# CMSC350 Project 3 Documentation

By Brian Yu

## Goal

The goal of this project was to develop a list-sorting tool that would read in a unsorted list of integers or fractions, insert the elements of the list into a binary search tree, and then traverse the binary search tree inorder or reverse-inorder to print out a sorted list by ascending or descending order. The tool will validate the unsorted list to make sure only integers or properly formatted fractions (based on the selection) are entered.

## UML Diagram

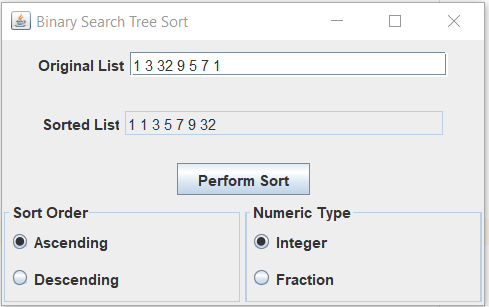
## 

## Test Cases

I performed basic functional testing to make sure my program could handle the expected uses cases of a Binary Search Tree Sorter. Specifically, I made sure that the sorter would throw an error for non-integers and improperly formatted fractions.

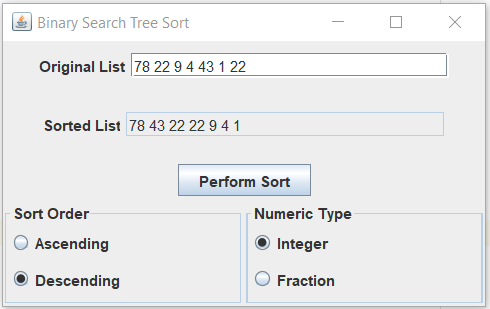
### Test Case 1

This test case inputted an unsorted list of integers (with duplicates) and requested an ascending sort. I inputted “1 3 32 9 5 7 1”. The resulting sorted list was “1 1 3 5 7 9 32” as expected. This test case passes.



