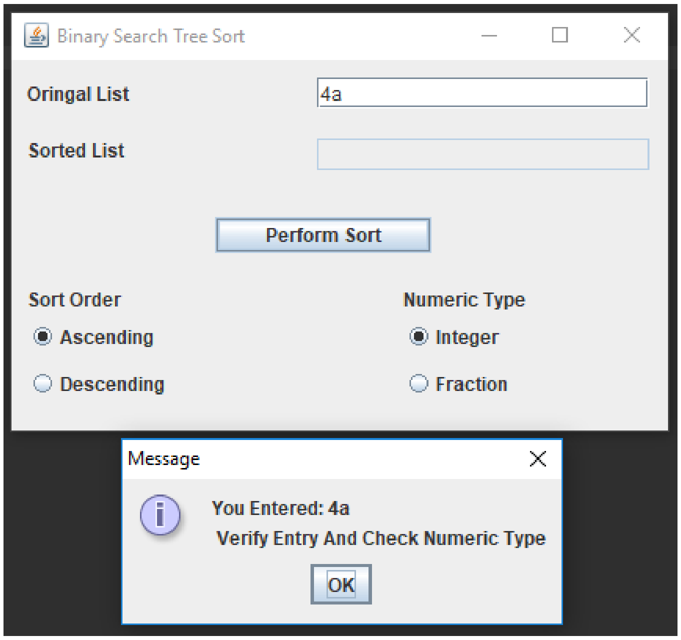
Kresimir Tokic
CMSC350 Project 3
Test Plan, UML diagram, Lessons Learned

TEST PLAN

Test Case 1: Testing user input invalid characters for integers ascending & descending

1. Input “4a”
2. Select appropriate sort order and numeric type click “Perform Sort” button
3. Verify Format Exception window pops up



Oringal List: 4a

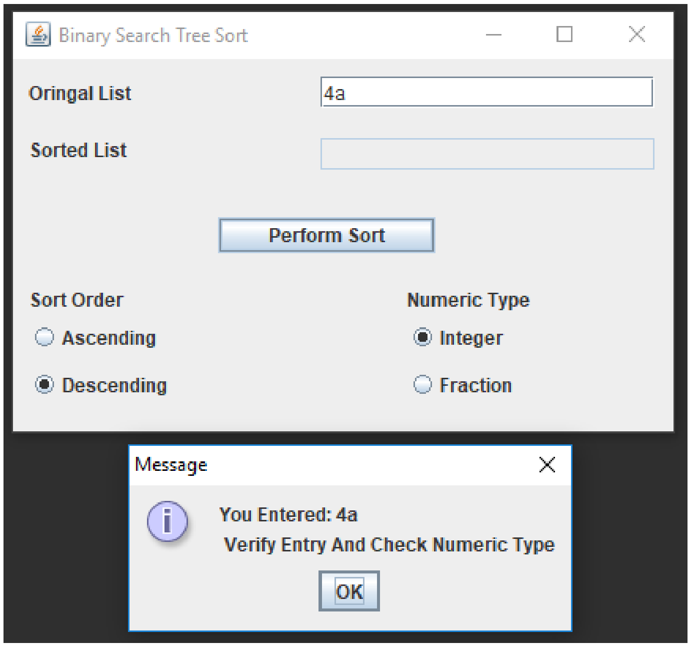
Sorted List:

Sort Order: ● Ascending ○ Descending

Numeric Type: ● Integer ○ Fraction

Message

You Entered: 4a
Verify Entry And Check Numeric Type



Oringal List: 4a

Sorted List:

Sort Order: ○ Ascending ● Descending

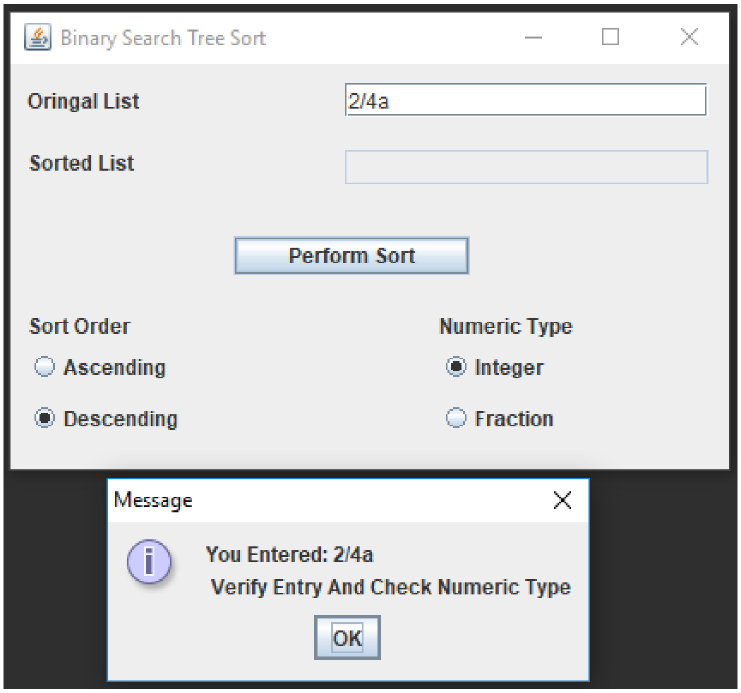
Numeric Type: ● Integer ○ Fraction

Message

You Entered: 4a
Verify Entry And Check Numeric Type

Test Case 2: Testing user input invalid characters for integers ascending & descending

1. Input "2/4a"
2. Select appropriate sort order and numeric type click "Perform Sort" button
3. Verify Format Exception window pops up



Oringal List: 2/4a

Sorted List:

Sort Order

○ Ascending

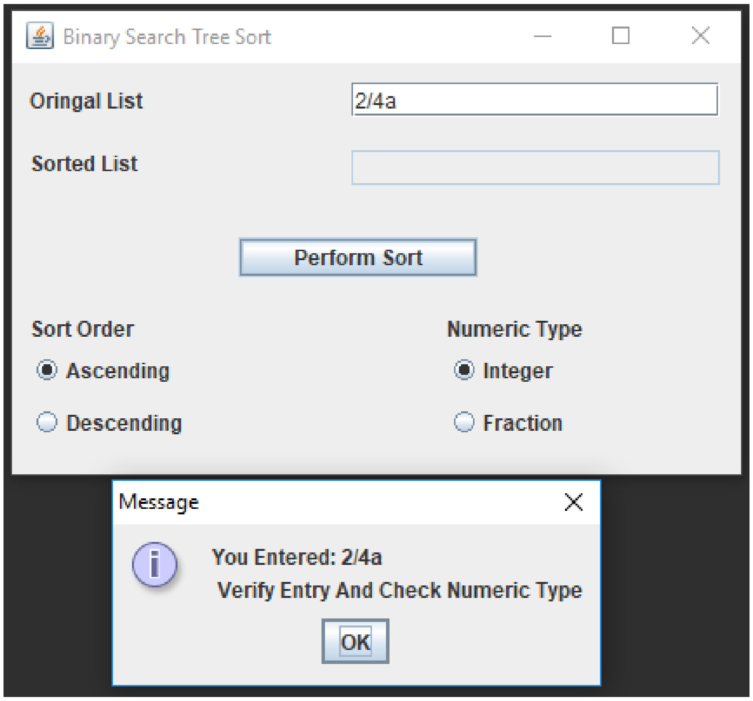
● Descending

Numeric Type

● Integer

○ Fraction

You Entered: 2/4a
Verify Entry And Check Numeric Type



Oringal List: 2/4a

Sorted List:

Sort Order

● Ascending

○ Descending

Numeric Type

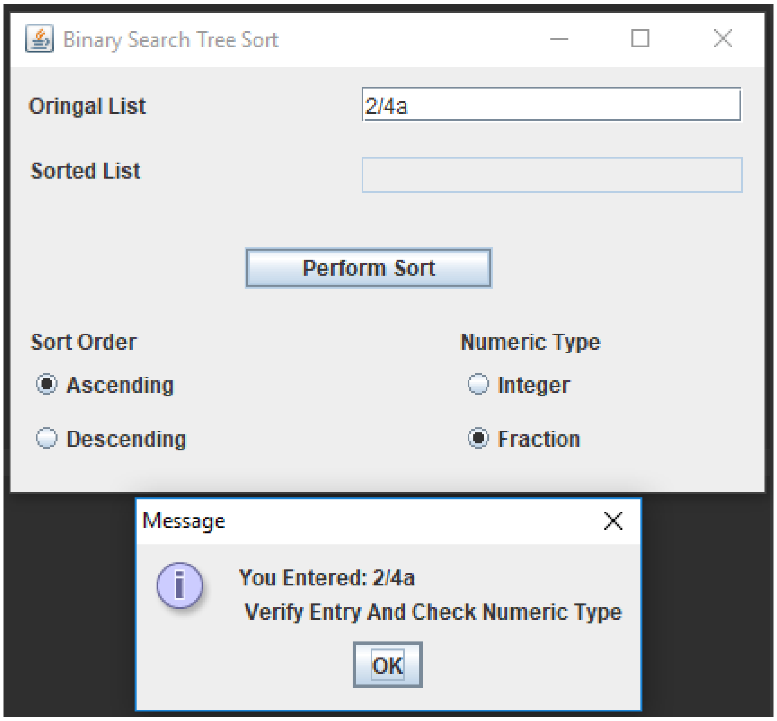
● Integer

○ Fraction

You Entered: 2/4a
Verify Entry And Check Numeric Type

Test Case 3: Testing user input invalid characters for fractions ascending & descending

1. Input “2/4a”
2. Select appropriate sort order and numeric type click “Perform Sort” button
3. Verify Format Exception window pops up



Oringal List: 2/4a

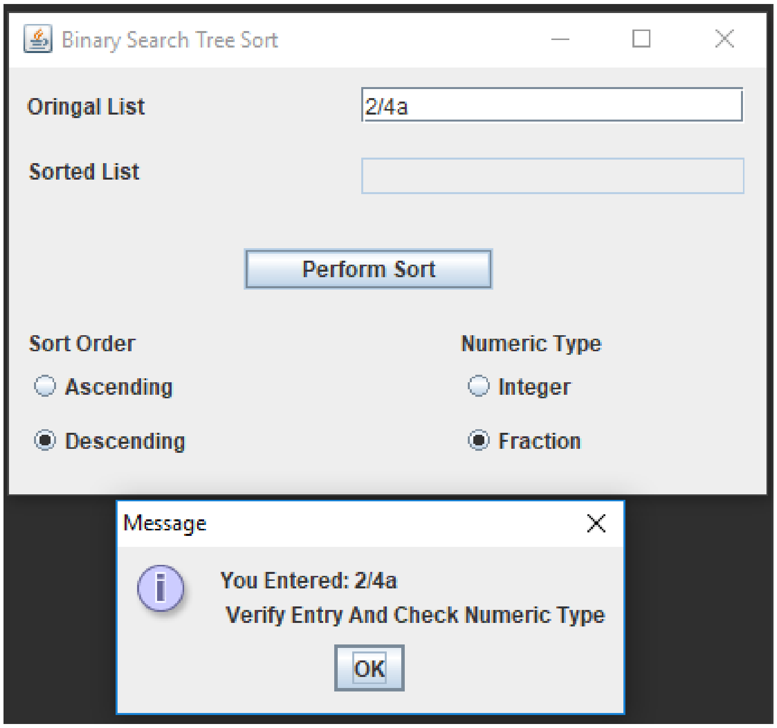
Sorted List:

Sort Order
● Ascending
○ Descending

Numeric Type
○ Integer
● Fraction

Message

You Entered: 2/4a
Verify Entry And Check Numeric Type



Oringal List: 2/4a

Sorted List:

