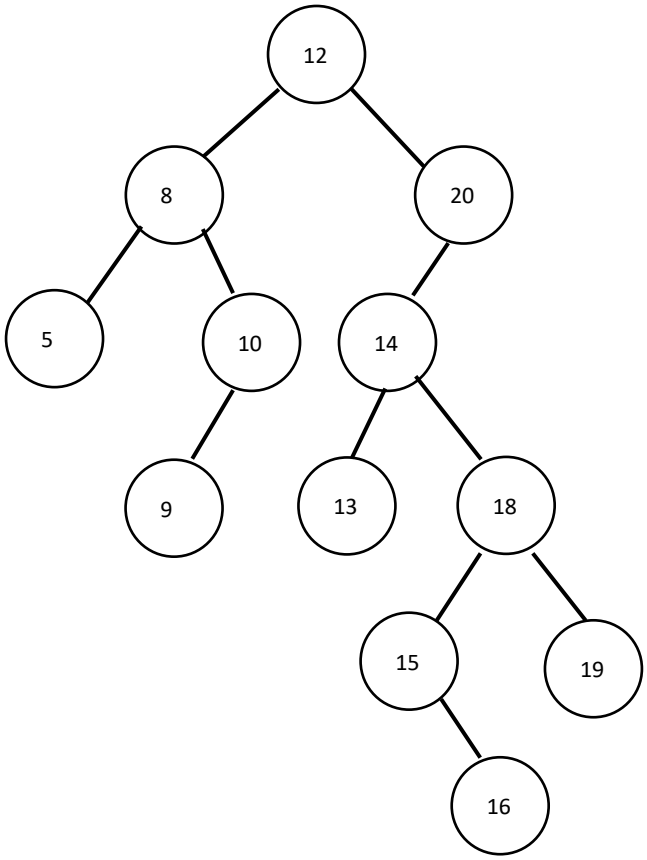
**Title: Binary Search Trees**
**Author : Atakan Akar**
**ID: 22203140**
**Section : 1**
**Homework : 1**
**Description: Report for Questions 1 and 3**

**Question 1)**
**Part A)**

Second sequence can be a true pre-order traversal of a binary search tree. In a binary search tree, the node itself contains the smaller values in its left sub-tree and greater values in its right sub-tree. In a pre-order traversal, the node itself is visited before its left and right child. It can be understood that 20 is the right child of the node with value 12. If first sequence was true than 15 would be the right child of the node with value 14 and 18 will be the right child of the 15. However, after these numbers 13 comes in the sequence which cannot be placed neither of the child nodes of 18 since it is in the right of the node with value 14. Also it cannot be inserted into de left child of the node 14 since it is a pre-order traversal. It cannot be placed into the left of the 15 too since it is also in the right of the 14. So, the first sequence cannot be the right traversal. However second sequence can be a pre-order traversal and can be illustrated as in the lower part.
Corresponding Binary Search Tree:

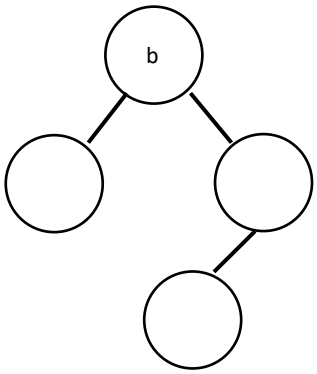


In-Order Traversal:<5,8,9,10,12,13,14,15,16,18,19,20>
Post-Order Traversal:<5,9,10,8,13,16,15,19,18,14,20,12>

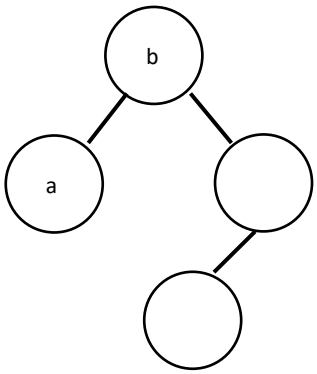
**Part B)**

Explanation:

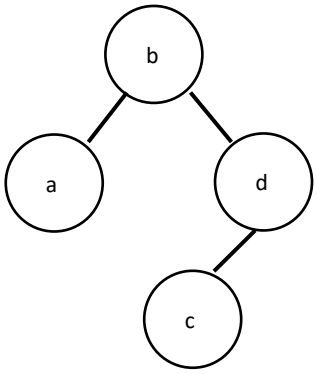
Placing b: We must put number b to the root of the binary search tree due to binary search tree property. In a binary search tree, the node must have smaller numbers than itself in its left sub-tree and greater numbers in its right subtree and in the given scheme there are 2 available nodes in the right of the top node and 1 available node in its left. Number b has 2 numbers greater and 1 number less than itself. Therefore, the root node is the only available choice for the number b. For example, if we put a to the root there won't be enough space in the right sub-tree of the root since the remaining will be greater than a. Also think that we put c or d to the root than there won't we enough space in the in the left sub-tree of the root since a and b are smaller than both numbers and there is only 1 node in the lefts sub-tree. Therefore, b is the only option for the root node.




Placing a: Since a is less than the root node whose value is b, we can only place a to the left-child node of b node. Therefore, there is no other possibility for a. If we wanted to place it to the right of the root than it won't satisfy the binary search tree properties. a is less than b.

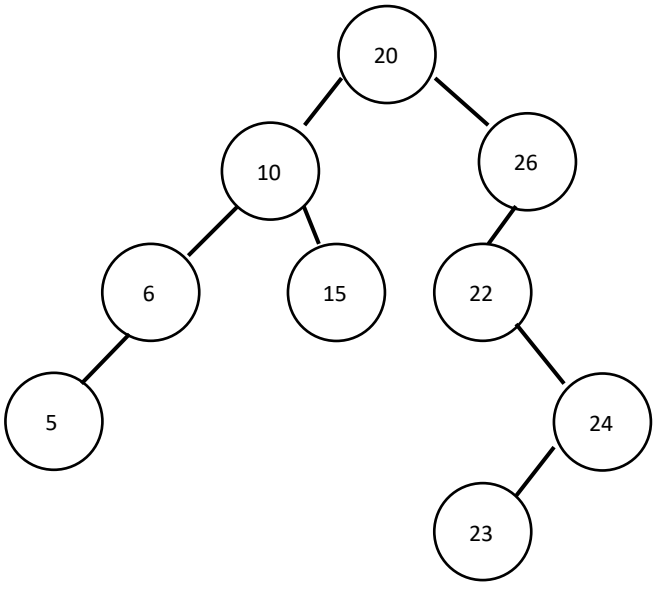


Placing c and d: There are only 2 nodes left which are both in the right subtree of node b. The remaining numbers c and d must be placed in these two nodes and c is smaller than d. In a binary search tree, the node value must be greater than the values in its left sub-tree. Therefore, there is only one possibility the right child of node b must be d and the left child of node d must be number c. If we put c as right child of b than d cannot be placed to the remaining one node because than the tree won't be a binary search tree.
Only Unique Placement:




So we find the unique placement of the values. We should put b first since it must be in the root and c after d since they have only opportunity. However, a can be placed after b in any order since it is only smaller value than b. So there are **3** possibile insertion ways: **<b,a,d,c> ,<b,d,a,c> ,<b,d,c,a>.**

**Part C)**
The post-order traversal is given, and structure of this binary tree is this.



So, the pre-order traversal is this: <20,10,6,5,15,26,22,24,23>
Therefore the 7th key: 22

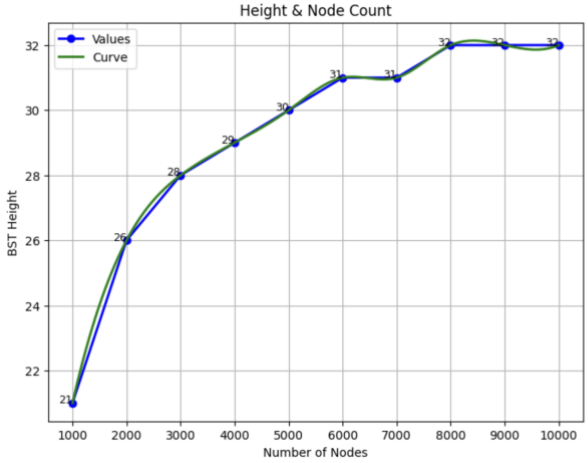
**Part D)**

Explanation: The node with value 43 is inserted first and it will be in the root. Due to the binary search tree features the values greater than 43 will be in the right sub-tree of this value and the smaller ones will be in the left. If two values are compared in insertion one of them must be the ancestor of the other ones. Think the opposite if their lowest common ancestor is different from one of the values then they won't be compared during the insertion process since this means a value between these numbers was inserted before one of these two (4 or 9) numbers and they are in different subtrees of this node (They were inserted to different branches of this node with the between value, and they are not compared). Therefore, numbers between 4 and 9 must not be inserted before at least one of these two numbers was inserted. Only these numbers' order of insertions plays a role in the possibility calculation of comparing 4 and 9. Other numbers less than 4 and greater than 9 do not play any role their order does not affect this possibility.

