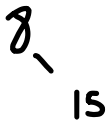
Alara Zeybek
22102544
Sec-003

CS-202 HW2 REPORT

Question 1-a) (5 points) Insert 8, 15, 10, 4, 9, 6, 16, 5, 7, 14 into an empty binary search tree in the given order. Show the resulting BST after every insertion.

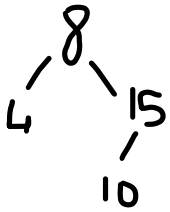
inserting 8:

inserting 15:

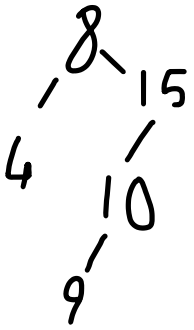


inserting 10:

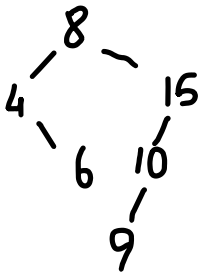
inserting 4:



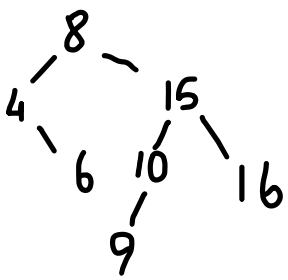
inserting 9:



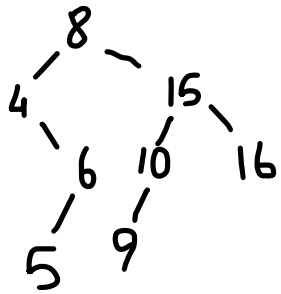
inserting 6:



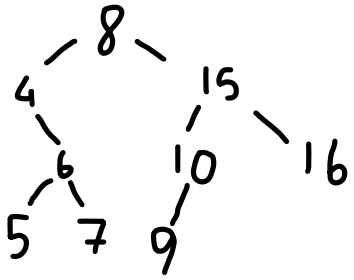
inserting 16:



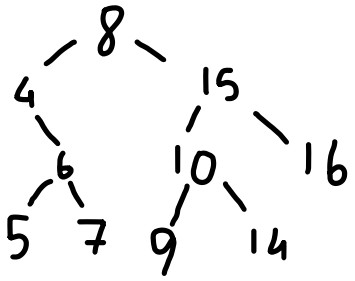
inserting 5:



inserting 7:



inserting 14:



Alara Zeybek
22102544
Sec-003

Question 1-b) (5 points) What are the preorder, inorder, and postorder traversals of the BST you have after (a)?

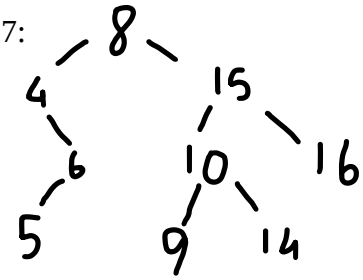
preorder: 8, 4, 6, 5, 7, 15, 10, 9, 14, 16

inorder: 4, 5, 6, 7, 8, 9, 10, 14, 15, 16

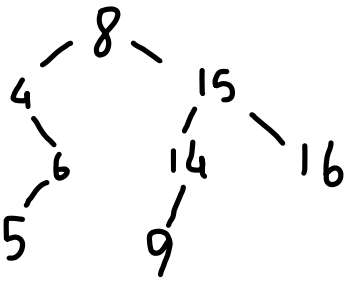
postorder: 5, 7, 6, 4, 9, 14, 10, 16, 15, 8

Question 1-c) (5 points) Delete 7, 10, 8, 9, 5 from the BST you have after (a) in the given order. Show the resulting BST after every deletion.

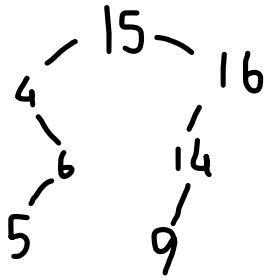
Deleting 7:



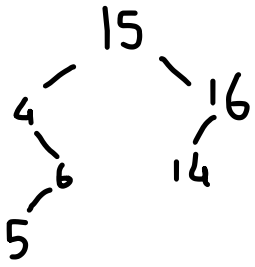
Deleting 10:



Deleting 8:

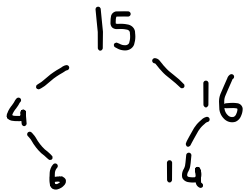


Deleting 9:



Alara Zeybek
22102544
Sec-003

Deleting 5:



Question 1-d) (5 points) Write a recursive pseudocode implementation for finding the minimum element in a binary search tree.

```
int findingTheMin(int* tempPtr){
if(tempPtr is null{
        return INT_MIN;
}
else if(tempPtr->left is null){
        return tempPtr->item;
}
findingTheMin (tempPtr->left);
}
```

Question 1-e) (5 points) What is the maximum and minimum height of a binary search tree that contains n items ?