Sort Order
○ Ascending
● Descending

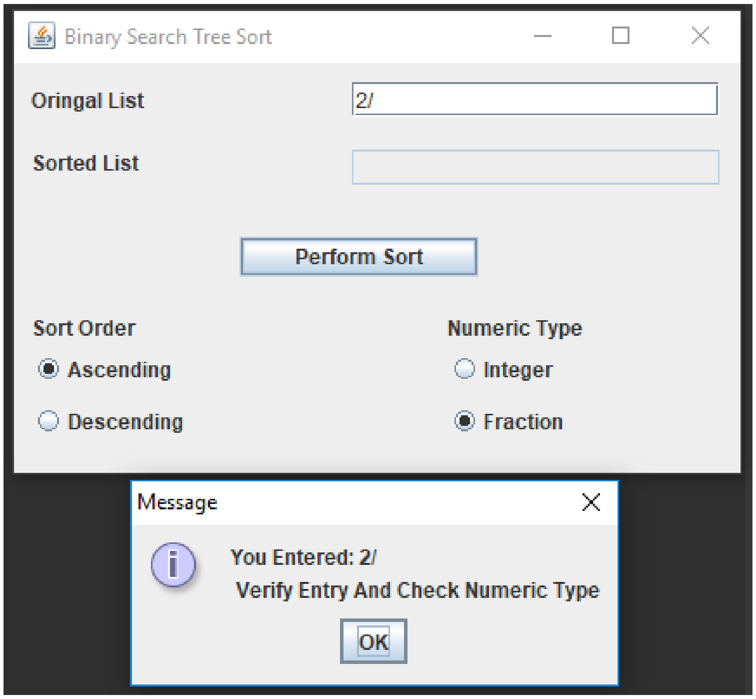
Numeric Type
○ Integer
● Fraction

Message

You Entered: 2/4a
Verify Entry And Check Numeric Type

Test Case 4: Testing user input invalid format for fractions ascending & descending

1. Input “2/”
2. Select appropriate sort order and numeric type click “Perform Sort” button
3. Verify Format Exception window pops up



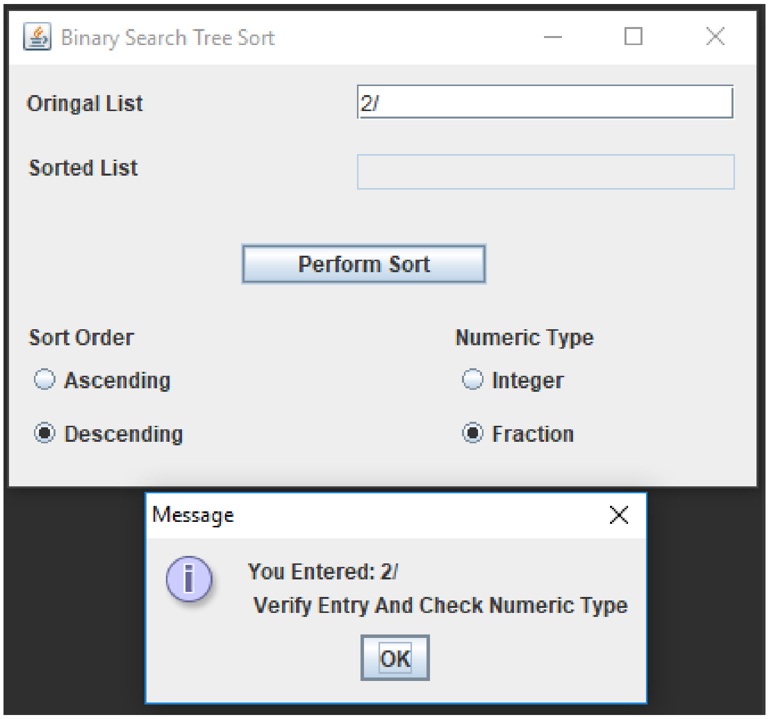
Oringal List: 2/

Sorted List:

Sort Order
● Ascending
○ Descending

Numeric Type
○ Integer
● Fraction

You Entered: 2/
Verify Entry And Check Numeric Type



Oringal List: 2/

Sorted List:

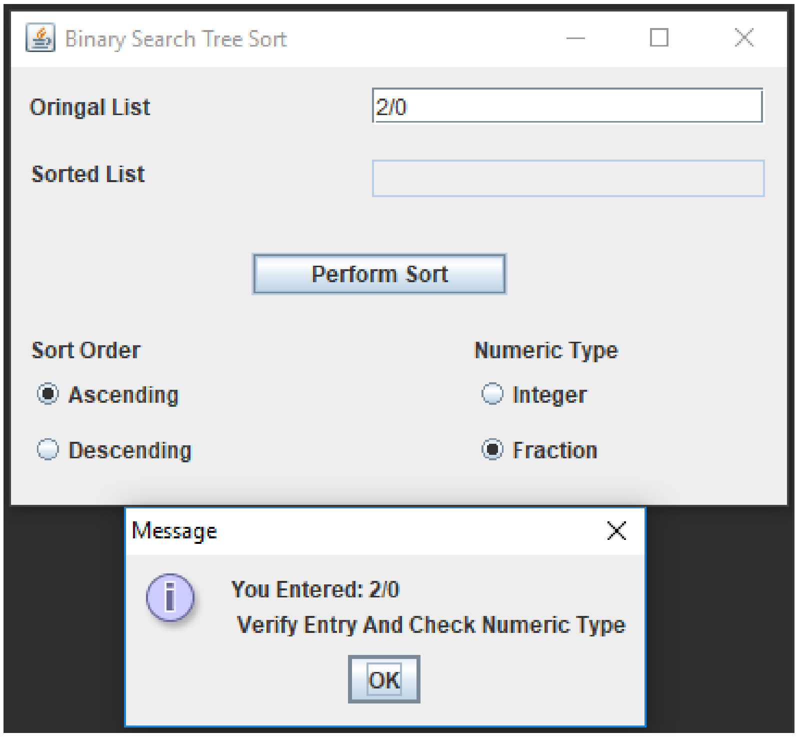
Sort Order
○ Ascending
● Descending

Numeric Type
○ Integer
● Fraction

You Entered: 2/
Verify Entry And Check Numeric Type

Test Case 5: Testing user input divide by 0 for fractions ascending & descending

1. Input “2/0”
2. Select appropriate sort order and numeric type click “Perform Sort” button
3. Verify Format Exception window pops up



