**Research Paper: The Exploration of Expression and Splay Trees**

Farah Mahmoud, Mai Abd El-Naby, Menna Emam, Yasmeen Mohamed

Department of Computer Science and Engineering, The American University in Cairo

CSCE 1102: Fundamentals of Comp II Lab

Dr. Esraa Hamdi

December 1, 2022

**Table of Contents**

**Abstract 3**

**Introduction 4**

**Proposed Approach 5**

**Results and Evaluation 12**

**Discussion and Conclusion 14**

**References 16**

**Appendix 18**

**The Exploration of Expression and Splay Trees**

**Abstract**

This research paper aims to identify the time complexities and executable outputs in different forms based on different input sizes and different traversal methods among expression and splay trees. The difference between the time taken to compile varied size inputs reflect the trees’ efficiency and limits. This includes the evaluation of the post, pre, and in order traversals along with their time complexities against the expression binary tree. The structure of the expression tree resembles a stack using linked list data structure to insert and calculate equations using a time efficient and professional form of coding style. The splay tree, on the other hand, is a self-adjusting binary search tree that has an expression tree-like structure and functionality. Instead of speeding up each individual task, the splay tree's goal is to speed up the entire sequence of operations ("Splay Trees," 2022).

*Keywords: expression trees, stack, splay trees, time complexity, traversals*

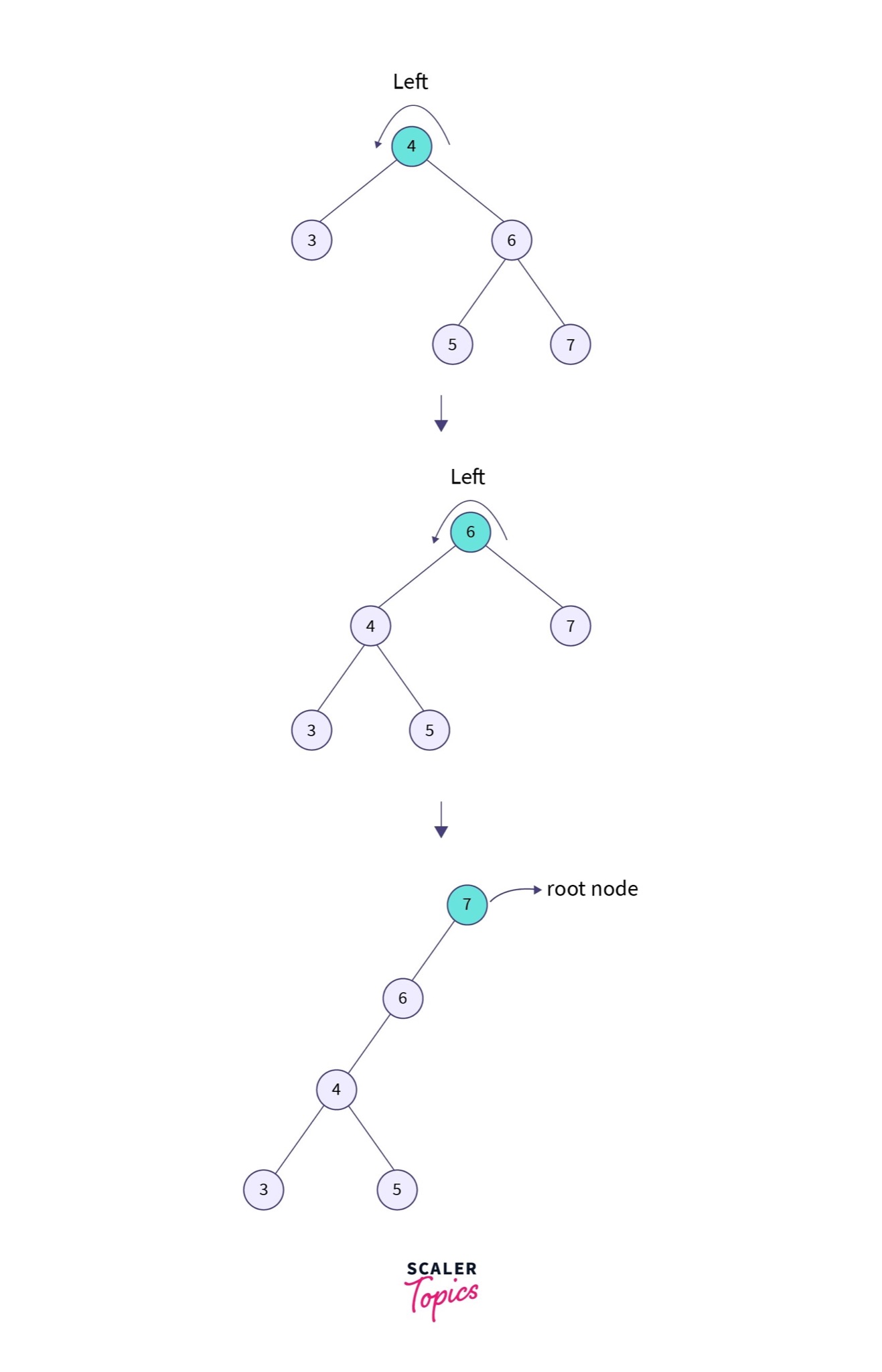
**Introduction**

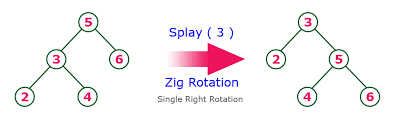
A tree is a non-linear, hierarchical data structure that is composed of nodes and edges. Their value is represented by nodes, which are linked together via edges. The following characteristics of trees are the root-named node that makes up the tree which is where the tree originates from, so it doesn't have any parents. Despite having more than one child, each node only has one parent. Each node has an edge connection to its children. The last nodes that have no children are referred to as leaf nodes ("Great Learning Team", 2020). In this paper, the main focus of exploration are expression and splay trees. Expression trees’ follow the procedure of the parent node containing the operator and the leaf nodes containing the operands to be calculated. It adopts the binary tree system in which each parent node is connected to a limit of two children. The stack implementation of the expression tree aids in the construction of the output using the push and pop mechanisms. The obscure component this research in expression trees aims to achieve is the time measurement aspect during variously sized data intake. On the other hand, the splay tree is a self-adjusting binary search tree with a contrast structure and function to an expression tree. The splay tree's purpose is to speed up the process of operations rather than each individual activity("Splay Trees",2022). All of the operations of a binary search tree, such as insertion, deletion, and searching, are available in a splay tree. However, it also includes another operation known as splaying. Every operation in a splay tree is performed at the tree's root . Splaying is the process of getting an element to the root by appropriate rotations. By doing a sequence of single and double tree rotations, the splay tree moves node x to the tree's root for the search operation. Every double rotation moves x to its grandparent's position, whereas every single rotation moves x to its parent's position. We repeat these rotations until x reaches the tree's root. This is known as splaying(moving a node to the root). Splaying, in addition to shifting x to the root, shortens the height of the tree, making it more balanced. Single rotations are classified into two types while double rotations are classified into four types. Each of them is discussed in depth below.

