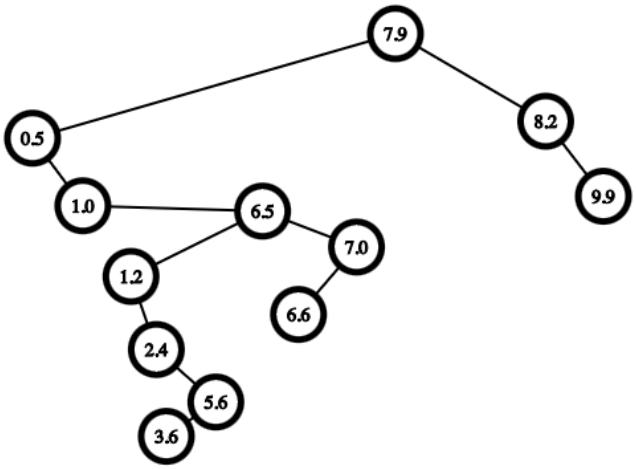
GitHub Repo: https://github.com/Connor-Cash532/Assignment-6

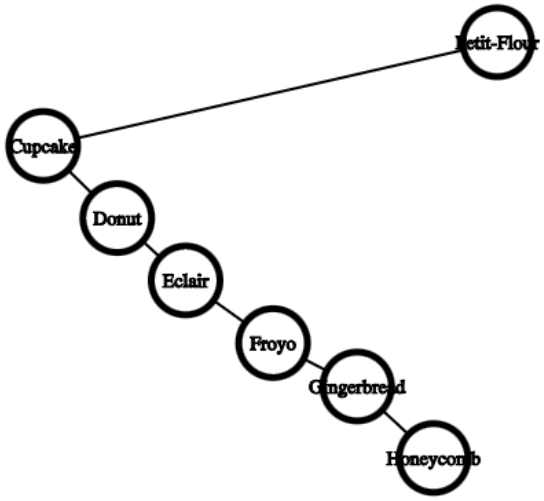
1.

BST of [7.9, 0.5, 1.0, 6.5, 8.2, 7.0, 6.6, 9.9, 1.2, 2.4, 5.6, 3.6]



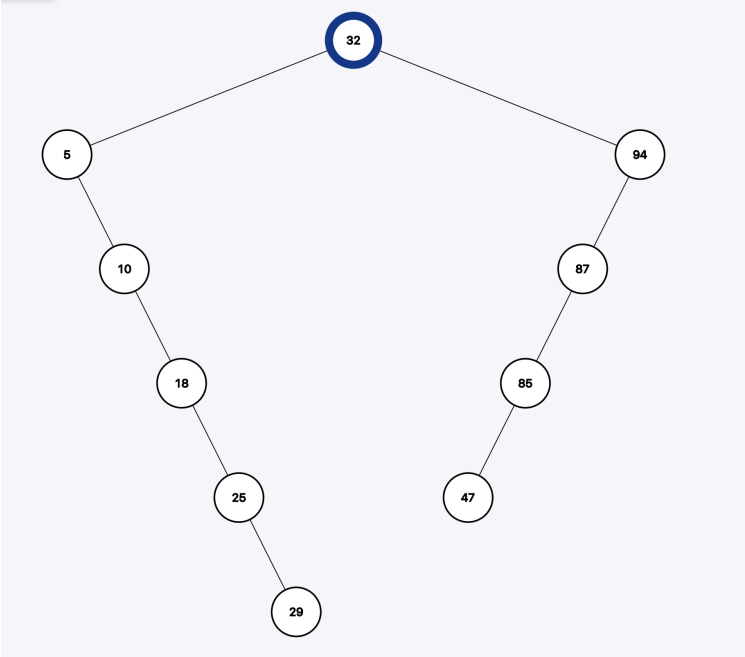
Height = 7

BST of [“Petit Four”,“Cupcake”, “Donut”, “Eclair”, “Froyo”, “Gingerbread”, “Honeycomb”]