Oringal List 2/0

Sorted List

Sort Order

● Ascending

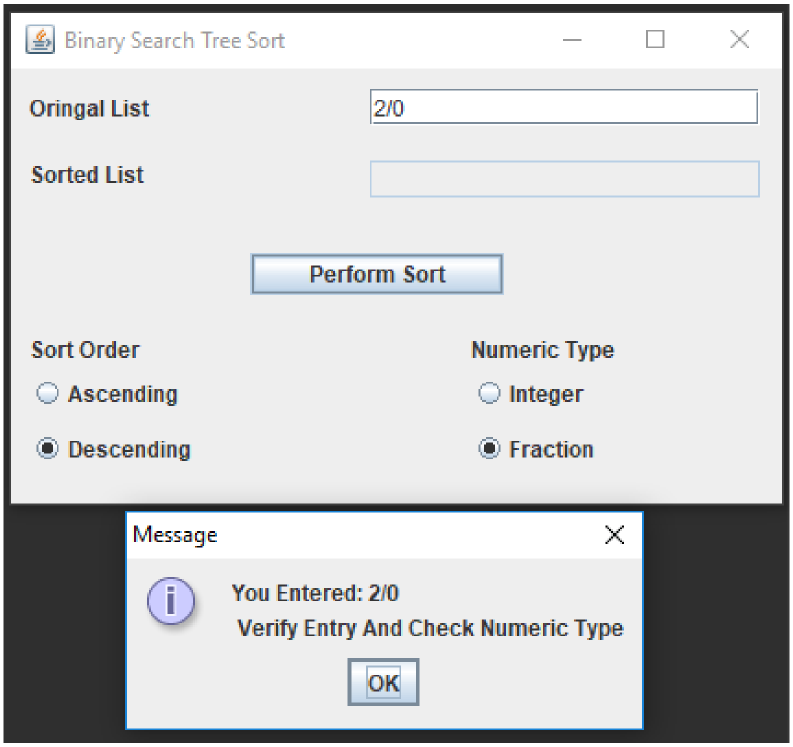
○ Descending

Numeric Type

○ Integer

● Fraction

You Entered: 2/0
Verify Entry And Check Numeric Type



Oringal List 2/0

Sorted List

Sort Order

○ Ascending

● Descending

Numeric Type

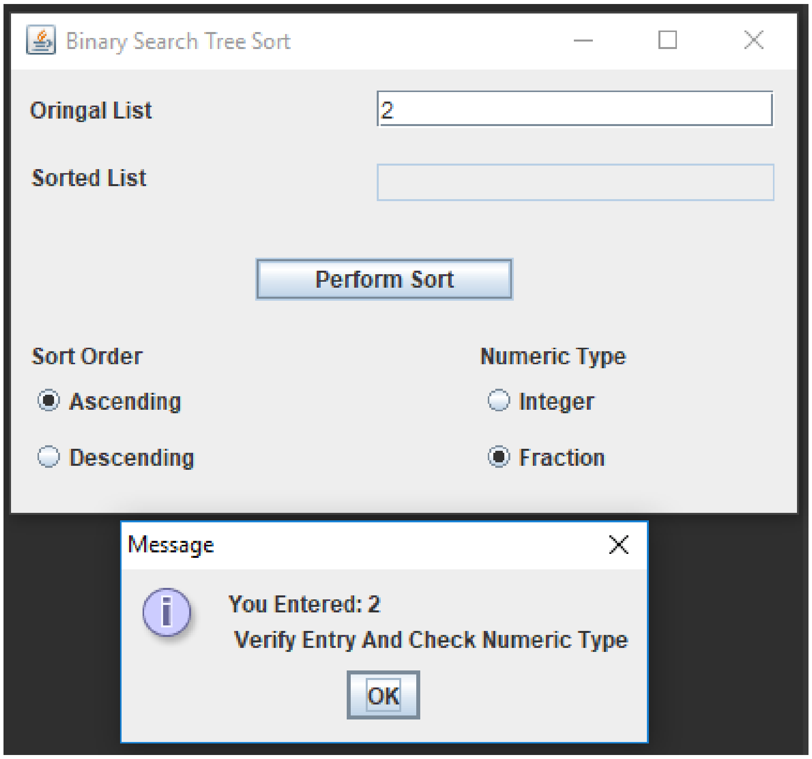
○ Integer

● Fraction

You Entered: 2/0
Verify Entry And Check Numeric Type

Test Case 6: Testing user input invalid format for fractions ascending & descending

1. Input “2”
2. Select appropriate sort order and numeric type click “Perform Sort” button
3. Verify Format Exception window pops up



Oringal List: 2

Sorted List:

Sort Order

● Ascending

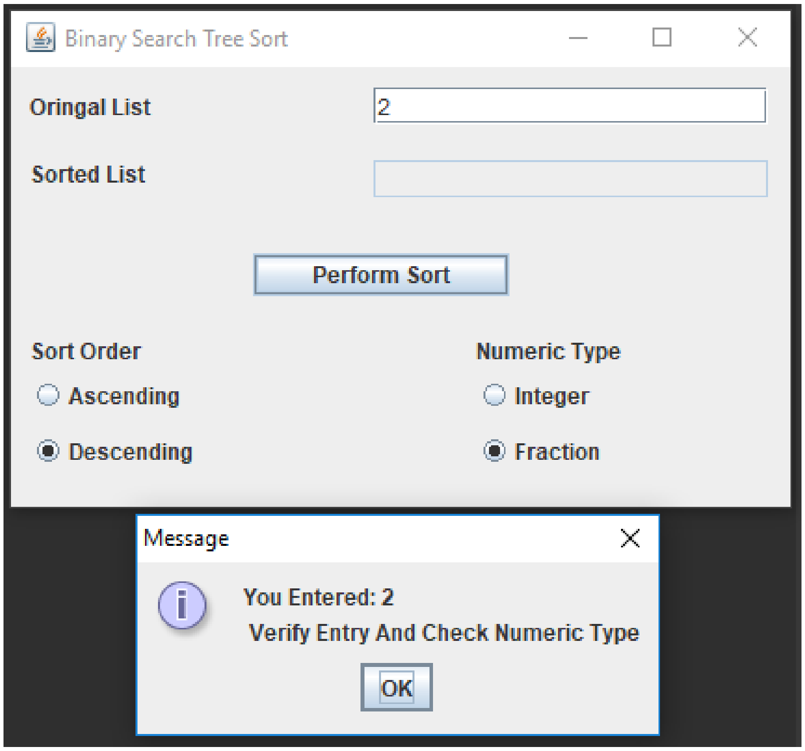
○ Descending

Numeric Type

○ Integer

● Fraction

You Entered: 2
Verify Entry And Check Numeric Type



Oringal List: 2

Sorted List:

Sort Order

○ Ascending

● Descending

Numeric Type

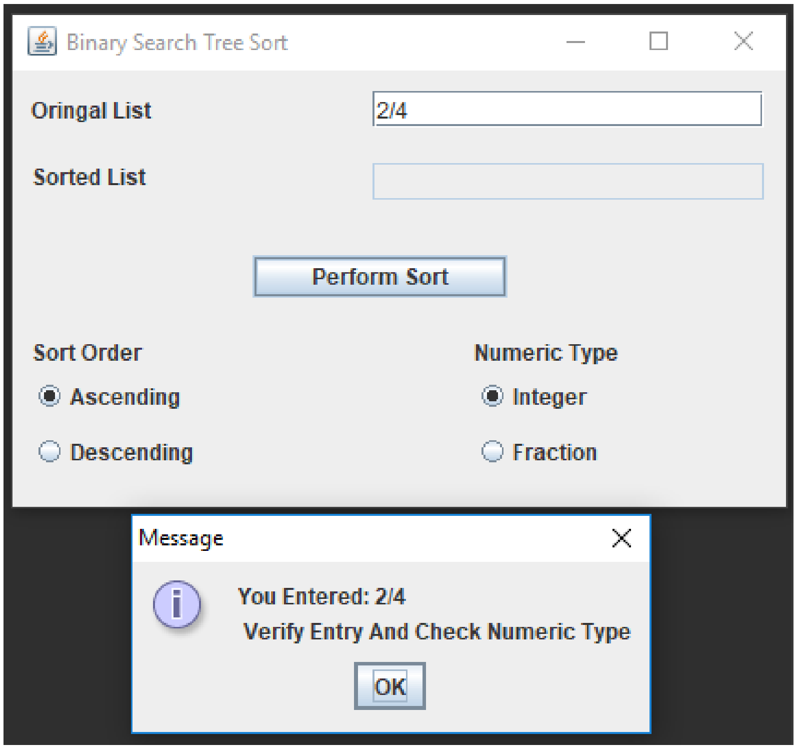
○ Integer

● Fraction

You Entered: 2
Verify Entry And Check Numeric Type

Test Case 7: Testing user input invalid format for integers ascending & descending

1. Input "2/4"
2. Select appropriate sort order and numeric type click "Perform Sort" button
3. Verify Format Exception window pops up



Oringal List: 2/4

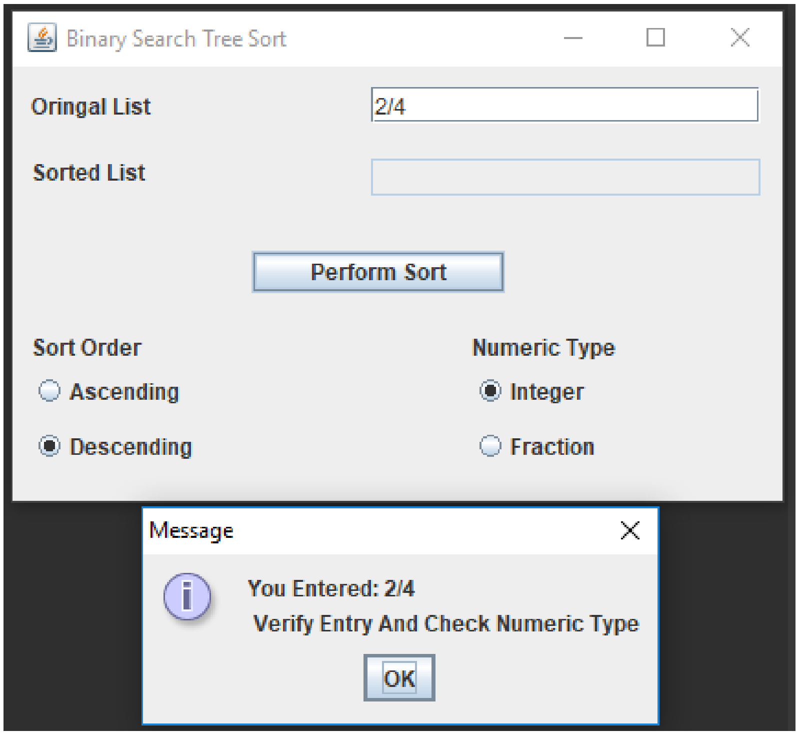
Sorted List:

Sort Order
● Ascending
○ Descending

Numeric Type
● Integer
○ Fraction

Message

You Entered: 2/4
Verify Entry And Check Numeric Type



Oringal List: 2/4

Sorted List:

Sort Order
○ Ascending
● Descending

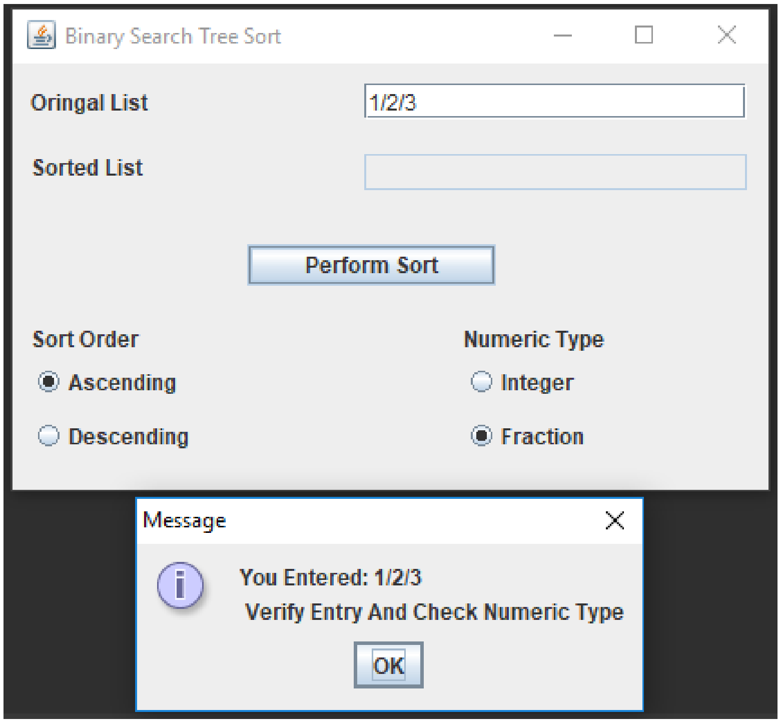
Numeric Type
● Integer
○ Fraction

Message

You Entered: 2/4
Verify Entry And Check Numeric Type

Test Case 8: Testing user input invalid format for fractions ascending & descending

1. Input “1/2/3”
2. Select appropriate sort order and numeric type click “Perform Sort” button
3. Verify Format Exception window pops up



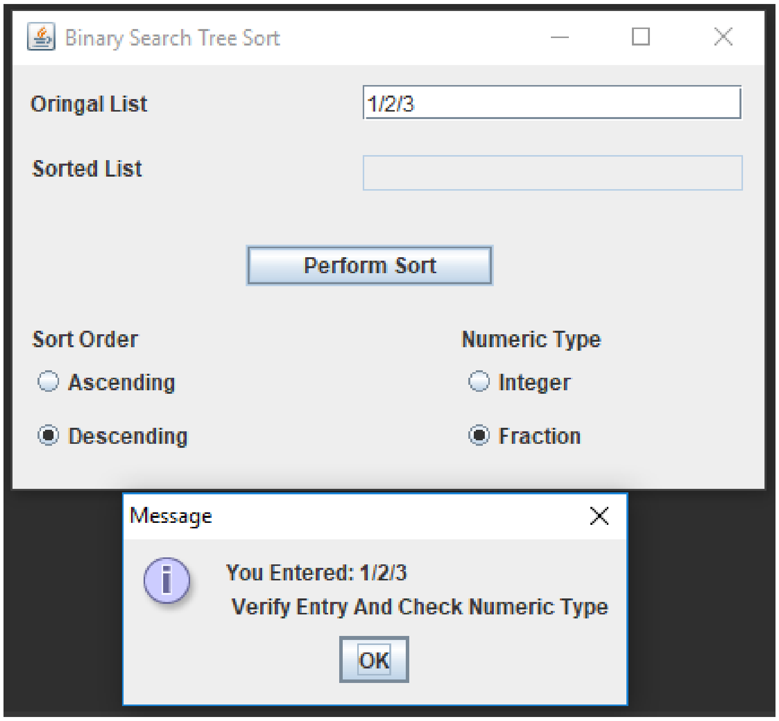
Oringal List: 1/2/3

Sorted List:

Sort Order: ● Ascending ○ Descending

Numeric Type: ○ Integer ● Fraction

You Entered: 1/2/3
Verify Entry And Check Numeric Type



Oringal List: 1/2/3

Sorted List:

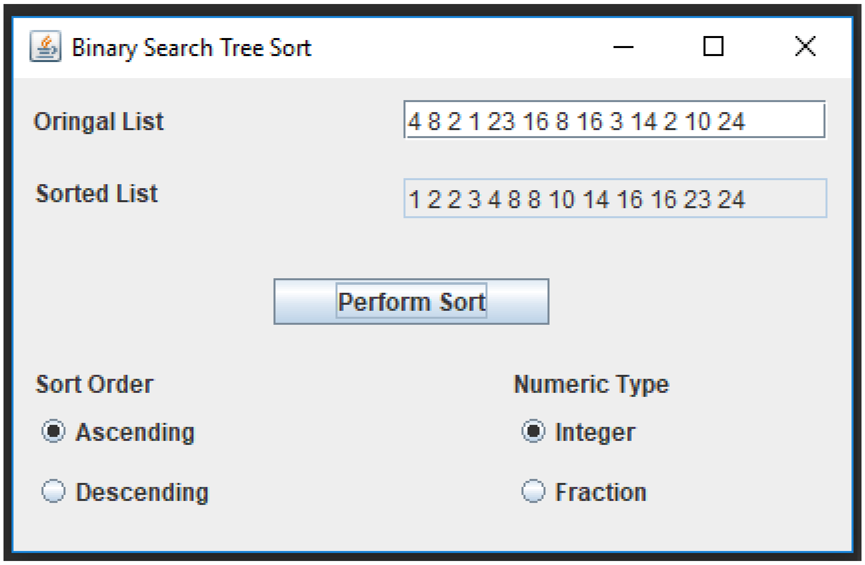
Sort Order: ○ Ascending ● Descending

Numeric Type: ○ Integer ● Fraction

You Entered: 1/2/3
Verify Entry And Check Numeric Type

Test Case 9: Testing sorting of integers both ascending and descending

1. Input "4 8 2 1 23 16 8 16 3 14 2 10 24"
2. Select appropriate sort order and numeric type click "Perform Sort" button
3. Verify Format Exception window pops up