**Proposed Approach**

The analysis of the expression tree through its uses requires the understanding of its structure and flow of input through its functions. The formatting of an expression for display maintains the same structure as that used to represent or evaluate it. Equations can also be simplified using symbolic manipulation, which is another use ("What are the uses of binary expression trees?"*, n.d.)*. This indicates that the code represents a sequence of expressions and data to be calculated that can be changed before run time to display different equations. Compilers use them to represent logical and numeric expressions in algorithms. It can also aid in the understanding of the traversals and the order in which the data is evaluated through the appliance of the equation through code. The different types of traversals and the recursive process are understood through the input of data and comparing it to the results of the traversal outputs. The input is presented in prefix order to better comprehend the process of application of the expression tree as it pushes the operators in parent nodes and operands in leaf nodes as prefix order intakes the operators first because of the nature of the order. Whilst that is easier to understand, it’s not a requirement to insert values in prefix order as it would perform the same in other orders. If the character is an operand, it is pushed into the stack. While if it is an operator, the two values in the stack are popped to be the children. Then, the operator is pushed into the stack as the parent node. To prove the time complexity, the time used for compilation is calculated for different input sizes and for also different types of traversals using the same size.

As for the splay trees, the main concept is the rotation types that are unique to facilitate the insertion and deletion process. The single rotations are classified into two types while double rotations are classified into four types("Splay Trees",2022). First, the single rotation are Zig rotation which is similar to the right rotation in the AVL tree.Every node in zag rotation advances one step to the right of its current position(as shown in diagram 1 & 2)



Diagram 1 &2

.also there is Zag rotation which similar to the left rotation in the AVL tree is this rotation. Every node advances one place to the left of its present location during a zag rotation (as shown in diagram 3).

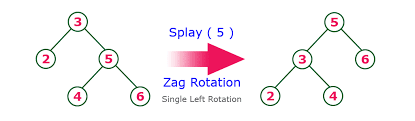


Diagram 3

Moreover , there are double rotations which are classified into four types. The first one is Zag zag rotates in a sort of double zag pattern(twice). Every node shifts two spaces to the left of its current position(as shown in diagram 4).

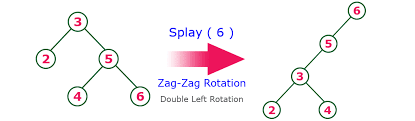


Diagram 4

The second one is Zig zig and it's similar to a double zig rotation. Every node shifts two spaces to the right of its current position(as shown in diagram 5).

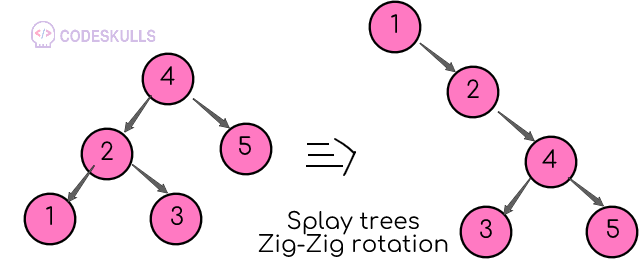


Diagram 5

The purpose of these last two types is that there are times when we must search for something that has both parents and grandparents. In these circumstances, splaying requires four rotations.The Zig zag is a rotation made up of zig rotations followed by zag rotations. This means that every node advances one place to the right of its current position, followed by one position to the left(as shown in diagram 6).

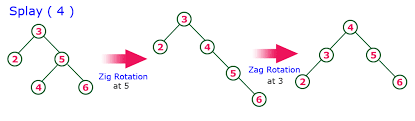


Diagram 6

The only difference between The Zag zig and the Zig-zag rotation is that each node moves one place to the left, followed by one position to the right of its current position(as shown in diagram 7).

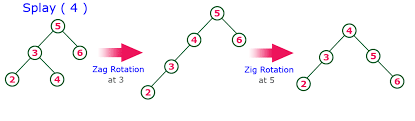


Diagram 7

We can use this for the search operation in a splay tree which is identical to a search operation in a regular search. The difference is that after the search operation, we splay the node,and to find a key, we perform the following steps. First we run the standard binary search tree search,then assume that the key is discovered in node x,and then splay the key("Splay Trees",2022).

**Insertion in splay tree:**

The insertion process in the splay tree is similar to that of a binary search tree, with additional steps to ensure that the newly inserted key becomes the new root (Imms, 2019).

Following are different cases to insert a key k in the splay tree ("Insertion in splay tree", 2022):

1) Root is NULL: allocate the new node location, add the key, and return it as the root.

2) If the key k is already present, splay it. Then it becomes the new root of the tree.

If it is not present, the last accessed leaf node will subsequently become the new root.

3) If the key to the new root is the same as k, do nothing because k is already present.

4) Else, allocate memory for the new node. Then, compare the root’s key with k.

There are two cases:

a) If k is smaller than the root’s key, make the root the right child of the new node copyleft child of the root the left child of the new node. Then, make a left child of the root NULL.

b) If k is greater than the root’s key, make the root the left child of the new node and copy the right child of the root as the right child of the new node. Then, make the right child of root NULL.

5) Return the new node as the new root of the tree.

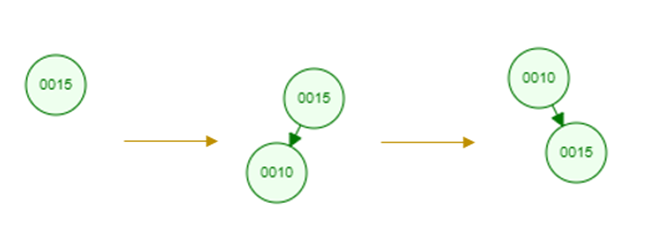
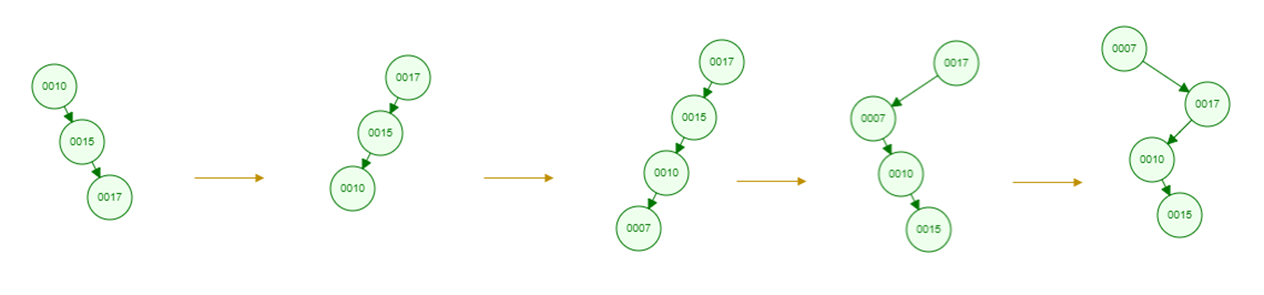
The insertion process is illustrated in diagrams 8,9 with different cases. 

Diagram 8

Diagram 9

**Deleting a node from a splay tree:**

The following are the different cases of deleting a key k from a splay tree ("Deletion in splay tree", 2022)

1) If Root is NULL: return the root.

2) If else, splay the given key k. If k is present, it becomes the new root. The last accessed leaf node will become the new root if it is not present.

3) If the new root’s key is not the same as k, then return the root because k is not present.

4) Else the key k is present.

