CS202: Fundamentals of Computer Science II

HOMEWORK - 2

Binary Search Trees

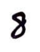
Zeynep Begüm Kara - 22003880

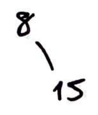
CS202 - 01

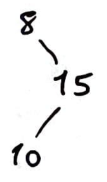
March 20, 2023

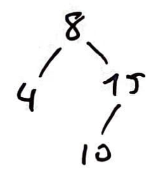
Question 1

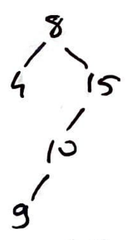
a)

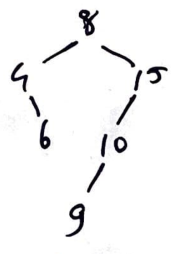

insert 8

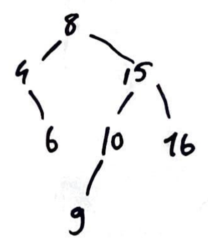

insert 15

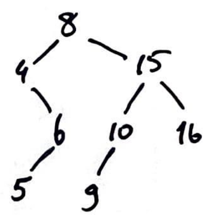

insert 10

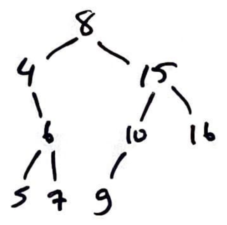

insert 4

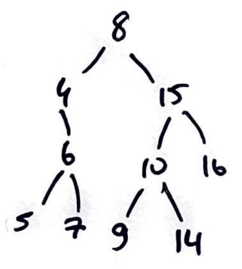

insert 9


insert 6


insert 16


insert 5


insert 7

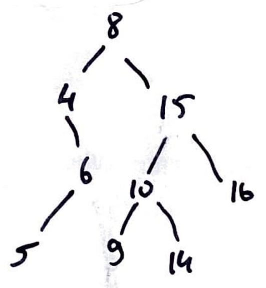

insert 14

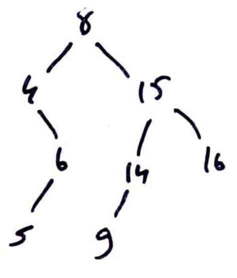
b) Preorder : 8 4 6 5 7 15 10 9 14 16

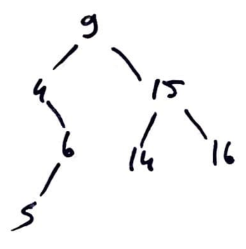
Inorder : 4 5 6 7 8 9 10 14 15 16

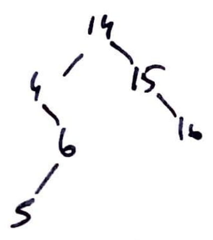
Postorder : 5 7 6 4 9 14 10 16 15 8

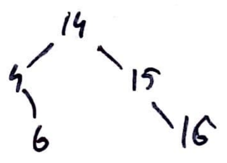
c)


delete 7


delete 10


delete 8

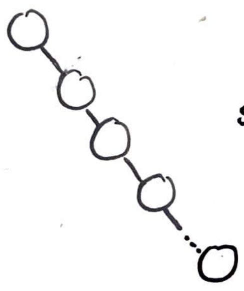

delete 9


delete 5

d)
```
function minOfBST (root)
    if root.leftChild == null
        return root.value
    else
        return minOfBST (root.leftChild)
```