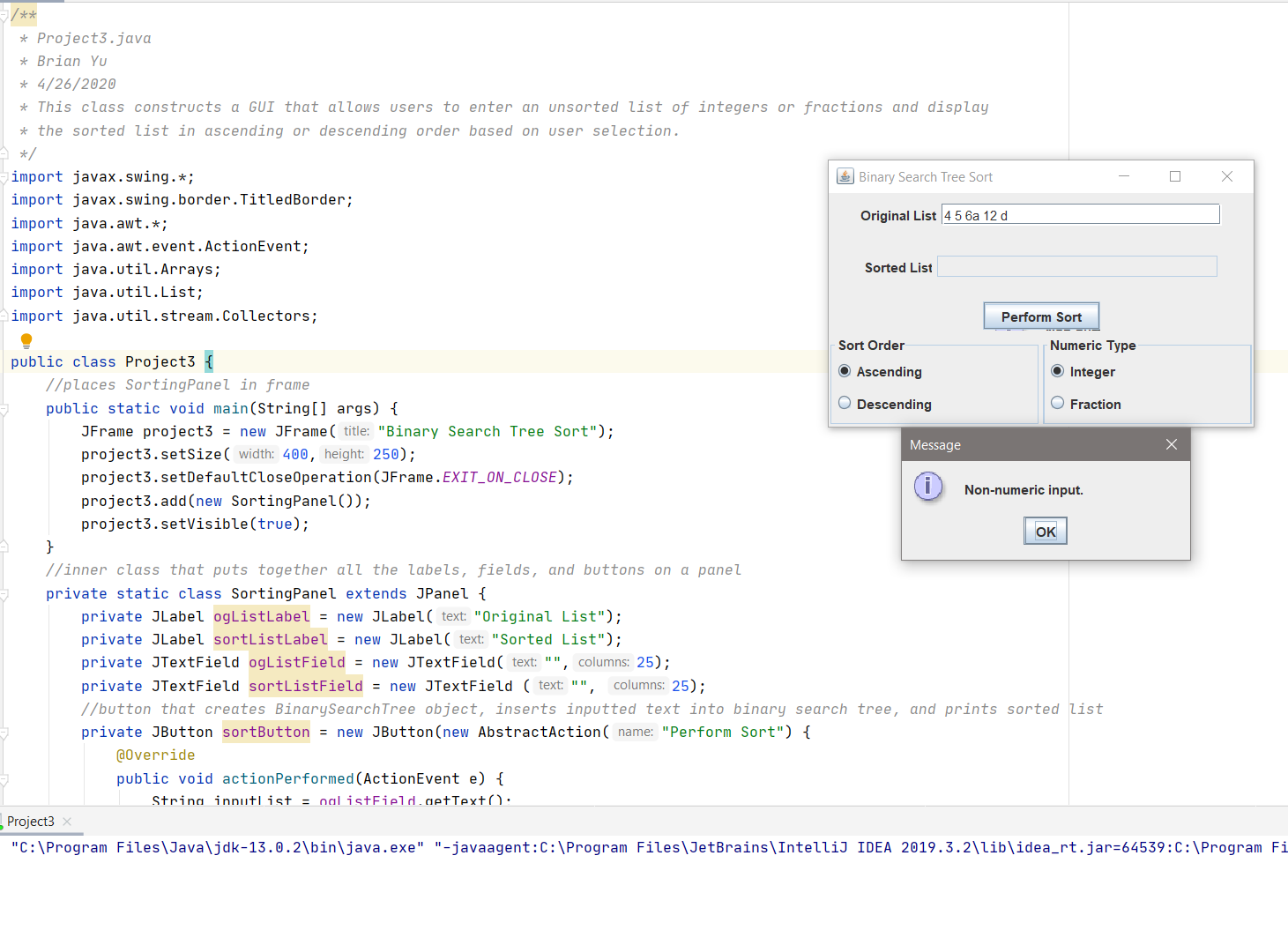
### Test Case 2

This test case inputted an unsorted list of integers (with duplicates) and requested a descending sort. I inputted “78 22 9 4 43 1 22”. The resulting sorted list was “78 43 22 22 9 4 1” as expected. This test case passes.



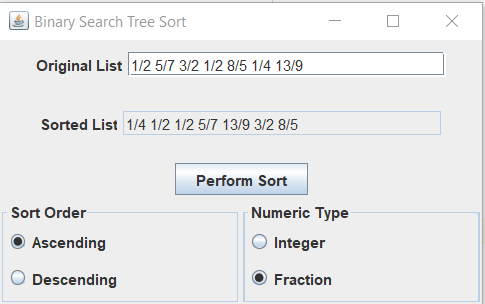
### Test Case 3

This test case inputted an unsorted list of integers (with alphabetic characters) and requested an ascending sort. I inputted “4 5 6a 12 d” and an error pop-up was displayed as expected. This test case passes.



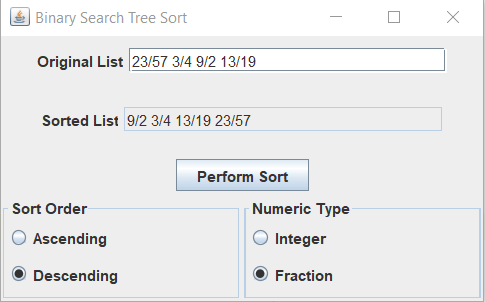
### Test Case 4

This test case inputted an unsorted list of fractions (with duplicates) and requested an ascending sort. I inputted “1/2 5/7 3/2 1/2 8/5 1/4 13/9". The resulting sorted list was “1/4 1/2 1/2 5/7 13/9 3/2 8/5” as expected. This test case passes.