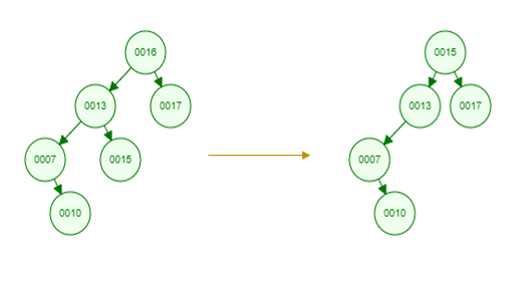
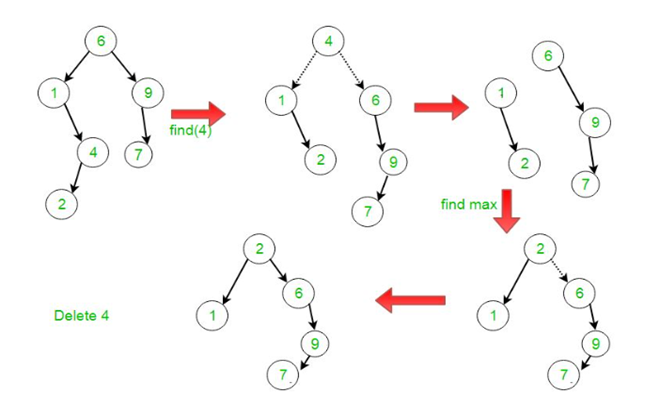
- Divide the tree into two trees: Tree1 represents the left subtree of the root, and Tree2 represents the right subtree, and remove the root node and

- Let the root of Tree1 and Tree2 be both Root1 and Root2, respectively.

- In case Root1 is NULL: Return Root2.

- Else, Splay the maximum node (the node having the maximum value) of Tree1.

After finishing the splaying procedure, make Root2 the right child of Root1 and return Root1. All cases are illustrated in diagram 10&11.

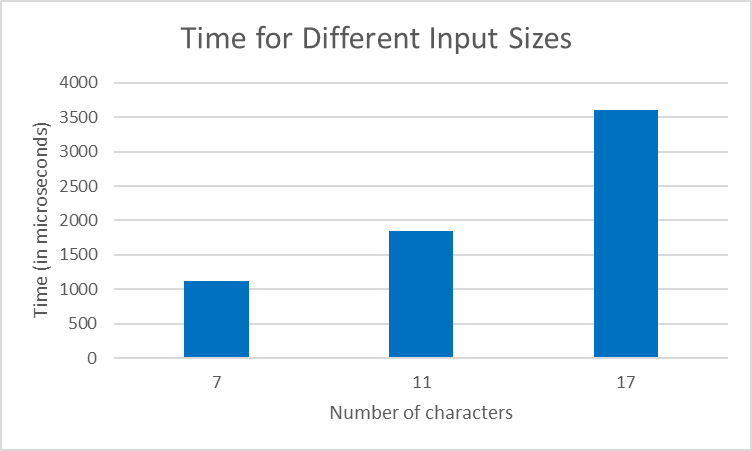
Diagram 10 Diagram 11 

**Traversing through splay tree:**

The same types as the binary search tree: pre-order, in-order, and post-order traversal. The methodology used to prove the time complexity was by inserting three varying sized inputs and calculating the time by utilizing a timer that starts after the user enters the expression and ends after all the functions are executed including the order traversals outputs. The same method was used to calculate the time spent in each traversal type.

**Results and Evaluation**

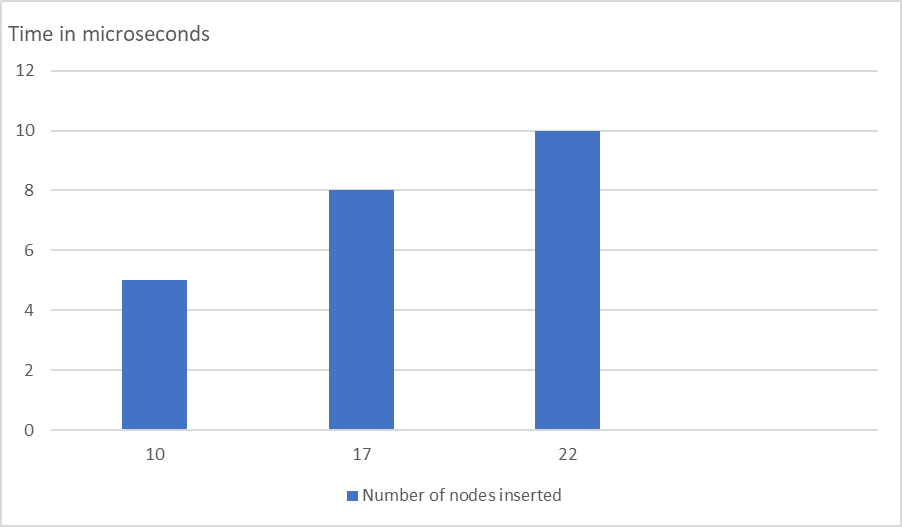
The measurement of time of varying input sizes in the expression tree reflects a decrease in calculated run time with every decrease of value intake. Therefore, the time complexity is O(n) as the values entered decrease the time taken to compile the algorithm decreases and vice versa. Lastly, the pre order, postorder, and inorder traversals only differ in the order in which the nodes are visited, the time of compilation doesn't showcase much difference since the time complexity depends on the number of nodes visited (O(n) complexity). As the order of the traversal doesn’t contribute to any change in time, the time calculated stayed similar with minor significance.



|  |  |
| --- | --- |
| Type of Traversal | Time (in microseconds) |
| Pre-order | 1774 |
| In-order | 1314 |
| Post-order | 1503 |

In the splay tree, the traversal algorithms are the same as in the binary search tree: pre-order, in-order, and post-order. They only differ in the order in which the nodes are visited. Therefore, almost all traversals take the same time with a slight difference. The results of different traversal algorithms with a fixed tree size were obtained as follows:

|  |  |
| --- | --- |
| Type of Traversal | Time (in microseconds) |
| Pre-order | 621 |
| In-order | 656 |
| Post-order | 707 |

The measurements of time complexity in inserting different input sizes showed an increase as the input size increases. The model time complexity of the splay tree is O(log n) for insertion. The following results was obtained by inserting different input sizes: 

**Discussion and Conclusion**

Given that the time complexity of the expression tree is O(n), the gradual increase in time accompanied by the increase in the characters in the equation ie. the number of nodes is portrayed in the results. The time calculated with respect to each type of traversal is equal as they all pass each node but in different orders. The expression tree does not follow the binary process but it has a unique technique that differentiates it from other trees in order to calculate equations. The purpose of the tree is to store and organize the equation in a data structure that’s easy to traverse through and can accept multiple different forms of the equation as prefix, post fix, and in fix. The stack implementation is helpful as the LIFO method facilitates the process of calculation of the expression tree, especially the pop of the last two digits entered in the list that are to be calculated.

The time complexity of insertion, deletion, traversal, and searching in a splay tree is O( log n). The measurements of time complexity in inserting different input sizes showed an increase as the input size increases. Similarly, the time complexity of different traversals increases by increasing tree size. However, the experimental complexity obtained from different trials was slightly higher than the model complexity. However, this happens for various reasons. First, in any algorithm, many factors and coefficients are ignored while calculating the theoretical complexity. There could be ignored factors in splay tree complexity as well. In addition, the time complexity of an algorithm is more accurate when calculated from a large input size. In the splay tree, the input sizes used were small compared to what scientists used to measure complexity. Nevertheless, the complexity obtained from our iterations was so close to model one indicating more accurate results with larger input sizes.

As for splay trees, there are some advantages and disadvantages. One of the main advantages is that splay trees do not need an extra piece of information, unlike the AVL and Red-Black trees. In AVL trees, the balancing factor of each node has to be calculated. In red-black trees, the color of the node has to be saved. Therefore, one of the fastest forms of binary search tree is the splay tree, which is used in a number of practical situations such as GCC compilers. It also puts regularly visited nodes closer to the root node to improve searching. One of the practical uses is cache implementation, which saves recently used data in the cache so that it is able to access the data more rapidly without having to travel into memory("Searching in Splay Tree",2022). However the main disadvantage of the splay tree is that it is not perfectly balanced, but rather roughly balanced.

