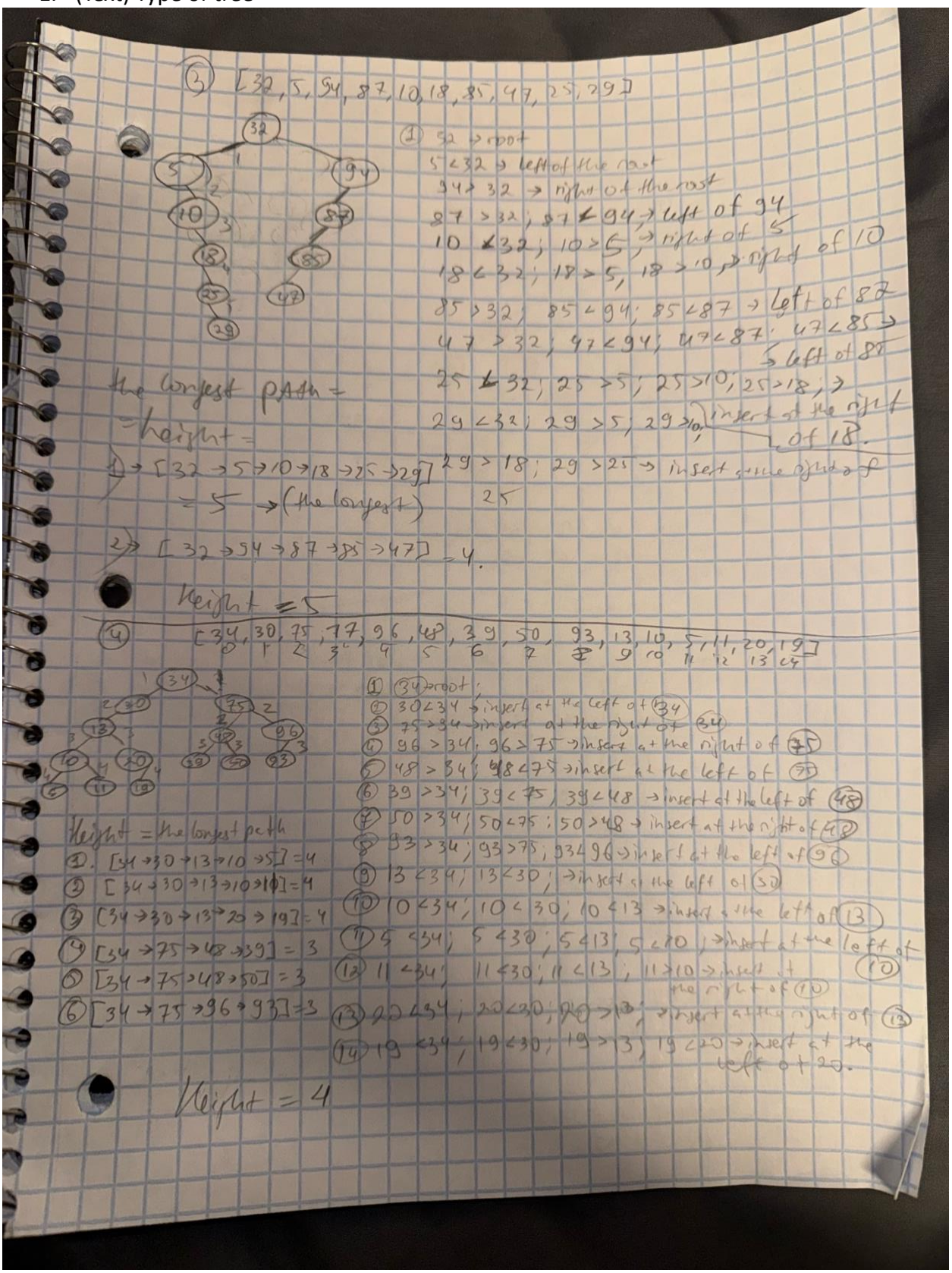
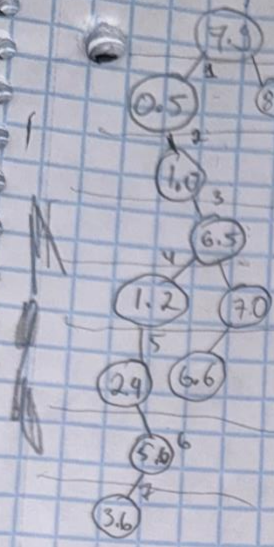
1. (Text) Type of tree