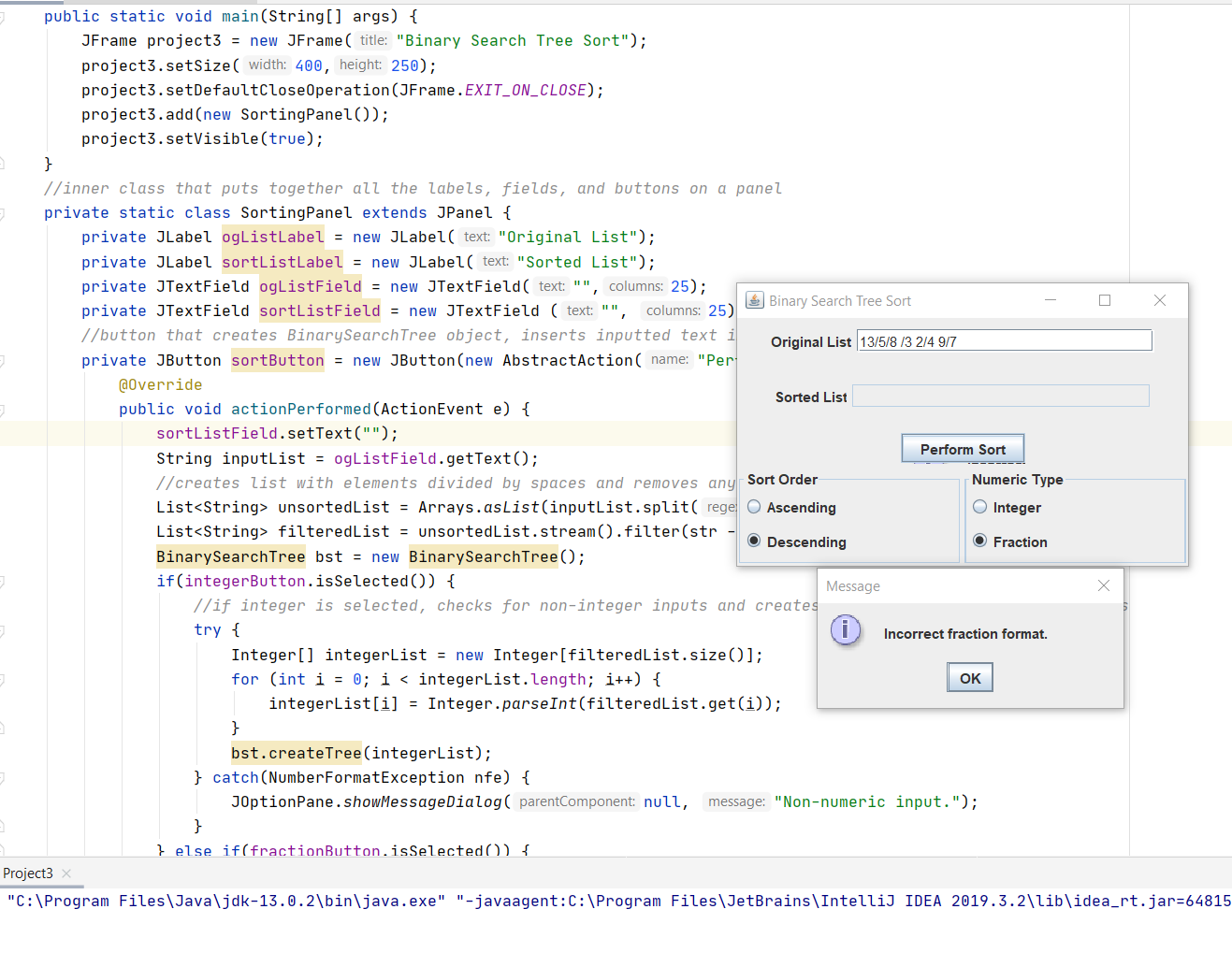
### Test Case 5

This test case inputted an unsorted list of fractions (with duplicates) and requested a descending sort. I inputted “23/57 3/4 9/2 13/19”. The resulting sorted list was “9/2 3/4 13/19 23/57”. This test case passes.



### Test Case 6

This test case inputted an unsorted list of fractions (with improperly formatted fractions) and requested a descending sort. I inputted “13/5/8 /3 2/4 9/7” and an error pop-up was displayed as expected. This test case passes.



### Lessons Learned

I found this project a little more straight-forward than the previous ones so I had less issue with the algorithm behind the sorting and more with the error catching and parsing of the inputs.

I decided to change up my strategy for parsing this time and used lists instead of arrays as I wanted to try using the stream and filter methods to eliminate any possibilities of empty strings or strings with only whitespace getting into the input array. In the previous project, I had worked around this by creating an empty if statement that handled these types of strings, but this time I used the filter method to only select strings that did not contain whitespace.

I also improved my integer checking method for strings by catching the expected NumberFormatException when my parseInt() method failed rather than checking the first character if it was an integer like my previous projects. This method allowed me to create more customized error messages based on where the error was thrown in the project.

Finally, I had to learn some more about regex in order to create a regex expression to catch improperly formatted fractions. I wanted to be able to accept any type of fraction, even fractions with multiple digit divisor and dividends so I ended up writing a regex expression that capture any number of digits before and after a division operator.