**References**

Cheruku, R. T. (2011, December 13). *Splay tree - Indiana State University*. Splay Tree, Indiana State University Terre Haute,IN,USA. Retrieved November 21, 2022, from<https://cs.indstate.edu/~rcheruku/splaytree.pdf>

*Deletion in splay tree*. GeeksforGeeks. (2022, November 28). Retrieved November 20, 2022, from<https://www.geeksforgeeks.org/deletion-in-splay-tree/?ref=gcse>.

*Expression tree in c++*. (2022, February 1). Sanfoundry. <https://www.sanfoundry.com/cpp-program-implement-expression-tree/>

*Great Learning Team*. (2020, November 3). Understanding trees in data structures. Great Learning Blog: Free Resources What Matters to Shape Your Career! https://www.mygreatlearning.com/blog/understanding-trees-in-data-structures/

Imms, D. (2019, November 10). *Splay tree*. Growing with the Web. Retrieved November 21, 2022, from<https://www.growingwiththeweb.com/data-structures/splay-tree/overview/>

*Insertion in splay tree*. GeeksforGeeks. (2022, November 28). Retrieved November 20, 2022, from<https://www.geeksforgeeks.org/insertion-in-splay-tree/>

*Searching in Splay Tree* (2022, November 28) *GeeksforGeeks*. Retrieved November 27, 2022, from <https://www.geeksforgeeks.org/searching-in-splay-tree/>

*Splay trees*. (n.d.). CodesDope. Retrieved November 27, 2022, from <https://www.codesdope.com/course/data-structures-splay-trees>

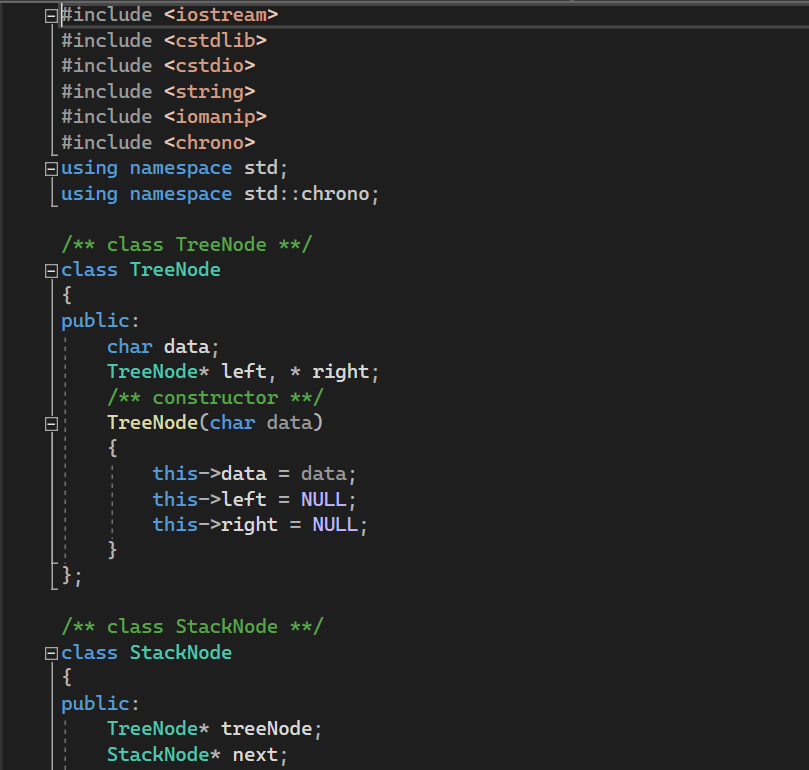
*Splay trees (with implementations in C++, Java, and python)*. (2019, February 27). Algorithm Tutor. Retrieved November 20, 2022, from<https://algorithmtutor.com/Data-Structures/Tree/Splay-Trees/>

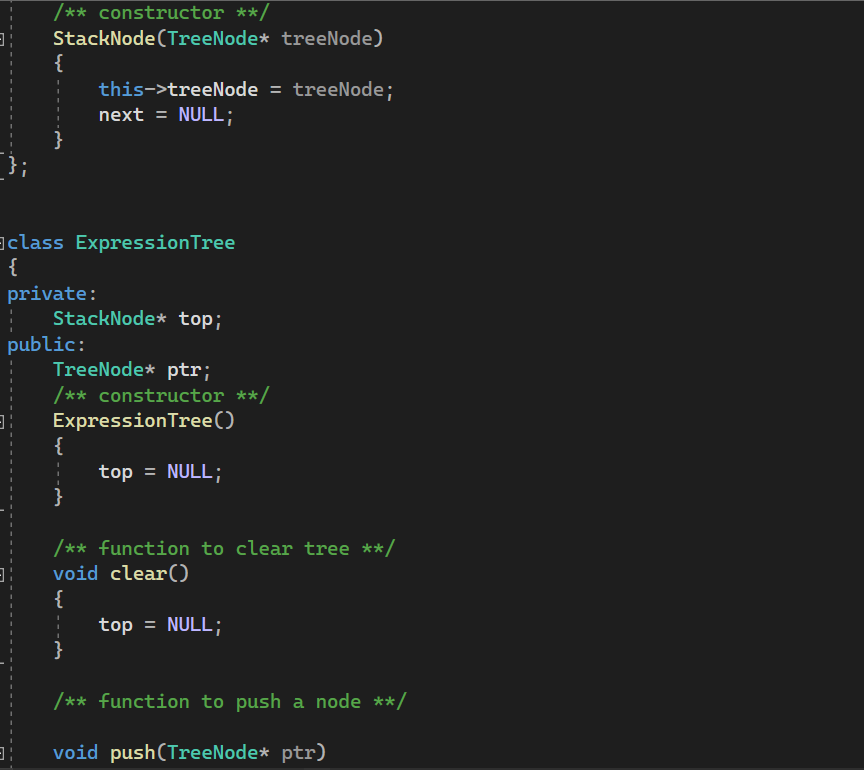
Tripathi, S. (2022, April 18) *Splay tree in data structures*, *Scaler Topics*. Retrieved November 27, 2022, from <https://www.scaler.com/topics/data-structures/splay-tree/>

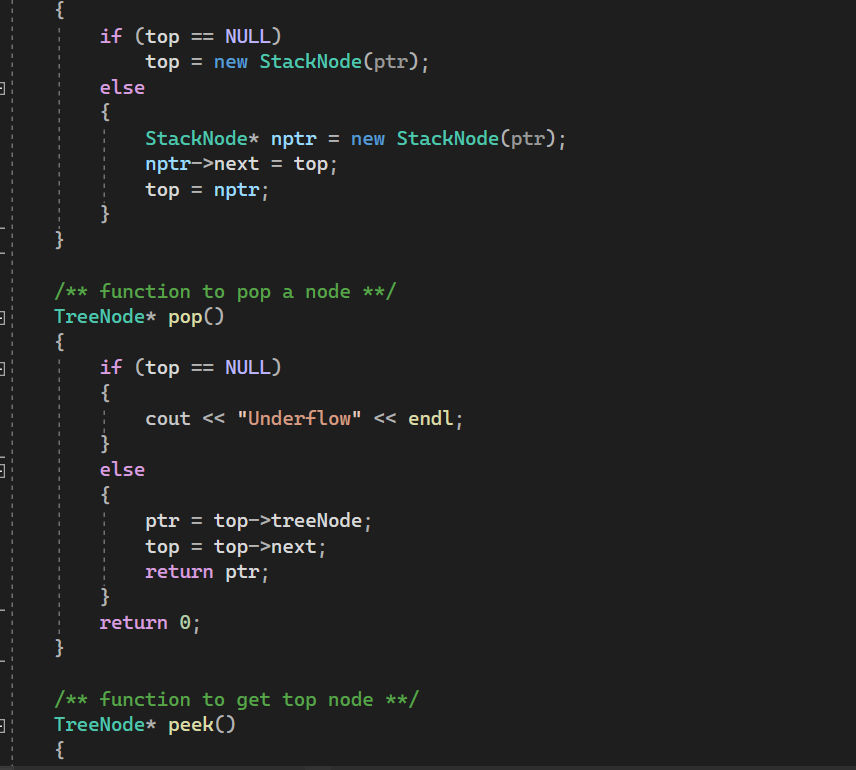
*What are the uses of binary expression trees?* (n.d.). Quora. Retrieved November 27, 2022, from https://www.quora.com/What-are-the-uses-of-Binary-Expression-Trees