__4 or 9__other 5 remaining numbers permutation_

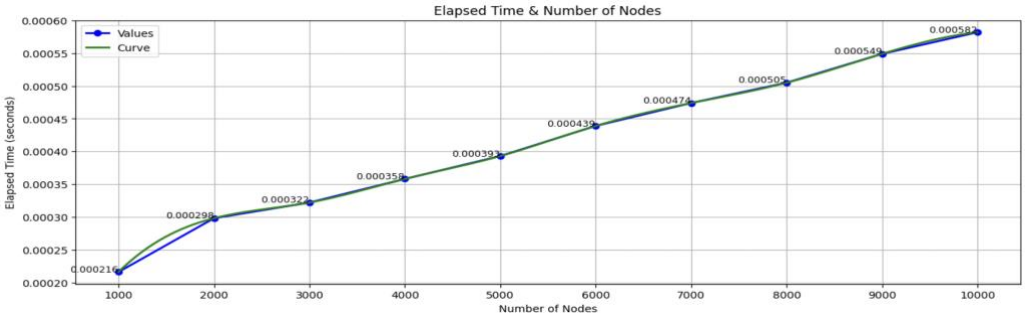
The two chosen numbers for first two must be either 4 or 9 (2!). 4 numbers left whose possible order count is 4!. There are 6! different sequences for 6 numbers. Therefore, the possibility of 4 and 9 to be compared is (2!X5!)/6! = 1/3.

**Question 3 Part 1)**



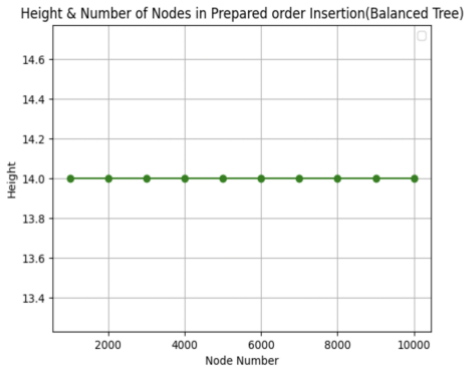
The function that can be seen in the plot looks like a logarithmic function. In a linear function, the dependent variables' values (Height of the BST) increase proportionally to the independent variables (Number of nodes inserted) however this is not the case when we look to the graph. The increase amount of the height of the binary search tree decreases as the added node number grows. Therefore, the graphic looks like logarithmic. It is the expected behavior of binary search tree insertion. According to the feature of the binary search tree one node contains smaller values in its left sub-tree and greater values in its right sub-tree. The insertion order of the values affects the total height of the binary search tree. The height of a binary search tree with n nodes can be between $\log_2(n+1)$ and n. If the insertion was in ascending or descending order the binary search tree height would be n. Since the node to be inserted will come to the left or right child always. However, the insertion process was in random order and the tree tends to be more balanced due to this. Of course, its height won't be exactly equal to $\log_2(n+1)$ in random insertion some sub-trees can have more nodes than other ones.