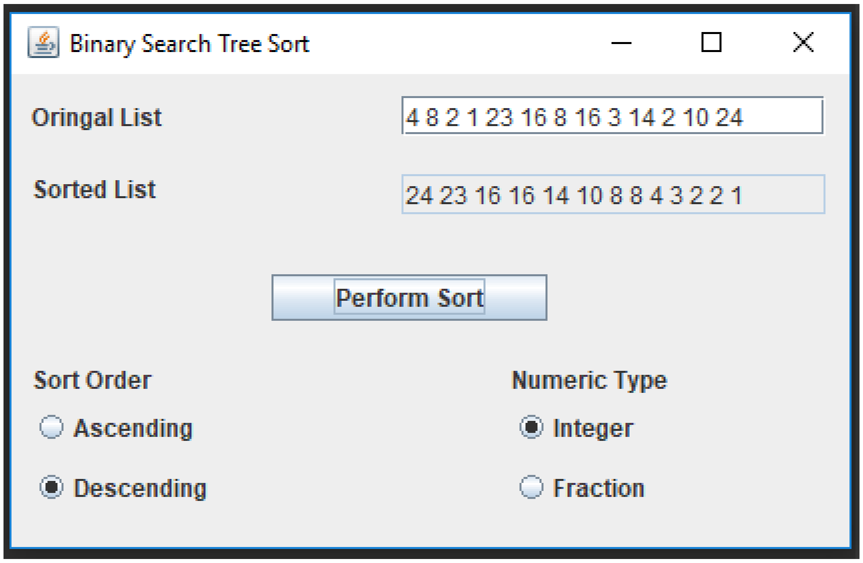

Binary Search Tree Sort

Oringal List: 4 8 2 1 23 16 8 16 3 14 2 10 24

Sorted List: 1 2 2 3 4 8 8 10 14 16 16 23 24

Sort Order
● Ascending
○ Descending

Numeric Type
● Integer
○ Fraction


Binary Search Tree Sort

Oringal List: 4 8 2 1 23 16 8 16 3 14 2 10 24

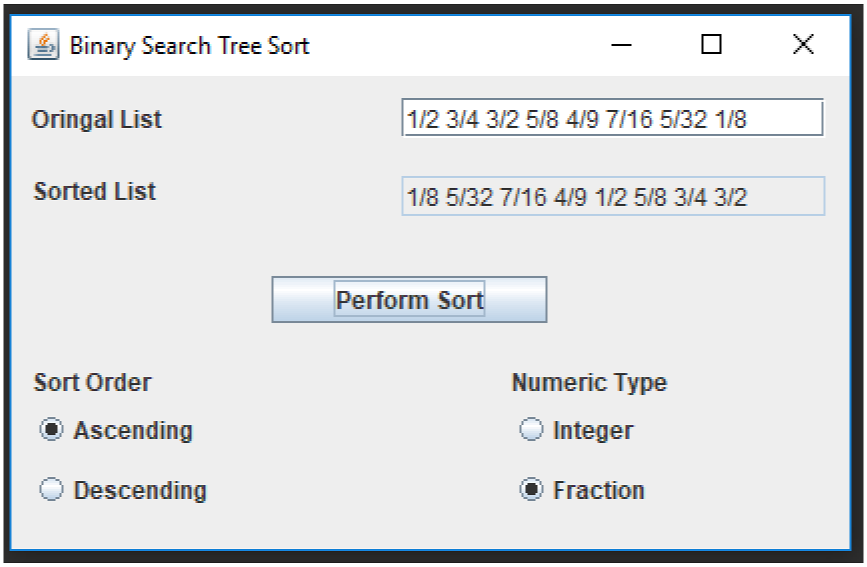
Sorted List: 24 23 16 16 14 10 8 8 4 3 2 2 1

Sort Order
○ Ascending
● Descending

Numeric Type
● Integer
○ Fraction

Test Case 10: Testing sorting of fractions both ascending and descending

1. Input "1/2 3/4 3/2 5/8 4/9 7/16 5/32 1/8"
2. Select appropriate sort order and numeric type click "Perform Sort" button
3. Verify Format Exception window pops up