**Appendix**

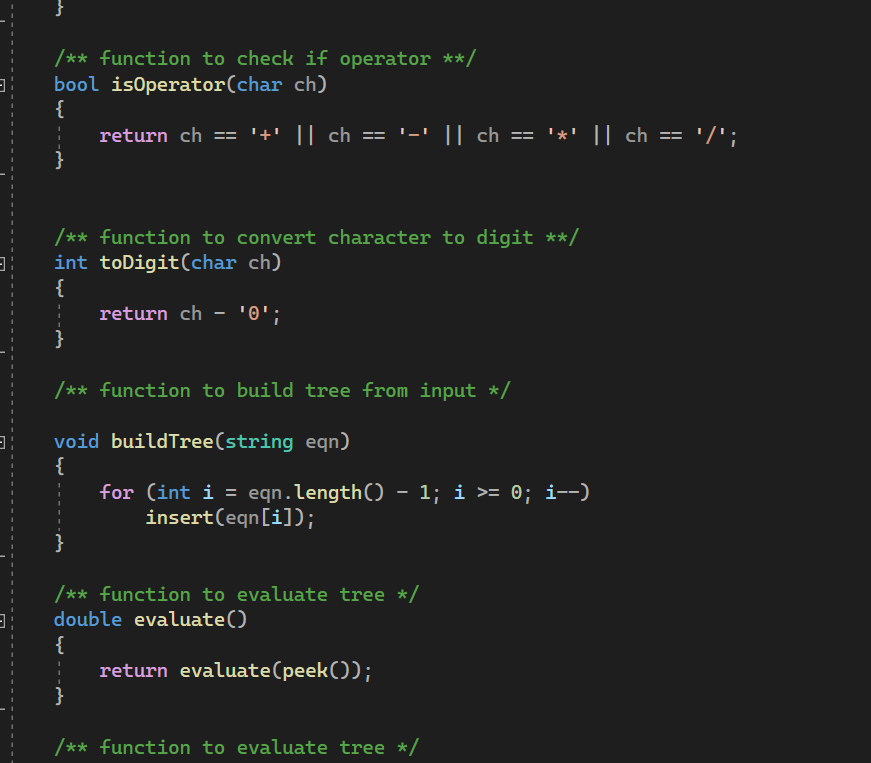
Expression tree implementation: (Expression tree in c++, 2022)

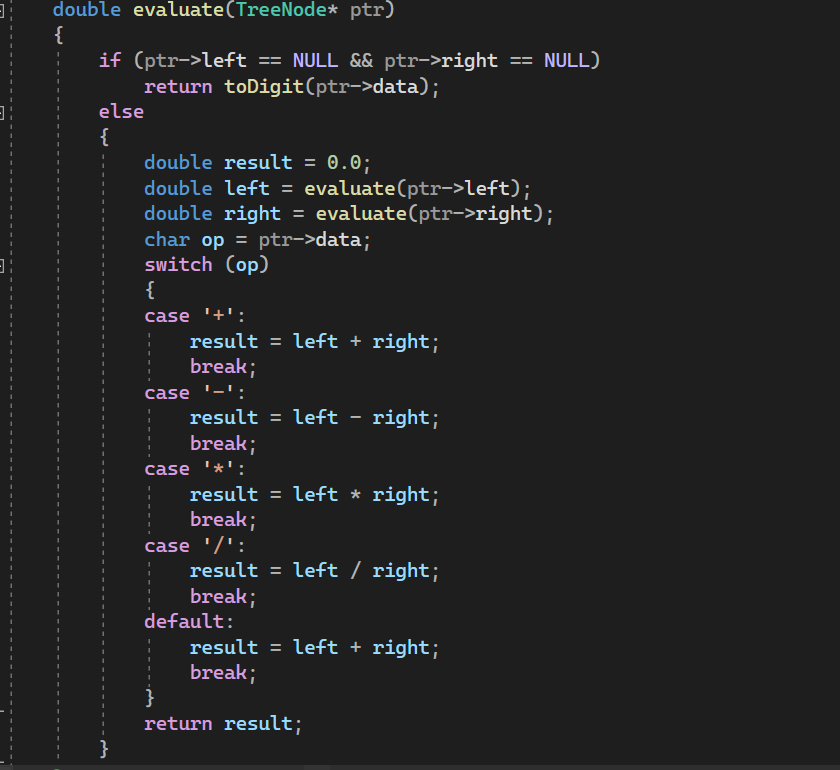






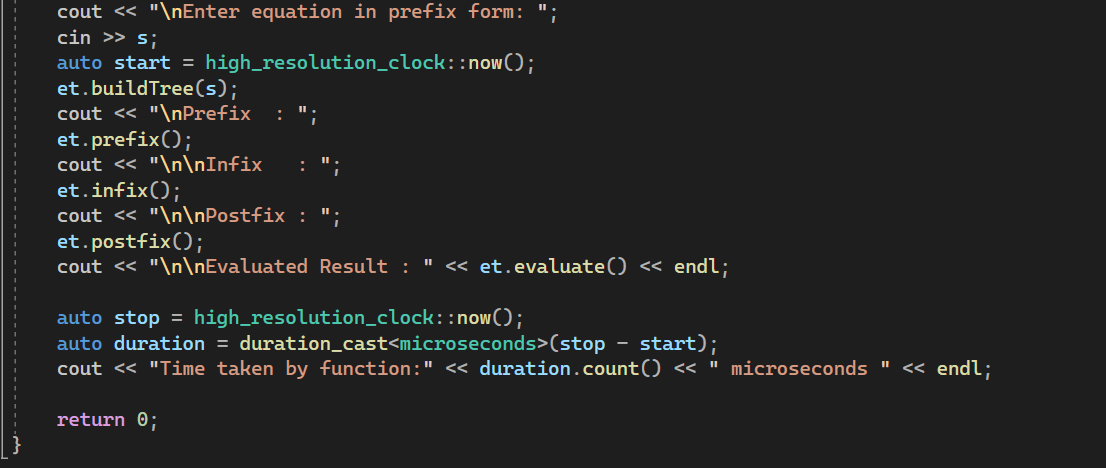






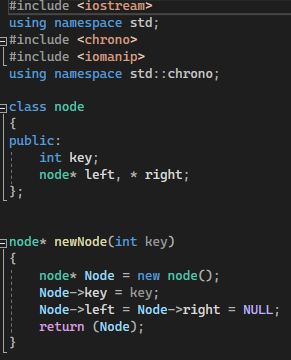




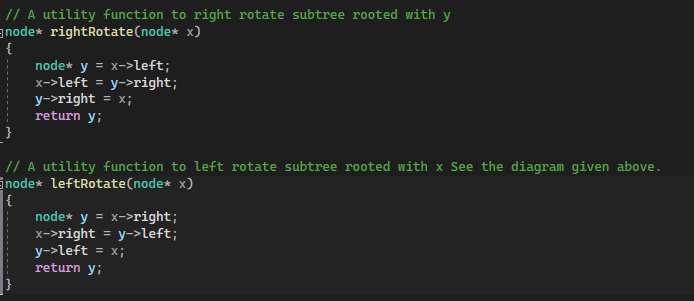


Splay tree implementation: (“*Insertion in splay tree*”, 2022), (“*Deletion in splay tree”*,2022)

