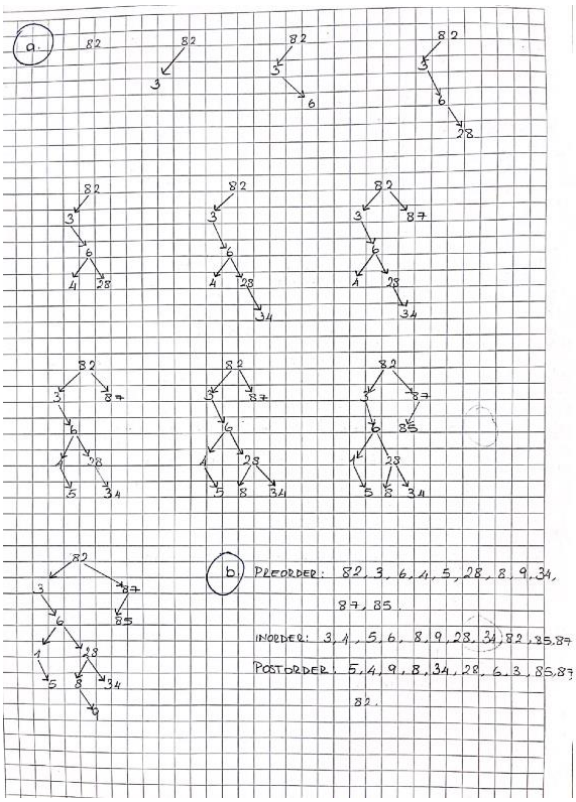
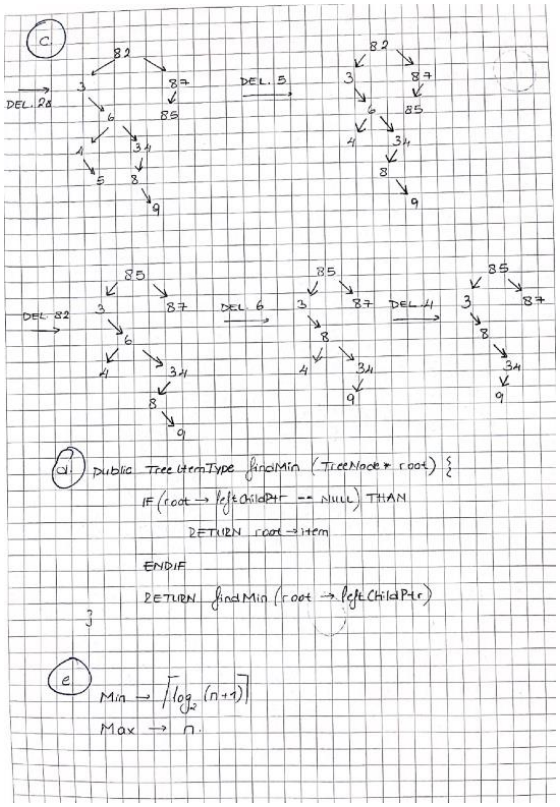
```
/**
* Title: Binary Search Trees
* Author: Yasemin Akın
* ID: 22101782
* Section: 001
* Homework: 2
* Description: Answers to questions 1 and 3
*/
```

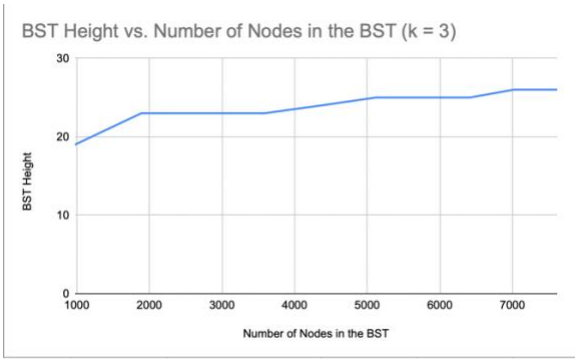
Question 1

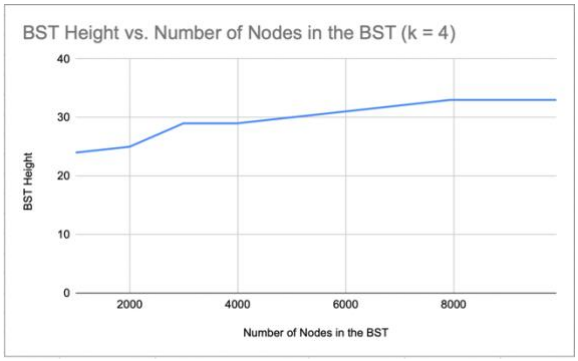




Question 3

1.



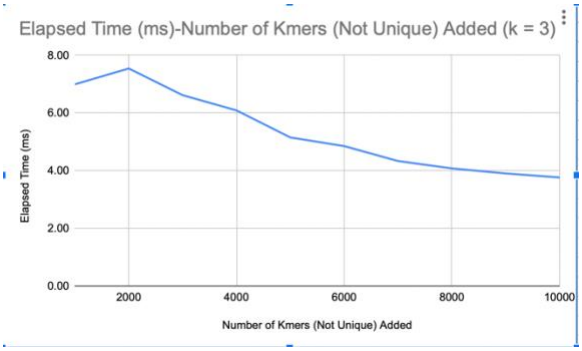


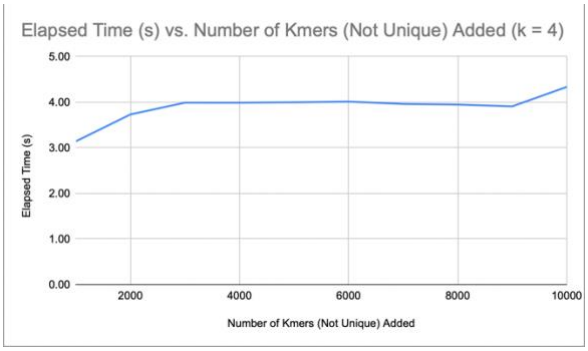
These graphs appear to depict a function that initially rises quite steeply before the rate of increase slows, suggesting a logarithmic pattern rather than a linear one. This is characterized by the curve flattening as the number of nodes increases, which is typical for a logarithmic function.

This behavior is expected for a non-self-balancing binary search tree (BST) height with randomly generated string inputs of a fixed length (k). The logarithmic growth is due to the binary nature of the tree: ideally, each level has twice as many nodes as the previous level, so the height of the tree increases by one for a doubling of the number of nodes. However, the randomness in input strings helps maintain an average-case scenario where the tree remains relatively balanced, avoiding the worst-case linear increase in height.

In summary, the graphs represent a typical scenario for inserting random strings into a BST, where the average height of the tree grows logarithmically with the number of nodes, indicating an efficient average-case insertion time.

2.





Inserting sorted kmers into a binary search tree (BST) would lead to a degenerate tree structure with operations such as insertion, search, and deletion all having a time complexity of (O(n)) instead of the average-case (O(logn)). This is because the BST would become a linked list, with each kmer being a child of the previous one, leading to a tree height equal to the number of nodes. Such linear complexity represents a significant decrease in efficiency, particularly for large datasets, compared to the logarithmic complexity achieved with random insertion in a balanced BST.