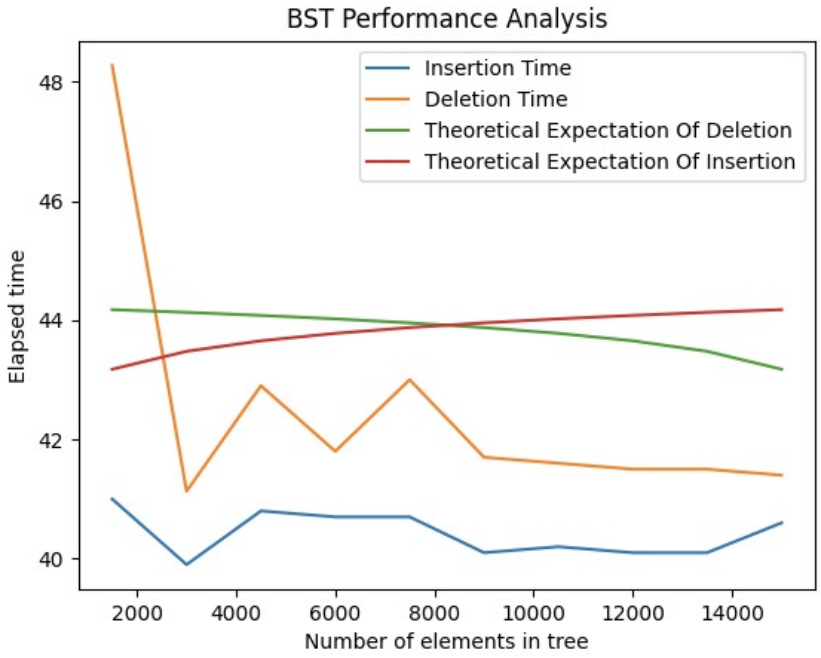
Max height = n

Min height = log(2 to n)

Question 3 Results & Analysis:

Array Size	Insertion	Delete after shuffle
1500	41.478ms	48.346ms
3000	40.090ms	41.148ms
4500	40.752ms	43.913ms
6000	41.459ms	42.650ms
7500	40.185ms	42.374ms
9000	40.962ms	42.067ms
10500	40.779ms	41.725ms
12000	39.610ms	41.347ms
13500	41.485ms	40.916ms
15000	41.377ms	40.367ms

Alara Zeybek
22102544
Sec-003



In this part of the homework, I am asked for comparing the time complexity of two algorithm considering a binary tree: insertNode and shuffle-deleteNode.

Theoretically, the best case of insertNode() algorithm is when the binary tree is empty because in that case, we only dealing with one pointer(root pointer) and **best case takes O(1) time**. In average case of this algorithm, we make recursive calls, in order to find the inserted item's location. Thus, each time we limited our search area by going right or left. That means dividing the task by two. Since each recursive call we divide the task by two, **the average time complexity for insertion in a binary tree is O(log(n)).** The worst case of insertion in a binary tree is when every node in the tree has only one child. This case is similar to the link list; thus, to go to the end of the tree, essentially, we have to go through every node in the tree. So, **the worst complexity of insertion is O(n).**

Theoretically, shuffle takes O(n) time for all cases (best,average,worst) and I am doing it only at the beginning with 15000 array size. Then, I am calling deleteNode() algorithm for small pieces of the array with 15000 elements.

Theoretically, the best case of the deleteNode() algorithm is when the want-to-delete item is in the first place because in that case, we only dealing with one pointer(root pointer) and **best case takes O(1) time**. In average case of this algorithm, we make recursive calls, in order to find the want-to-delete item's location. Thus, each time we limited our search area by going right or left. That means dividing the task by two. Since each recursive call we divide the task by two, **the average time complexity of deletion in a binary tree is O(log(n)).** The worst case of insertion in a binary tree is when every node in the tree has only one child and the item looking is the last element in the tree (at the end of the tree). Hence, having only one child case is similar to the link

list; in order to go to the end of the tree, essentially, we have to go through every node in the tree. So, **the worst complexity of deletion is O(n).**

In my experiment, the tangents of the algorithms are much more than I expected under theoretical examination. They do not exactly show O(logn) lines such as theoretical lines.

In my case, I wasn't dealing with inserting and deleting an ascending array. I was dealing with a randomly created array. Therefore, the worst case of the algorithms wasn't included. I was examining the average case of the algorithms. As, the size increases the expectation was that the insertion time would increase. Moreover, because the unit is milliseconds, there seems to be slight deviations in the time. Furthermore, in deletion the array size shrinks every run. The expectation in this case was that the time would decrease every time. The result was again parallel with the expectation but because the intervals are small it seems that the results go against the expectation.

Finally, I learned that Binary Search Tree insertion and deletion works fairly well in best and average cases. If the working database does not include consistently and linearly increasing or decreasing cases Binary Search Tree could be used. Linearly increasing or decreasing cases would cause linear results because of the time complexity O(n).