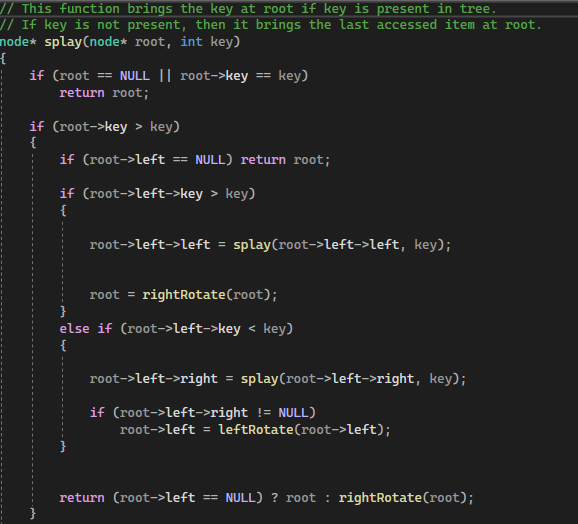
1. Creating splay tree’s class

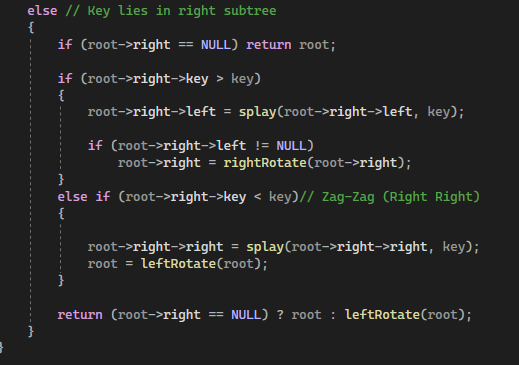


1. Rotation functions (left and right rotations)

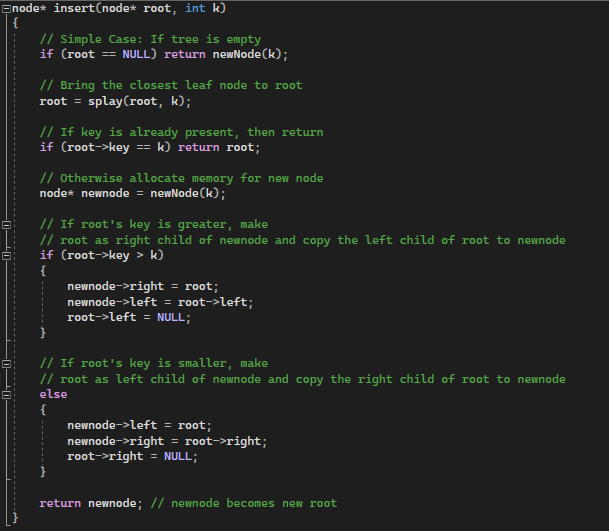


1. Splaying function