e) <u>Max heigt</u> : n

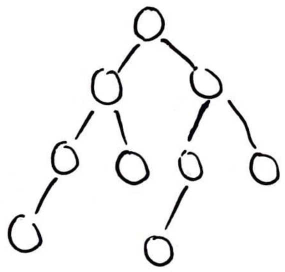
All internal nodes have one child



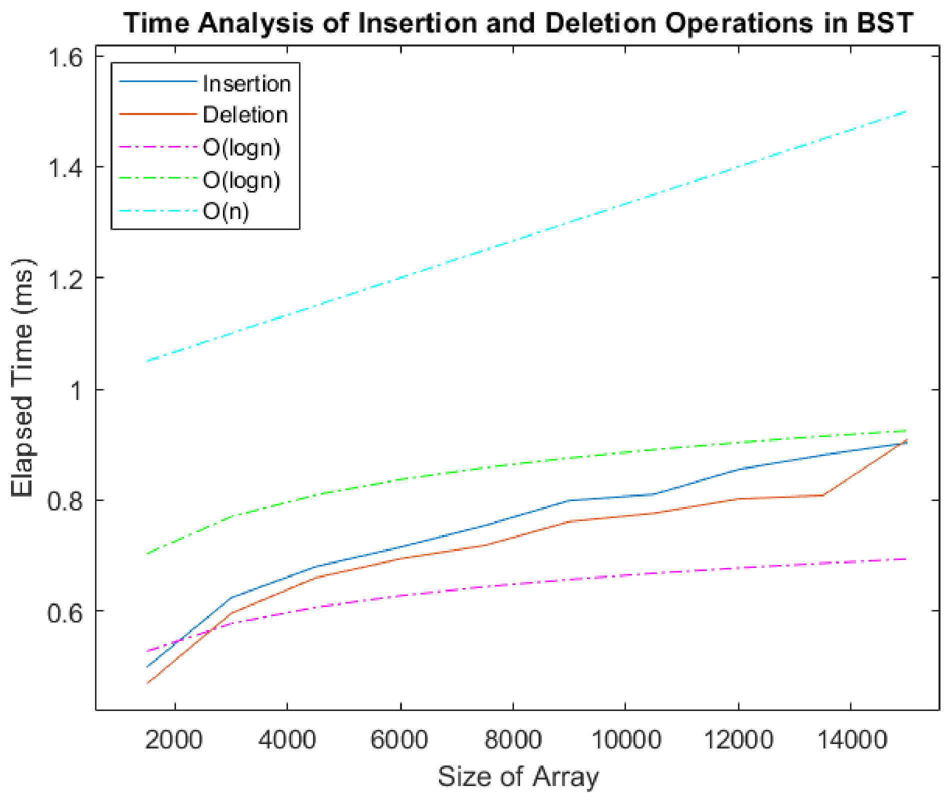
simply a linked list

<u>Min height</u> : $\log_2\lceil n+1\rceil$

baleneed binary trees



Results



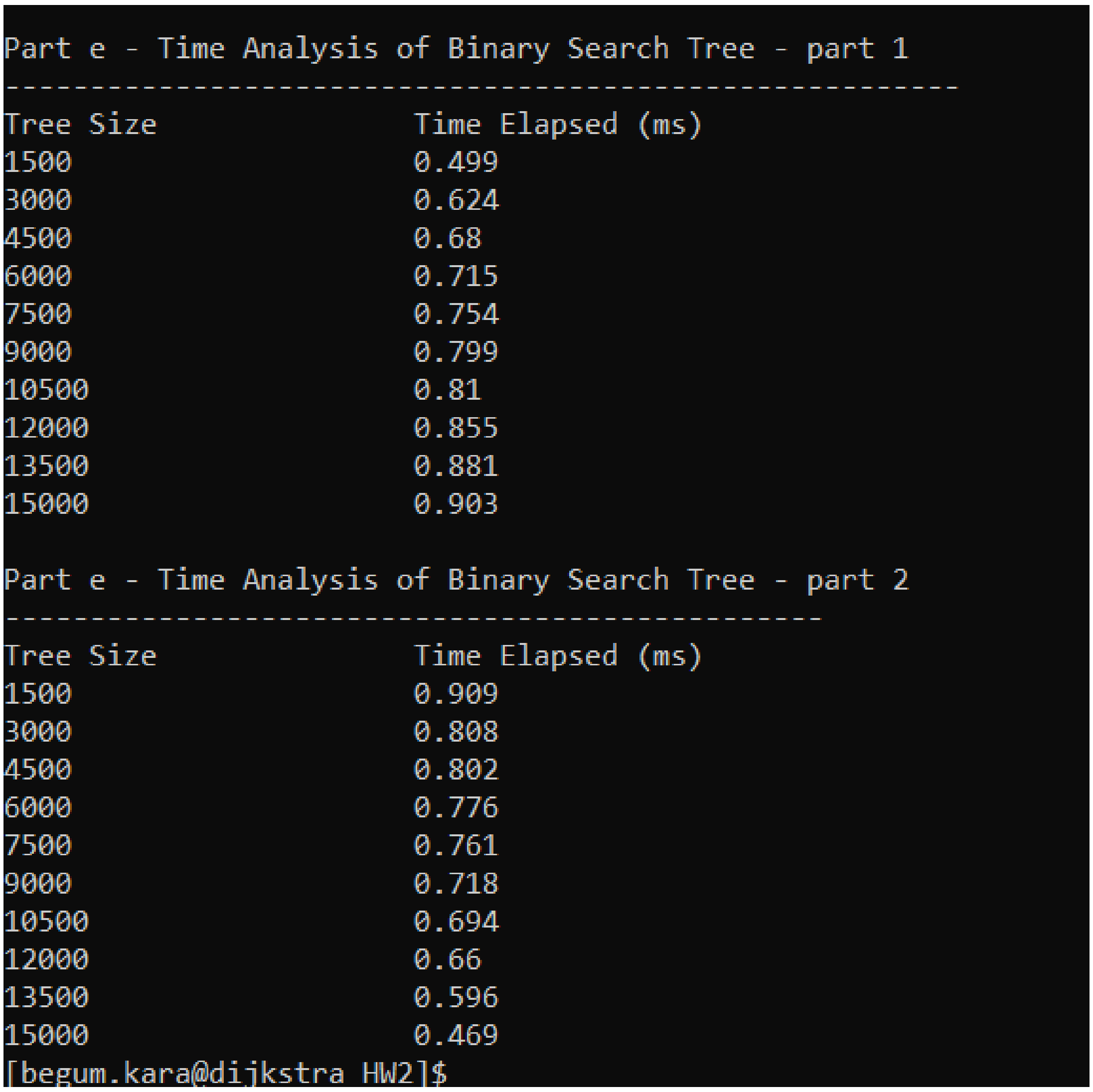
In this study, it is aimed to observe the time complexity of insertion and deletion operations in a binary search tree. This experiment is conducted on a randomly generated sequence of integers meaning that random numbers are inserted into or deleted from the binary search tree.