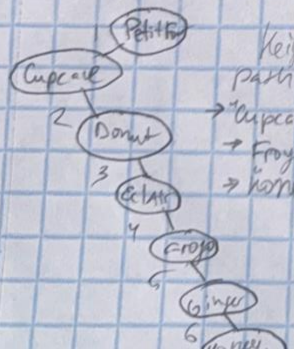
1) [7.9, 0.5, 1.0, 6.5, 8.2, 7.0, 6.6, 9.9, 1.2, 2.4, 5.6, 3.6]



1) 7.9 - root
2) 0.5 < root → left
3) 1.0 < root 1.0 > 0.5 → right of (0.5)
4) 6.5 < 7.9, 6.5 > 0.5, 6.5 > 1.0 → right of (1.0)
5) 8.2 > root → right of the root
6) 7.0 < root; 7.0 > 1.0; 7.0 > 6.5 → right of (6.5)
7) 6.6 < root; 6.6 > 0.5; 6.6 > 1.0; 6.6 > 6.5; 6.6 < 7.0 → left of (7.0)
8) 9.9 > root; 9.9 > 8.2; → right of (8.2)
9) 1.2 < root; 1.2 > 0.5; 1.2 > 0.5; 1.2 < 6.5 → insert at the left of (6.5)
10) 2.4 < root; 2.4 > 0.5; 2.4 > 1.0; 2.4 < 6.5, 2.4 > 1.2 → insert at the right of (1.2)
11) 5.6 < root; 5.6 > 0.5, 5.6 > 1.0, 5.6 < 6.5, 5.6 > 1.2, 5.6 > 2.4 → insert at the right of 2.4
12) 3.6 < root, 3.6 > 0.5; 3.6 > 1.0, 3.6 < 6.5, 3.6 > 1.2, 3.6 > 2.4, 3.6 < 5.6 → insert at the left of (5.6)

The longest path → height = [7.9 → 0.5 → 1.0 → 6.5 → 1.2 → 2.4 → 5.6 → 3.6] = 7,
Height = 7
height = [7.9 → 8.2 → 9.9] = 2 (not the longest)

2) Lexicographically: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
["Petit Four", "Cupcake", "Donut", "Eclair", "Froyo", "Gingerbread", "Honeycomb"]