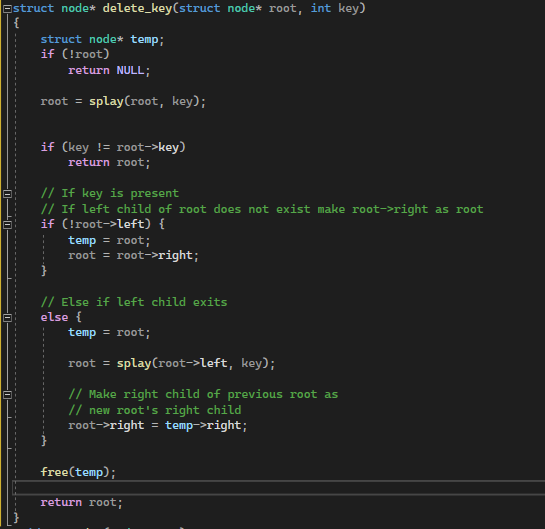




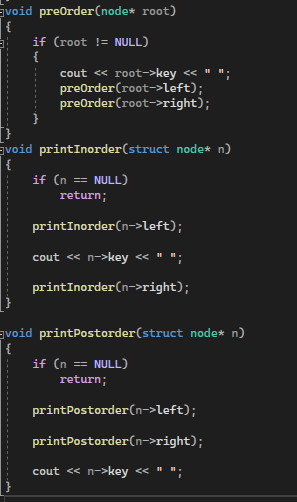
1. Insertion function



1. Deletion function

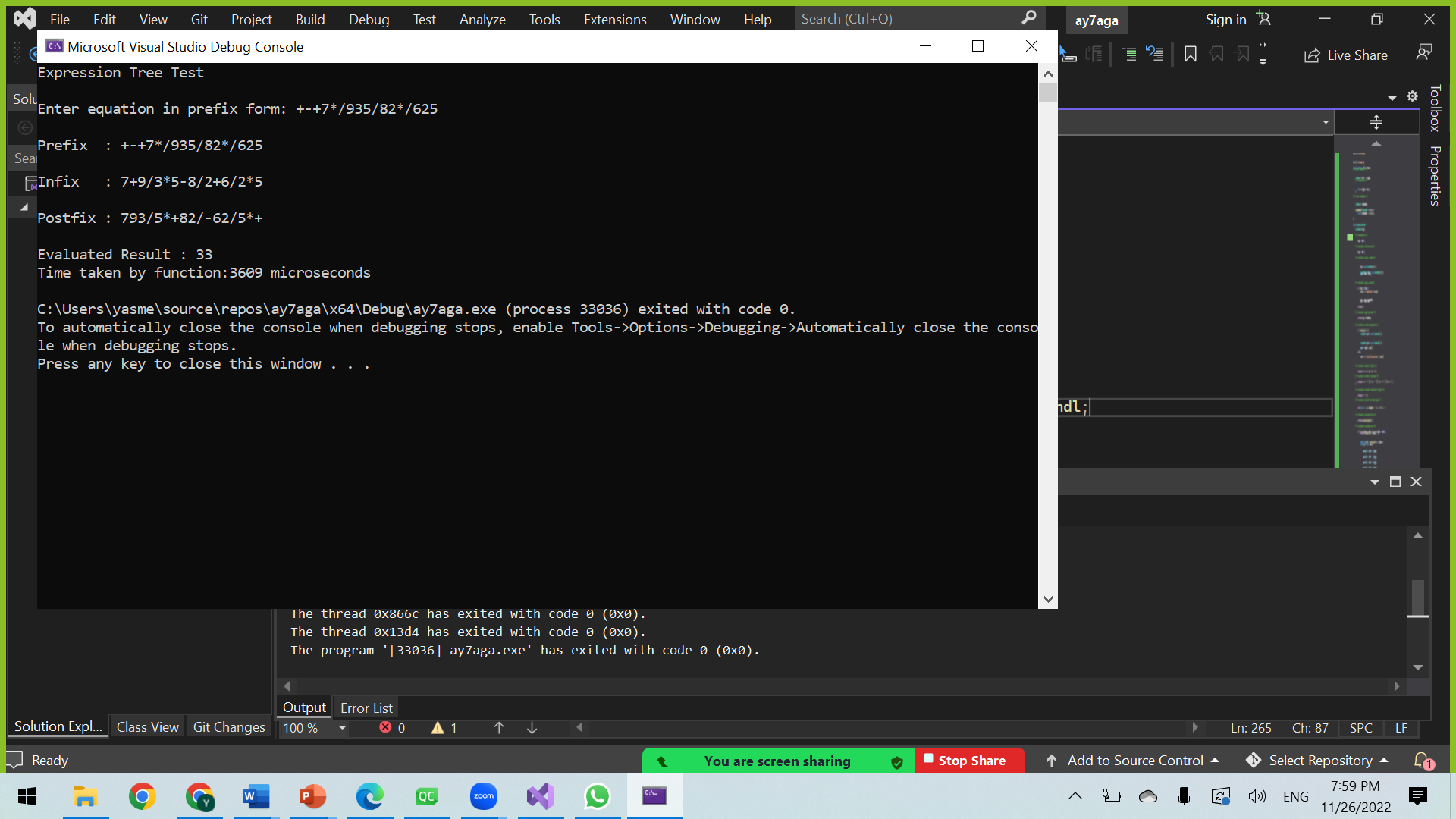


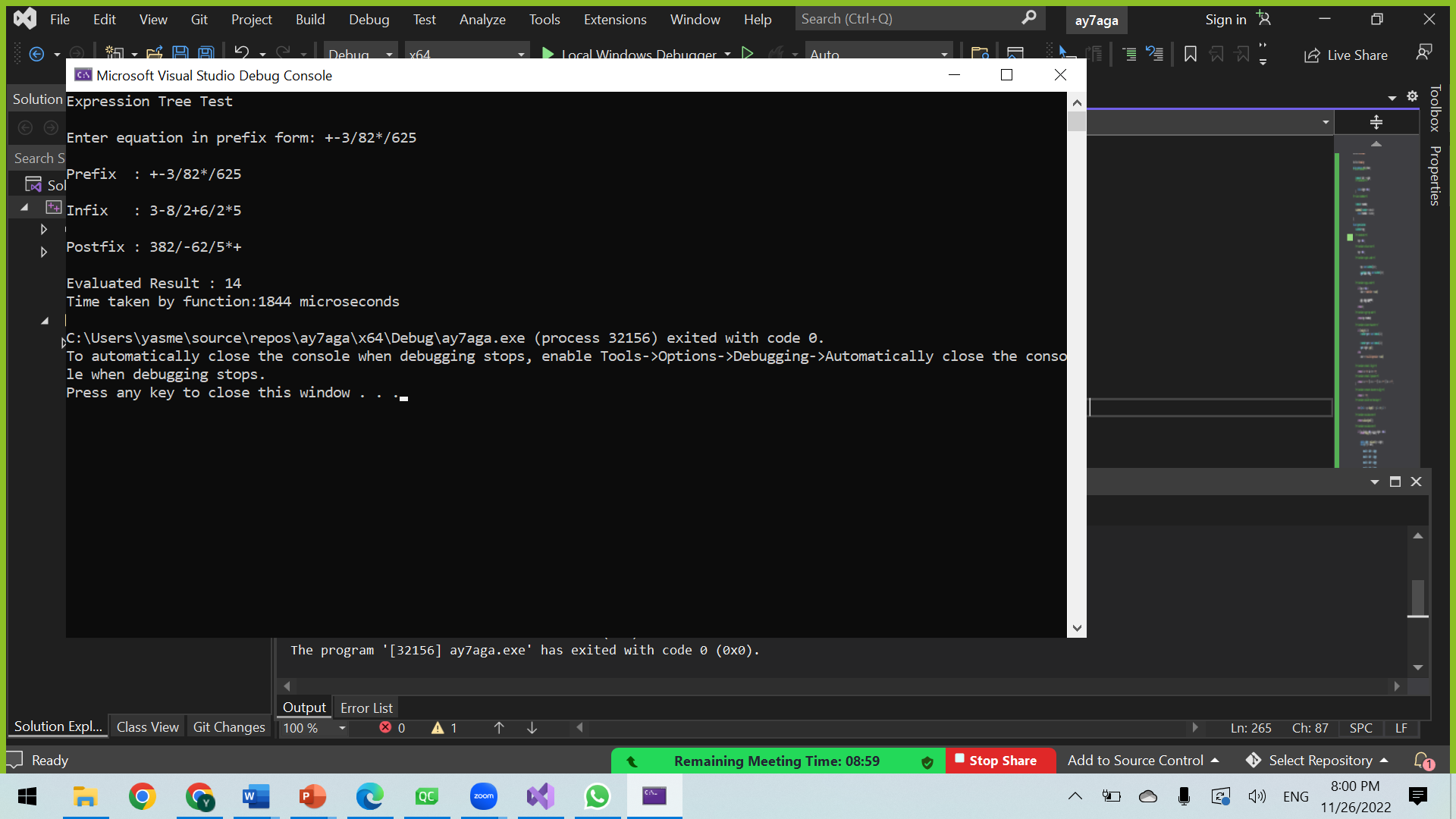
Traversal functions:

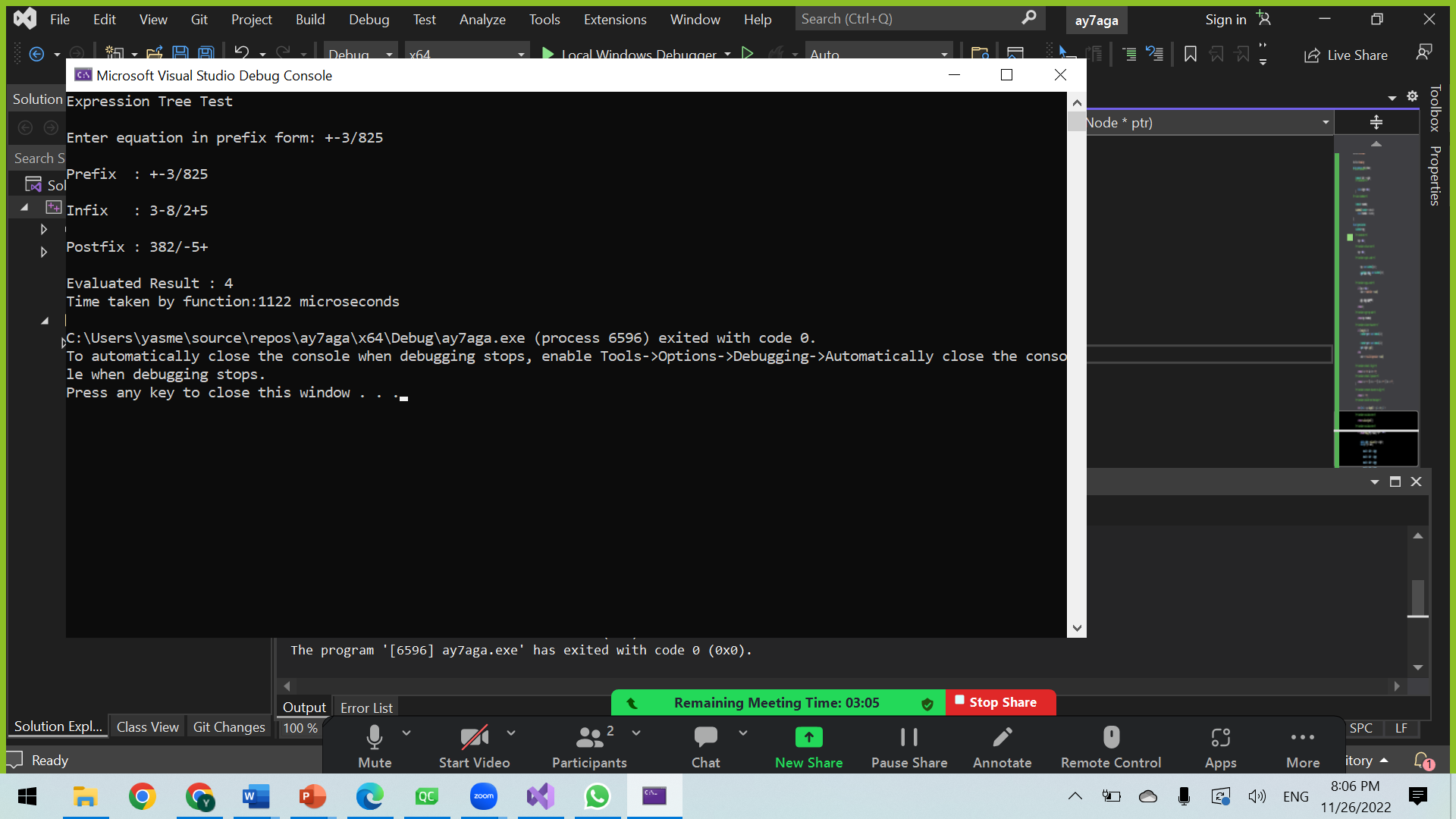


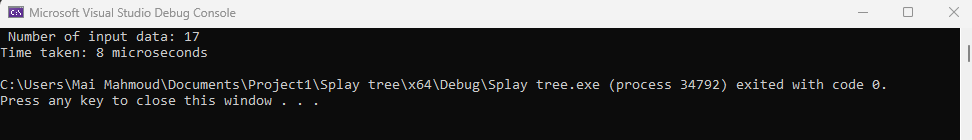
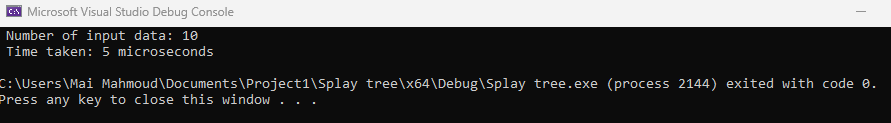
Measurement of time of different input sizes:

1)Expression tree

****

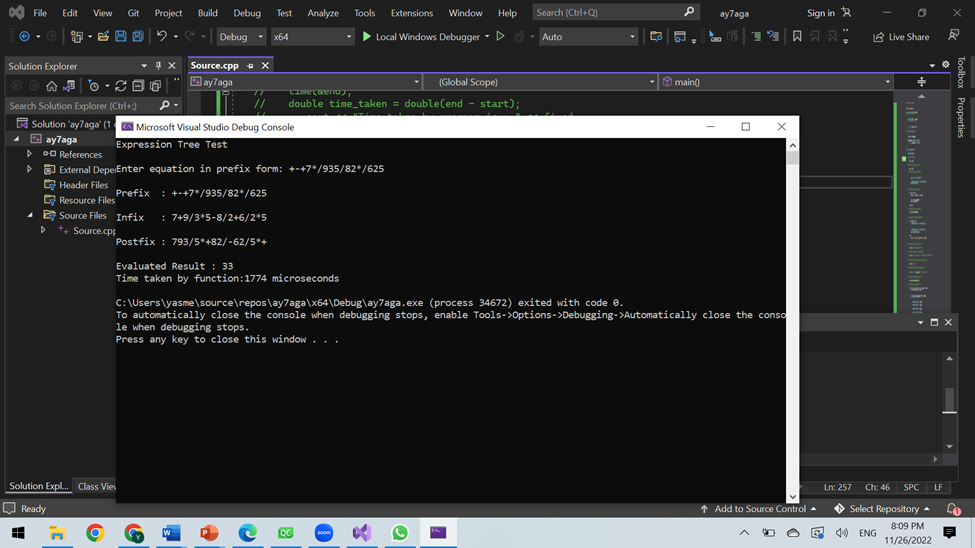
****

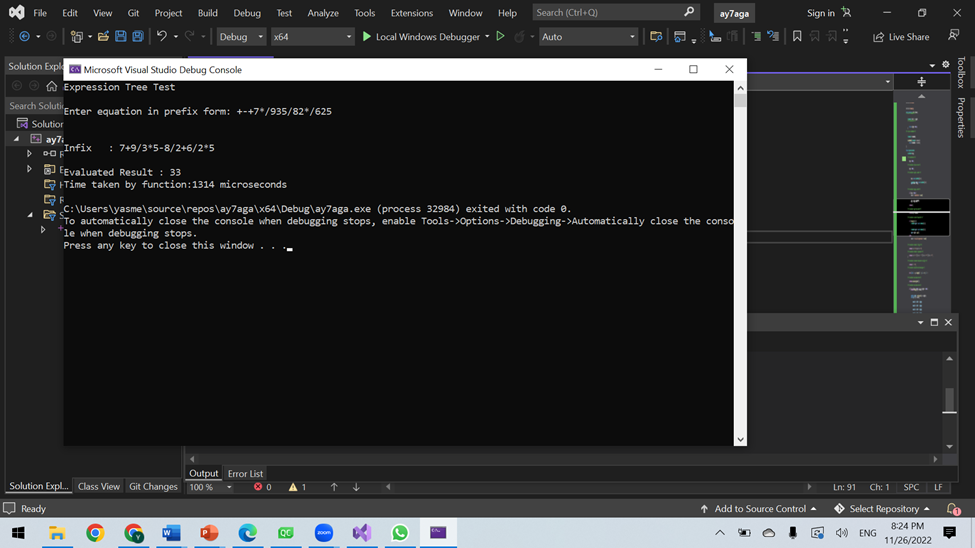
****

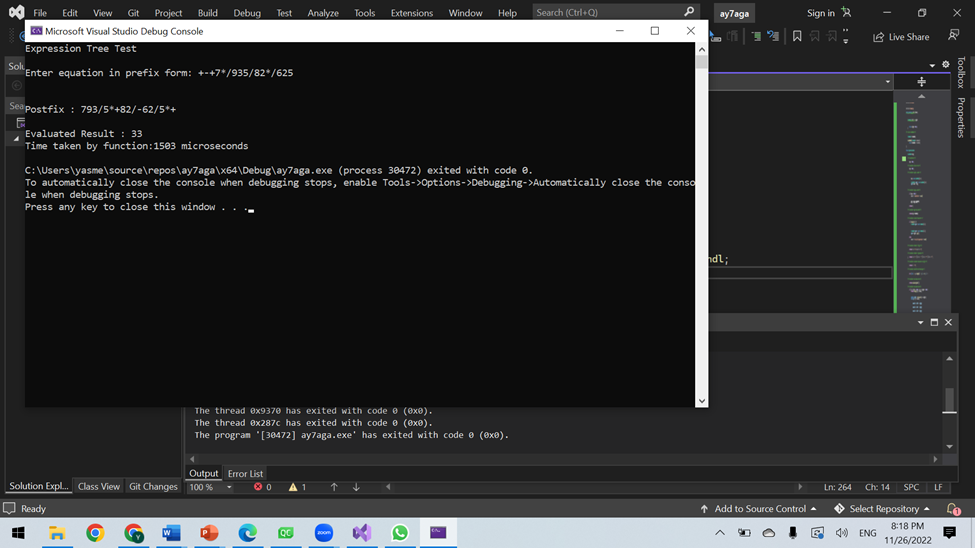
2) Splay tree:

Measurement of time of different traversals:

1. Expression tree:

****

****

****