Recall that in pointer-based implementation, deleting a node whose place is known is simply a pointer manipulation, which is $O(1)$. Also, recall that searching in a binary search tree is proportional to height, which is $O(h)$. Observe that insertion and deletion operations are nothing but a combination of these two operations. Hence, our theoretical expectation in both operations was $O(h)$ where h is the height of the binary search tree. Since BST is constructed from randomly generated integers in the average case the height h would be directly proportional to $logn$. Therefore, in the average case insertion and deletion operations is $O(logn)$. On the other hand, in the worst case the height of the binary search tree is n. This situation occurs when the inserted numbers are sorted either in ascending or descending ways. In such cases, when we are constructing the tree we will always insert the new items to only the left subtree or only the right subtree according to whether we have an increasing or decreasing sequence, which will lead search operation to visit every node in the structure. Thus, in the worst case insertion and deletion operations are $O(n)$. So, it is expected from the time elapsed to be directly proportional to $logn$ in this experiment since our application domain consists of randomly generated numbers. Of course, here we assume -and hope- that our pseudo random algorithms and the function *rand()* are random enough and it is not likely to have sorted number sequences.

It turned out that our experimental results satisfy these theoretical expectations. In the plot above, the elapsed time vs size graph was shown. Continuous lines, blue and red, represent our experimental result (see the end of the document) whereas dashed lines represent a function that has the complexity declared in legend. For our purposes, these are chosen as follows cyan: $c(n) = n/30000 + 1$, green: $g(n) = log_2 n/15$ and magenta: $g(n) = log_2 n/20$. The plot indicates that the results of the experiment were better than the worst case as we expected by definition. In other words, the growth rate of the experimental results is less than a function whose complexity is $O(n)$. Hence, one can claim that since the worst case scenario didn't realize, it is not possible to have fully sorted arrays of numbers, which means our randomly generated input was random enough. This situation leads us to an average case analysis. Observe that the green graph is an upper bound for our experimental result. Also, observe that the magenta graph is lower bound when $n > 4000$. Since experimental results can be bounded by functions whose complexity is $O(logn)$, the experimental results are strictly bounded and it is $\theta(logn)$. Overall, when we have randomly generated numbers, which is an average case, insertion, and deletion operations are $O(logn)$, as we expected.

Notice that both red and blue functions have fluctuations -not a pure logarithmic increase- through n increasing, which contradicts the theoretical results. Such fluctuations can result from partially ordered number sequences in the randomly generated sets due to pseudo-random algorithms. Also, there might be errors in the measurement of elapsed time due to the sensitivity of time measurement, non-stable computer performance, and significant figure errors. Additionally, the fluctuations might be a result of limited size arrays. In other words, if we had examined much bigger-sized arrays, the results would be closer to the theoretical values. Overall, except for the fluctuations in results, the experiment justifies our theoretical expectations.

Experimental results that are plotted:



```
Part e - Time Analysis of Binary Search Tree - part 1
---------------------------------------------------------
Tree Size                Time Elapsed (ms)
1500                     0.499
3000                     0.624
4500                     0.68
6000                     0.715
7500                     0.754
9000                     0.799
10500                    0.81
12000                    0.855
13500                    0.881
15000                    0.903

Part e - Time Analysis of Binary Search Tree - part 2
-------------------------------------------------
Tree Size                Time Elapsed (ms)
1500                     0.909
3000                     0.808
4500                     0.802
6000                     0.776
7500                     0.761
9000                     0.718
10500                    0.694
12000                    0.66
13500                    0.596
15000                    0.469
[begum.kara@dijkstra HW2]$
```