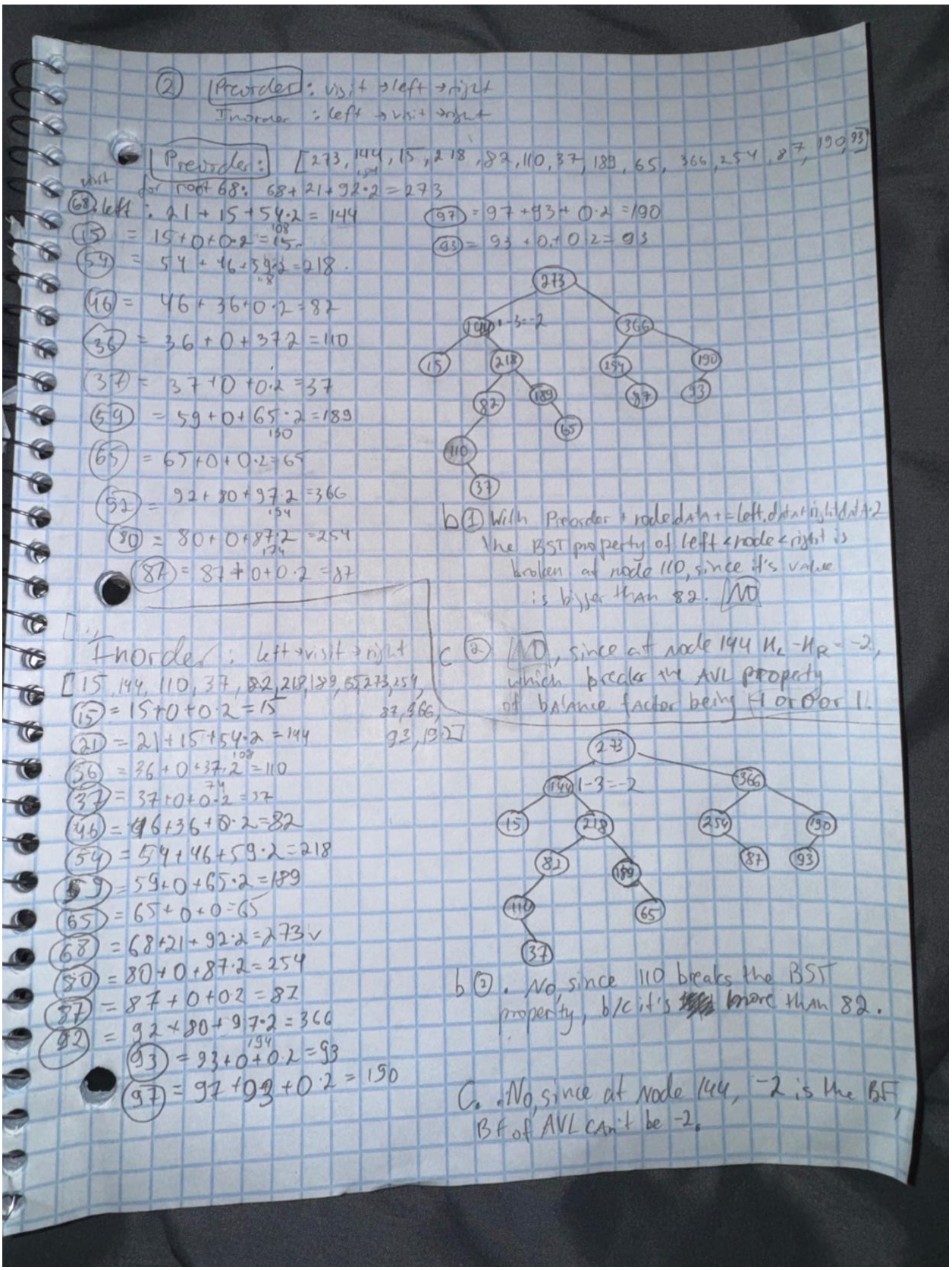


Height = the longest path = ["Petit Four" → "Cupcake" → "Donut" → "Eclair" → "Froyo" → "Ginger" → "Honeycomb"] = 6.

1. "Petit Four" - root
2. "Cupcake" < "Petit four" → left of the root
3. "Donut" < "Petit four", "Donut" > "Cupcake" → right of "Cupcake"
4. "Eclair" < "Petit Four", "Eclair" > "Cupcake", "Eclair" > "Donut" → right of the "Donut"
5. "Froyo" < "Petit Four", "Froyo" > "Cupcake", "Froyo" > "Donut", "Froyo" > "Eclair" → insert at the right of "Eclair"
6. "Gingerbread" < "Petit Four", "Gingerbread" > "Cupcake", "Gingerbread" > "Donut", "Gingerbread" > "Eclair", "Gingerbread" > "Froyo" → insert at the right of "Froyo"
7. "Honeycomb" < "Petit Four", "Honeycomb" > "Cupcake", "Honeycomb" > "Donut", "Honeycomb" > "Eclair", "Honeycomb" > "Froyo", "Honeycomb" > "Ginger" → insert at the right of "Ginger"

Height = 6

2. (text) BST Traversal



5. Time complexity analysis for each method:
Time and Space Complexity Analysis (Worst Case)
m = Number of candidates
p = Total number of electorate votes

Election Class:

1.initializeCandidates (LinkedList<String> candidates)
Time: 0(m)
Initializes a map with m candidates (loops through all candidates once).
Space: 0(m)
Stores m candidates in the votes map and candidates list.

2. castVote(String candidate)
Time: O(1) (average case) / O(m) (worst case for hash collisions)
Checks and updates the candidate's vote count in the hash map.
Space: O(1)

3. castRandomVote ()
Time: O(1)
Randomly selects a candidate and casts a vote.
Space: O(1)

4.rigElection (String candidate)
Time: O(m)
Iterates over all m candidates twice:
Calculates the total votes of other candidates.
Redistributes votes to rig the election.
Space: O(1)

5.getTopKCandidates (int k)
Time: O(m + k log m)
Heap construction: O(m) (adding m candidates to a max-heap).
Extracting top k: O(k log m) (each extraction from the heap takes 0(10g m)).
Space: O(m) (stores all m candidates in the heap).

6.auditElection()
Time: O(m log m)
Sorts all m candidates by votes (descending order).
Space: O(m) (stores the sorted list of m candidates).

7.setElectorateVotes (int p)
Time: O(1)
Space: O(1)

ElectionSystem Class (Main Method)

1.Shuffling Candidates
Time: 0(m)
Randomizes the order of m candidates using Collections. shuffle.
Space: 0(m) (stores the shuffled list).

2.Casting Votes
Time: 0(p)
Casts p votes (each castRandomVote is 0(1)).
Space: 0(1)

3.Rigging the Election
Time: 0(m)
Same as rigElection in the Election class.
Space: 0(1)

4.Top-K Queries & Audit
Time: 0(m log m)
Dominated by sorting in auditElection.
Space: 0(m) (stores sorted results).

Overall Time : O(m + p + m log m)
Overall Space : O(m)