Types of Trees (1)

1.2

0.5 5.6

1.0 2.4 7.9

6.5 8.2

6.6 9.9

7.0

If it’s not clear, 5.6 has 2.4 and 7.9 as children.

Petit Flour

Cupcake

Donut

Éclair

Froyo

Gingerbread

Cupcake is the left child of Petit Flour, and all other nodes are right children of the previous node.

32

5 85

18 47 94

10 25 87

29

This text is just to mark the distinction between C and D.

34

13 75

10 20 48 93

5 11 19 30 39 50 77 96

BST Traversal (2)

The trees below are meant to be a 1-1 copy of the tree in the assignment graph, with the specific values replaces with the modified ones.

Preorder:

138

63 220

15 130 123 190

82 92 87 93

54 65

37

Postorder:

302

79 310

15 86 123 190

100 59 87 93

54 65

37

b) neither of these are BSTs. The ordering property is broken on both trees with the right child of 220/310, which is the root’s right child, as 190 is less than both of these numbers, but is the right child regardless.

c) One of the prereqs for being an AVL is that the tree must be a BST, and neither is a BST, so no.

Algo Analysis (5):

initializeCandidates: Time and space complexity are both O(n), with n being the size of the input string. My foreach loop goes through every string in the input, and within that loop, a node is created storing the data of each string, hence O(n) space.

castVote: Time complexity is O(n), with n being the size of the existing heap of candidates. This is due to a for loop which runs (at worst) across every element in the heap, as well as the PriorityQueue’s toArray method that I called. I’m not sure of the specific time complexity of this, but at worst, it would be O(n). My space complexity is also O(n), because I create a new array equal in size to the input heap.

randomVote: Both complexities for this are the same as castVote. This is because castVote is called at the end of randomVote, and nothing added in randomVote increases the time or space complexity.

rigElection: rigElection is, for the most part, the same code as castVote, with modifications to the interior of the for loop. These modifications don’t affect the time or space complexity, however, so the time and space complexity for this is the same as for castVote, for the same reasons.

getTopKCandidates: Both time and space complexity for this is O(k), with k being the integer provided to the method. The while and for loop both run k times, leading to O(k) time complexity, and a String array is created at the beginning of the method, with size k, leading to O(k) space complexity.

auditElection: Time and space complexity are both O(n), with n being the size of the input heap. The for loop runs once for each element in the array, leading to O(n) time complexity, and an array of nodes is created at the beginning of the method that is equal to the size of the input heap, hence O(n) space complexity.

main: This method is in ElectionSystem. The most intensive method call in both time and space complexity is, say, castVote (a few are tied), so we’ll go with that and say time and space complexity are the same as castVote, for the same reasons.

compareTo: This method is in my Node class. Space complexity is zero, as no variables are declared or initialized. Time complexity is constant, as Integer.compare() has a constant time complexity.