Binary Search Tree Sort

Oringal List: 1/2 3/4 3/2 5/8 4/9 7/16 5/32 1/8

Sorted List: 1/8 5/32 7/16 4/9 1/2 5/8 3/4 3/2

Sort Order

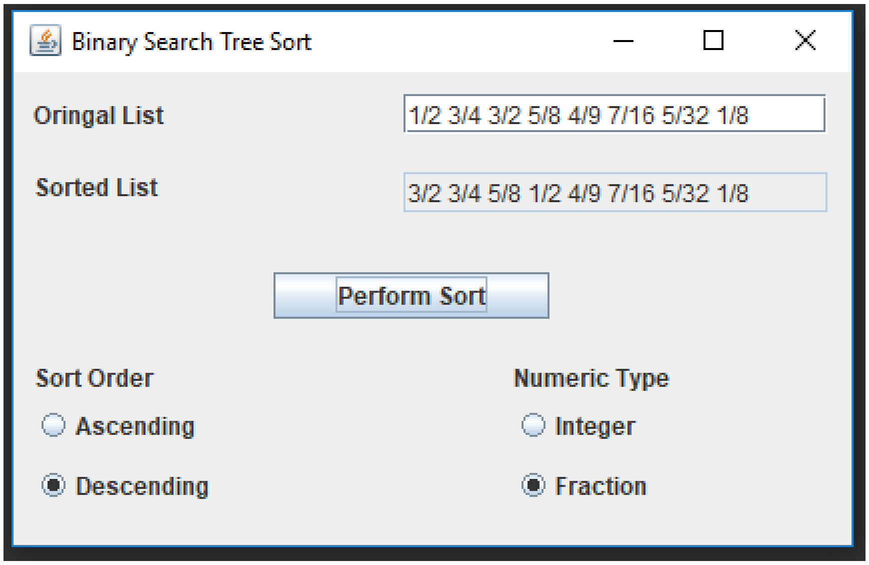
● Ascending

○ Descending

Numeric Type

○ Integer

● Fraction



Binary Search Tree Sort

Oringal List: 1/2 3/4 3/2 5/8 4/9 7/16 5/32 1/8

Sorted List: 3/2 3/4 5/8 1/2 4/9 7/16 5/32 1/8

Sort Order

○ Ascending

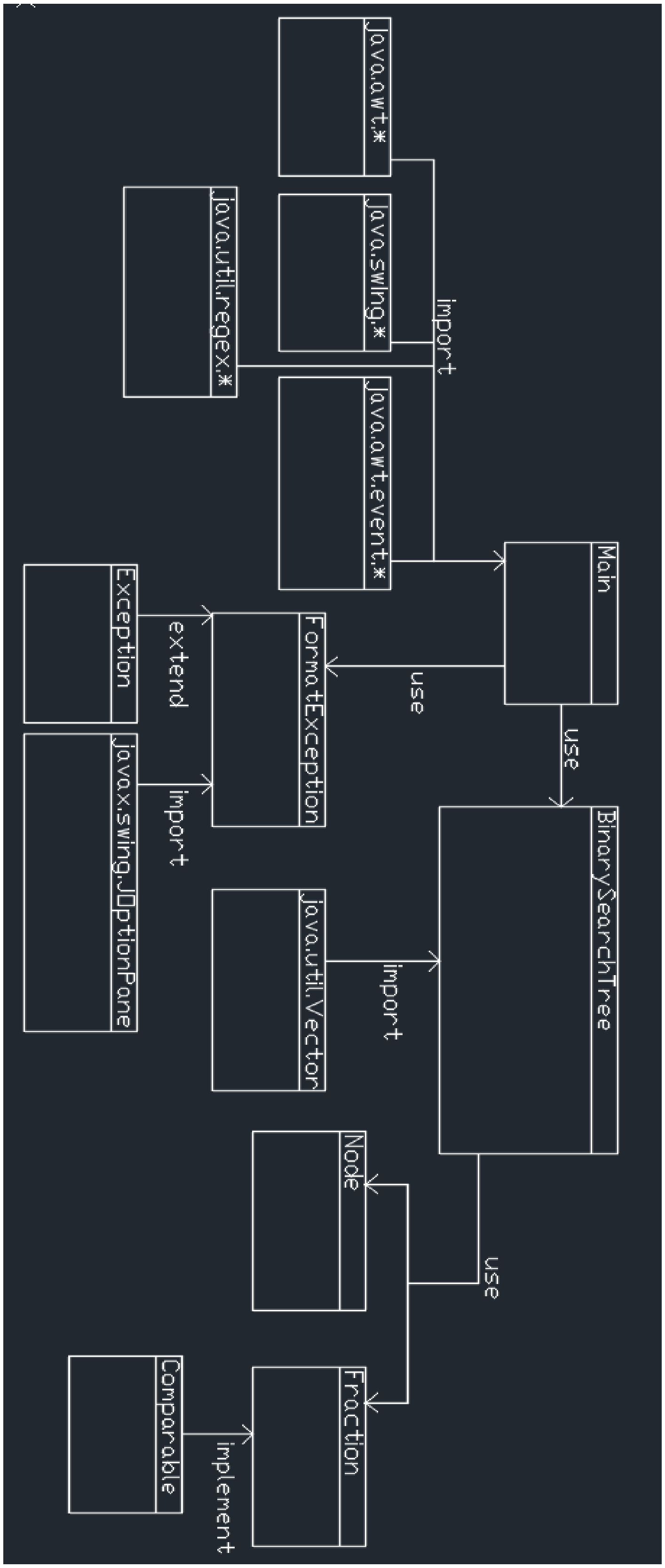
● Descending

Numeric Type

○ Integer

● Fraction

UML Diagram


java.awt.*
java.swing.*
java.util.regex.*
java.awt.event.*
import
Main
Exception
extend
FormatException
use
javax.swing.J□ptionPane
import
use
BinarySearchTree
java.util.Vector
import
Node
use
Comparable
implement
Fraction

Lessons Learned

I usually just concatenate strings but after some digging around on Stack Overflow I found the following statement “At the point where you're concatenating in a loop - that's usually when the compiler can't substitute StringBuilder by itself.” I originally thought I could reuse a lot more code from project 2 than I realized was going to be possible. After a bit of digging through Geeks for Geeks and Stack Overflow I found some similar problems to the ones I was having and figured out how to carry the user inputted strings through my nodes while using a doubles for the comparisons. With implementing the Comparable interface I had to do a little research again. I browsed Oracle’s site to read and learn about the Comparable interface, eventually I was able to implement the required compareTo method. Upon futher digging and discussion in the student lounge I found alternate means to implement the method as well. I choose to catch all of exceptions in the main class, thereby prohibiting the program from running through my other classes. Checking the fractions took a bit work but I eventually figured out how to use regex and patterns and matchers to compare the fracture structuring. I also opted to use a single exception for all the user input errors I check for. After some discussion about linked lists and web browsers I decided I’d try to leverage vectors for this assignment. I don’t know that they’re necessary for this project, but I wanted to try them out. This and some additional if else statements allowed me to reduce the total number of methods used in my Binary Search Tree class.