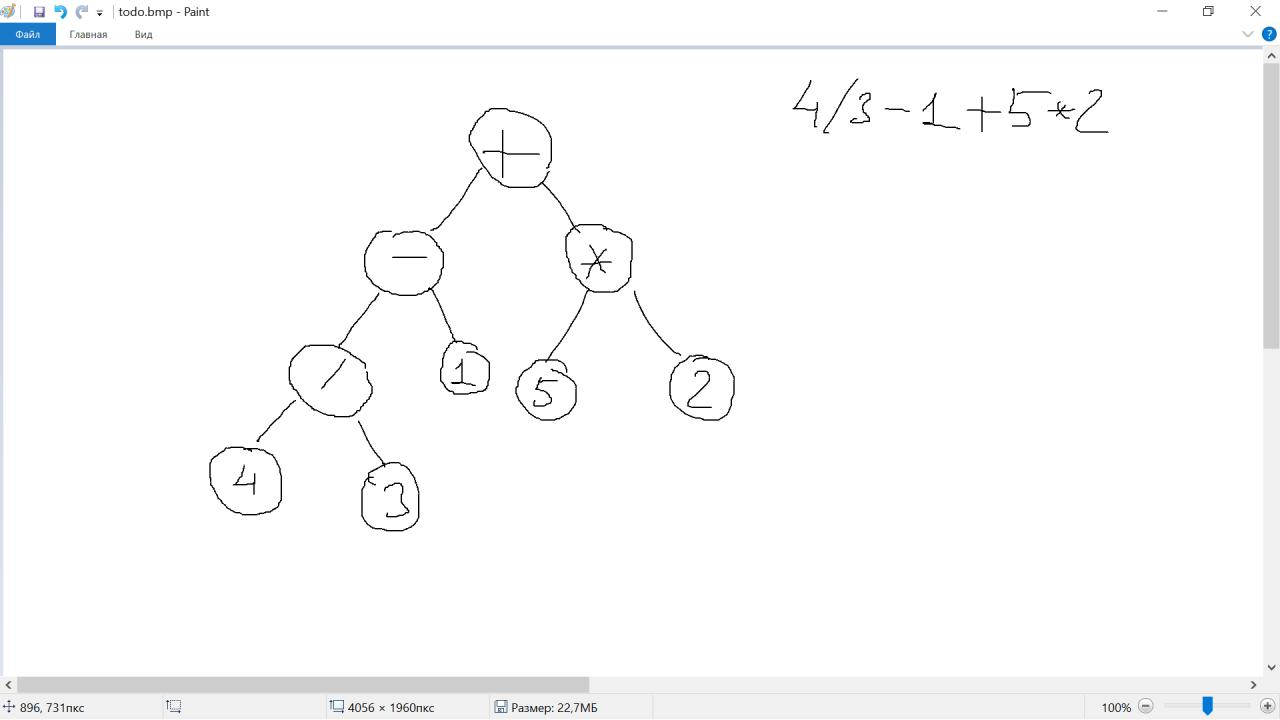
**Binary Search Trees**

1. Create a classes **BinarySearchTree<T>** and **Node<T>. Node** class must have right and left child at least(if you need you can create a parent attribute). Also **Node** class must have a data(T obj).

**BinarySearchTree** must have ***at least*** following:

* needed constructors.
* insert(T obj): inserts an element to it’s place
* delete(T obj): removes object from the tree (note: we have 3 cases of deletion)
* find(T obj): returns the node with given element, if not found returns null
* print(): outputs all elements in “in-order” way(use in-order traversal over a tree)
* smallest(): returns smallest element in a tree
* largest(): returns largest element in a tree

1. Create a class **ExpressionTree** (also use Node class). This tree is used to solve small mathematical expressions like (“5+3\*6/2-2”). So each node can hold either a number or a operand. Nodes with numbers are only leafs and nodes with operands are only parents. Least significant operands (like + and -) are first operands to take as a parent nodes. Note: for this task we use only 1-digit numbers (from 0 - 9). After building an expression tree you must show the answer of the expression. For example the given below example must show **10.3**. Note: use post-order traverse to solve expression.



.