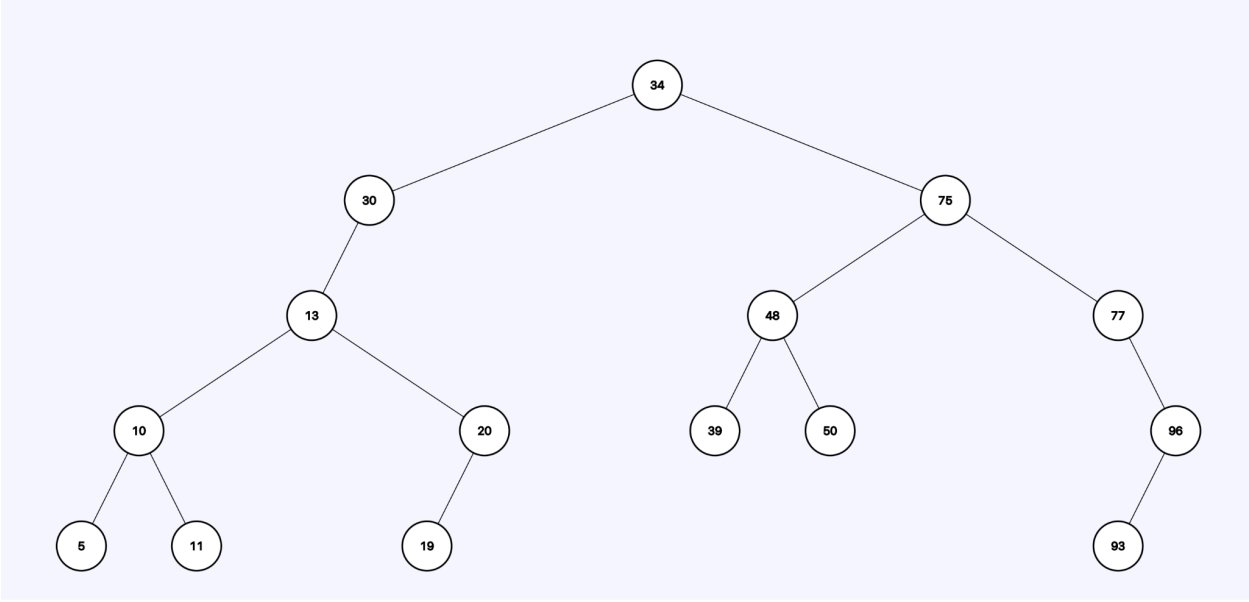
Height = 6

BST of [32,5,94,87,10,18,85,47,25,29]



Height = 5

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

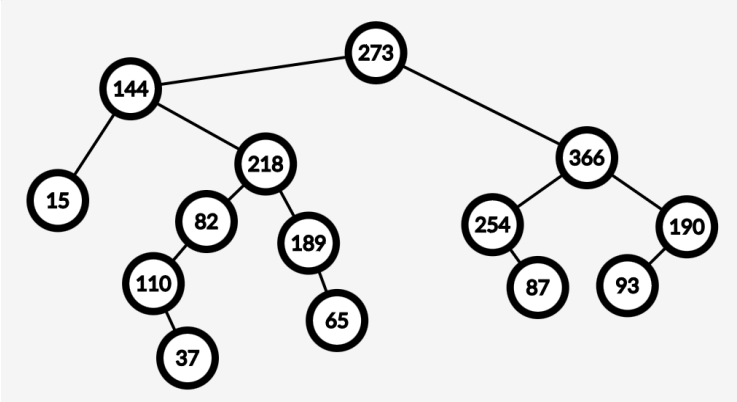


Height = 4

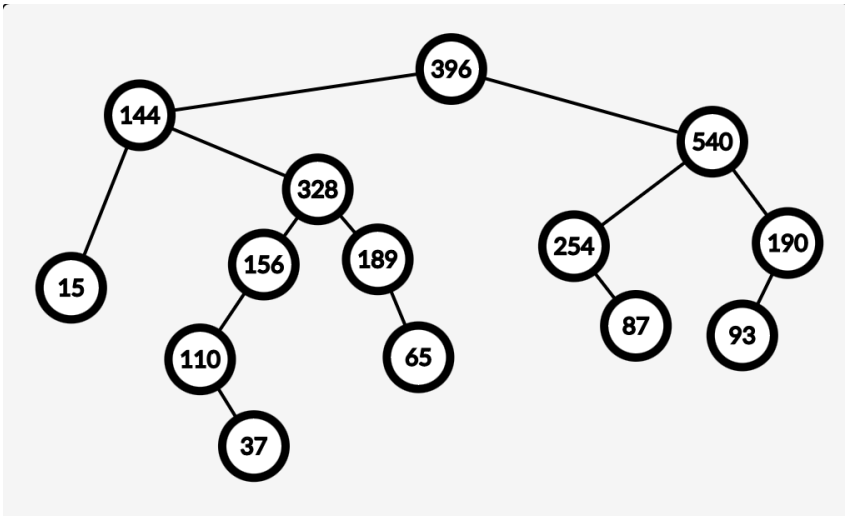
2.

a.

Traversal:
Visit 68
273
Visit 21
144
Visit 15
15
Visit 54
218
Visit 46
82
Visit 36
110
Visit 37
37
Visit 59
189
Visit 65
65
Visit 92
366
Visit 80
254
Visit 87
87
Visit 97
190
Visit 93
93
Resulting Subtree:



Inorder
Visit 15
15
Visit 21
144
Visit 36
110
Visit 37
37
Visit 46
156
Visit 54
328
Visit 59
189
Visit 65
65
Visit 68
396
Visit 80
254
Visit 87
87
Visit 92
540
Visit 93
93
Visit 97
190
Resulting Subtree:



B. Both resulting trees are not BSTs and have multiple nodes breaking the BST property. For example in the preorder tree, 87 < 254 but 87 is a right child of 254. In the inorder tree 65 < 189 but 65 is a right child of 189.