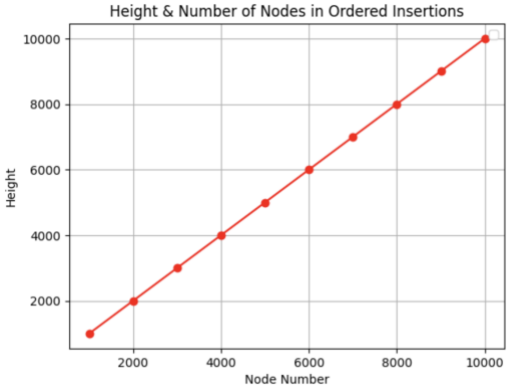
**Question 3 Part 2)**

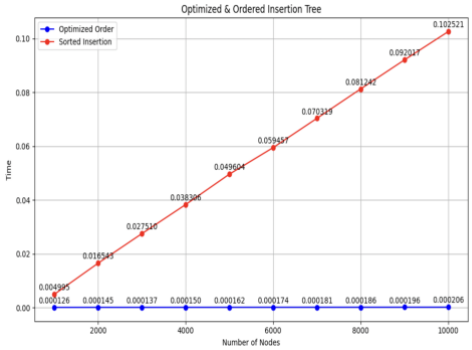


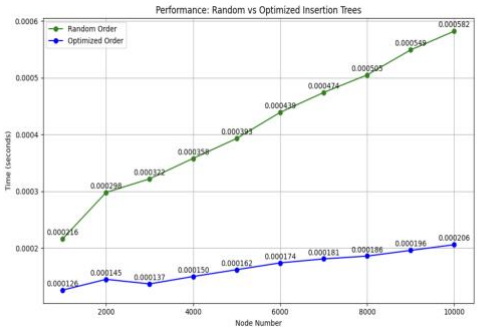

The current time complexity of inserting a value into a binary search tree with n nodes is O(logn) when the elapsed time values that corresponds to the number of nodes are considered since the values to be inserted are in random order. If it was O(n) the amount of the elapsed time would proportional to numbers of nodes that has the binary search tree but it can be seen from the graph that it is not. When we look at the elapsed time in in the 1000 node and 10000 node it can be seen that the total time did not even come close to its 10 times more value. If the sorted integers were inserted into the binary search tree the time complexity would be O(n). To give an example if the values were in ascending order the new coming node would always be replaced to the right child of the leaf node which will require n number of visits in each insertion process when tree has n number of nodes. In other possibility if the numbers were in descending order then the new node to be instead would always be inserted to the left child of the which will also require n number of node visit in a tree with n number of nodes. However when we insert it in a random order the tree becomes more near to minimum height binary search tree.

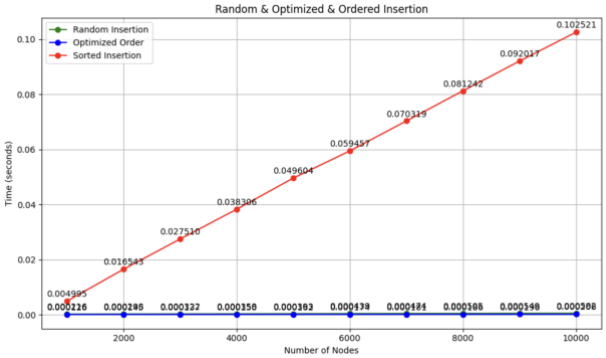
**Question 3 Part 3)**




According to the binary search tree features a node contains the greater values in its right subtree and smaller values in its left subtree. Therefore, the insertion order of the values into a binary search tree affects that binary search tree's height and the time complexity of the next insertion process. For example, in the plot that is given for an ordered insertion every node to be inserted was inserted in the right child of the leaf node since the values to be inserted were in ascending order. This increased the height by one in every insertion and lead to n visits in every insertion. Therefore, the this ordered insertion lead to an O(n) height and O(n) time complexity for insertion process. Also when we look at the graph of the random

insertion which is in the top of the report we can see that random insertion lead to not an exact balanced binary search tree. Although random insertion gives rise to a more balanced tree than inserting the values in order, some sub-trees had more nodes than others. Since the height of the binary search tree affects the insertion process's time complexity random insertion took more time than inserting the nodes into tree in optimized order.

We can keep the tree balanced if the insertion order of the values are suitable. First the values to be inserted were sorted in ascending order. Then taking the middle element (not by its value but the index in the ascending order array) and selecting it as the first to insert into the binary search tree will lead the number of the nodes in the left and right subtrees of that node to be balanced. After this process selection of the middle index from the remaining left and right part of the array recursively with the same approach will lead the other subtrees that are in the lower part of the binary search tree to also be balanced. For example after the selection of the first element from the sorted array to be inserted first when we get the middle element from the remaining left part, that middle element will come as the left child of the first selected element. And also that element will have same numbers of nodes in its right child and left child. This approach will ensure that the tree is balanced and its height becomes optimized for inserting operations. Its height will become $\log_2(n+1)$. In the insertion process the height of the tree plays a role in time complexity since the new value to be inserted will be placed by compared in each level of the tree once recursively. And since the height of the tree by this approach is minimized the binary search tree insertion operation will be $O(\log n)$. Which makes it faster than inserting the numbers randomly or in sorted order.