C. No, both resulting trees are not AVLs because an AVL tree is balanced, and the original tree, preorder tree, and inorder tree are all not balanced. No rotations are made when creating the resulting trees so the resulting trees can not become balanced. Multiple nodes break the balance property. For example, nodes 21, 144, and 144 in the trees all have a BF of -3 where BF is the balance factor defined by left subtree height minus right subtree height where no subtree is equal to -1 and height increments by from -1.

5.
Time Complexity:
Let n=# of candidates in the election, p=number of electoral votes, and m=(p/2)+1.
I am going to explain the time complexities of each method starting with the methods in the Election class.
initiliazeCandidates() method has time complexity O(n) since it contains a loop that runs n times. The castVote() method has calls to remove and add methods of a priority queue, the get and replace methods of a HashMap, and a call to the updateVoteTotals2() method. Priority queue add method has time complexity O(log(q)) where q is the size of the priority queue. Since the priority queue has n elements each representing each candidate's votes the add method has time complexity O(logn). The priority queue remove method has time complexity O(n) since the underlying heap structure for the priority queue does not provide a way to search for an element in less than linear time, so in the worst case all elements in the priority queue would have to be looked at to find the element to be removed. Both HashMap get and replace methods are O(1) since the key is used to find the value and from then can be either accessed or changed. The updateVoteTotals2() method has time complexity O(n^2) because the hashmap clear method is called and a loop runs n times which contains a call to get method of a linked list which in the worst case will have to iterate over the entire linked list to find the element being retrieved. Since the linked list is of the candidates it is size n which leads to the time complexity for the

loop being O(n^2). The other method calls in the method use HashMap constant retrieval with a key. Ultimately this makes the time complexity of the castVote() method n^2. The castRandomVote() method makes method calls to nextInt() of a random object, get() of a linked list, castVote() method, and updateVoteTotals2(). The time complexity of nextInt() is constant and the time complexity of get() is n since the linked list has size n. Then as discussed earlier castVote() and updateVoteTotals2() both have time complexities O(n^2). The rigElection() method makes calls to priority queue remove and add, HashMap replace and get, and has a loop. As discussed earlier the priority queue remove method has time complexity O(n) and add has time complexity O(logn) and HashMap get and replace methods have time complexity O(1). The outer loop and the while rearranges the votes to give the rigged winner the majority, so in the worst case the rigged winner would have 0 votes so m total votes would have to be rearranged. Inside the while loop a call is made to the removeVotes() method and linked list get method. The removeVotes() method has time complexity O(n^2) for the same reason cast votes does since the only difference between the two methods is changing the assignment value made in the replace method call. Then the inner body of the loops would have time complexity n^2 + n, so the total time complexity of rigElection() would be O(m*n^2). The getTopKCandidates() method has calls to updateVoteTotals2(), peek method for priority queues, and a loop. As discussed earlier updateVoteTotals2() has time complexity O(n^2). The peek method has time complexity O(1) since it just gets the top element and does not change the heap. The loop runs at most k times where k is the parameter to the function. Inside the loop calls are made to the add and get method of linked lists. The add method has time complexity O(1) since it simply updates the pointer but as discussed earlier the get method has worst case time complexity of the size of the list. In regards to the method, this means the worst case would be when all candidates have the same number of votes and the list being searched is size n. Thus the call to the get method would be O(n), so the total time complexity would be O(n^2 + k*n)=O(n^2). The final method of the election class is auditElection() which calls the getTopKCandidates() method and has a loop which runs n times. The auditElection() uses the getTopKCandidates() but k is equal to n since we want all candidates, so the linked list has size n also. Thus, the total time complexity of auditElection() is O(2n^2)=O(n^2).

Time complexity of ElectionSystem class

The initRandomElection() method contains a loop which runs count times and a call to the get method of a linked list, but the count variable is a random integer with bounds 0-250, since names.length=250. The linked list will never be bigger than 250 and the loop will never run more than 250 times, so the total time complexity of the method is O(250*250(1))=O(1). The executeElection() method makes calls to initRandomElection(), initializeCandidates(), castRandomVote(), auditElection(), and getTopKCandidates(). The sum of all of these methods time complexity in big-o notation is 1+n^2+n^2+n^2+n^2+n^2=4n^2=O(n^2), so the total time complexity of the method is O(n^2)

Time complexity of Main

The main class initializes an ElectionSystem object and contains a loop which runs 10 times and executeElection() each time. The executeElection() method has time complexity O(n^2), so the total time complexity would be O(10*n^2)=O(n^2).

Space complexity:
Throughout all classes the data that affects space complexity is the voteTotals and voteTotals2 hashmaps, the candidates linked list, maxHeap priority queue, and names array. Note that other temporary linked lists are included in methods but each linked list will never be more than size n, so the total space complexity will still be n. Additionally, the linked list values of voteTotals2 will always add up to n since each lists represent each candidate number of votes, All data structures besides the names array will have at most n items in them since they all hold data about each candidate. The names array's space complexity is O(1) since it never grows with the input and is always size 250. Overall, this makes the space complexity O(n) since all data contains at most n items and no variables are initialized in nested loops.