- Lab 12 is to start working on the GUI for programming assignment 8
  - Or working on the functionality if the GUI is already complete
- Programming assignment 8 is the final project
- For lab 12, you want to get started on the GUI
  - You don't need to finish it, but you want to make significant progress on it
  - The lab is not until next week
- The final project is due at midnight, on the last day of classes, 12/9/2022
  - But I will accept submissions, for full credit, until midnight, on 12/16/2022

- For program 8 we will be reading two files of integer values
  - The first file is the “sort” file
  - The second file is the “search” file
- When the files are read, the values are added to two int arrays
  - In my code, I read the values into an ArrayList<Integer>, and once the entire file was read, my code copies the values into the appropriate int[]

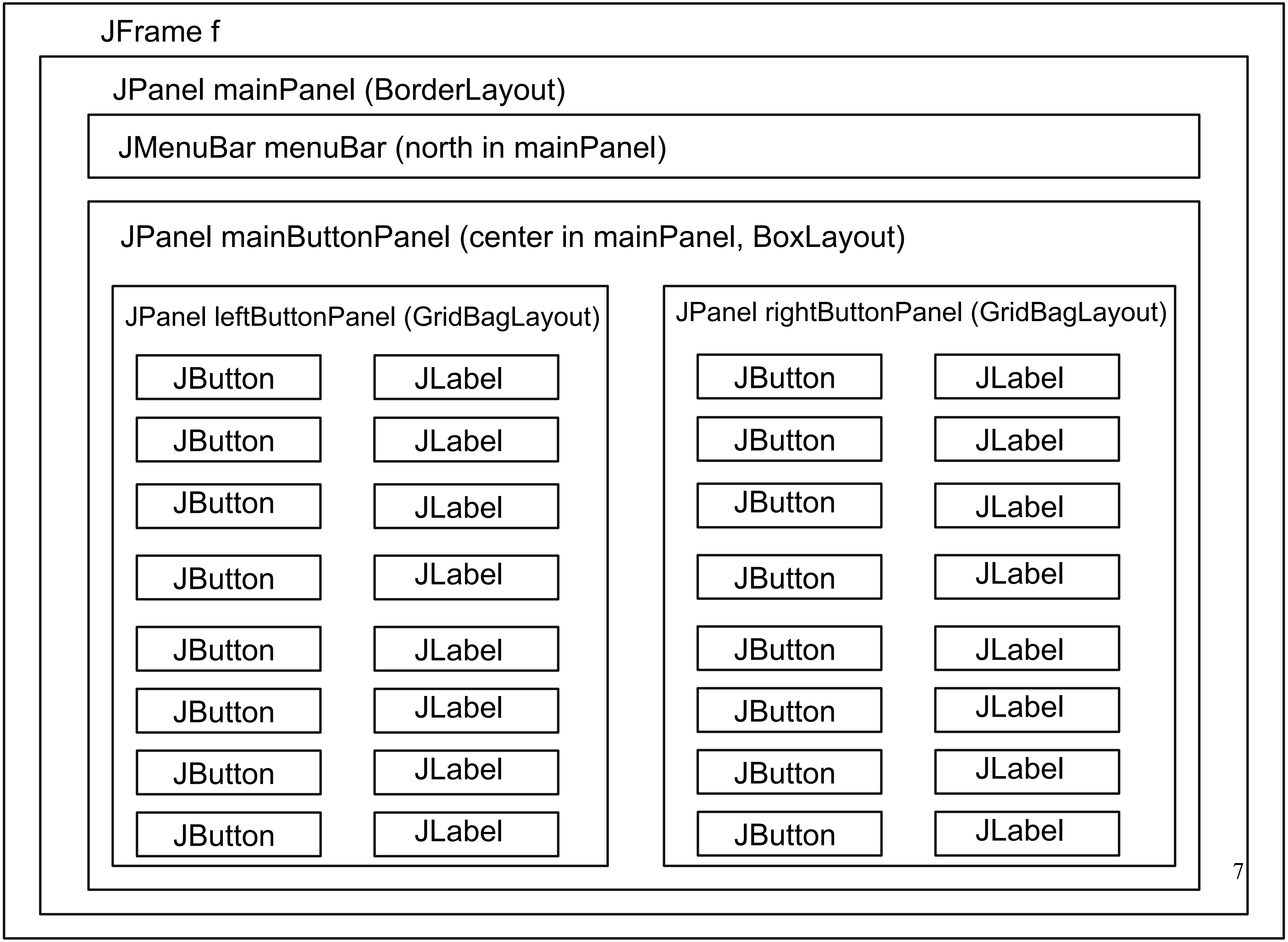
- For programming assignment 1 we implemented selection sort to sort a String[]
- We will be re-using that method, except we will be updating it to sort an int[]
- This update should be easy
  - Replace String[] with int[]
  - Assuming the String[] is named values
    - Replace “if(values[j].compareTo(values[min]) < 0)” with “if(values[j] < values[min])”
  - That should be pretty much it

- The first part of the remainder of the program is to evaluate the performance of copying the “sort” int[] values into
  - An int[] and sorting it using the updated selection sort that we implemented in program 1 (O(n^2))
  - Add the values as keys into a binary search tree (O(n log n))
  - Add the values to a TreeSet (O(n log n))
  - Add the values to a PriorityQueue (O(n log n))
  - Add the values to a HashSet (O(n))
  - Add the values to an ArrayList (unsorted, O(n))
  - Add the values to an ArrayList and use the java.util.Collections.sort to sort the values (O(n log n))
  - Add the values to an int[] (unsorted, O(n))
- Other than the HashSet, the ArrayList (unsorted), and int[] (unsorted), each of the above data structures will have sorted the data either while adding it or after adding it
- Keep track of the time spend performing each of the above

- The second part of the remainder of the program is to evaluate the performance of searching for the “search” int[] values from
  - The int[] (O(log n) for each search)
    - Implement a binary search
  - The binary search tree (O(log n) for each search)
    - This is fast, use the getNode() method
  - The TreeSet (O(log n) for each search)
    - This is fast, use the contains() method
  - The PriorityQueue (O(n) for each search)
    - This is slow, even though the PriorityQueue uses a heap to store the values, and polling is quick, using the contains() method is very slow (and documented as such, I was surprised how slow this was)
  - The HashSet (O(log n) for each search, this might be O(1))
    - This is fast, use the contains() method
  - The ArrayList (O(n) for each search)
    - This is slow, use the contains() method
  - The ArrayList (O(log n) for each search)
    - This is fast, use the java.util.Collections.binarySearch
  - The int[] (O(n) for each search)
    - This is slow, use a for loop to search

- The program has a GUI component and a non-GUI component
  - The GUI component we will discuss momentarily
  - The non-GUI component is the
    - Reading the two files
    - Creating the various data structures
    - Populating the data structures
    - Searching the data structures
    - Tracking the time to do the various tasks
    - Reporting the number of search element found in the various data structures

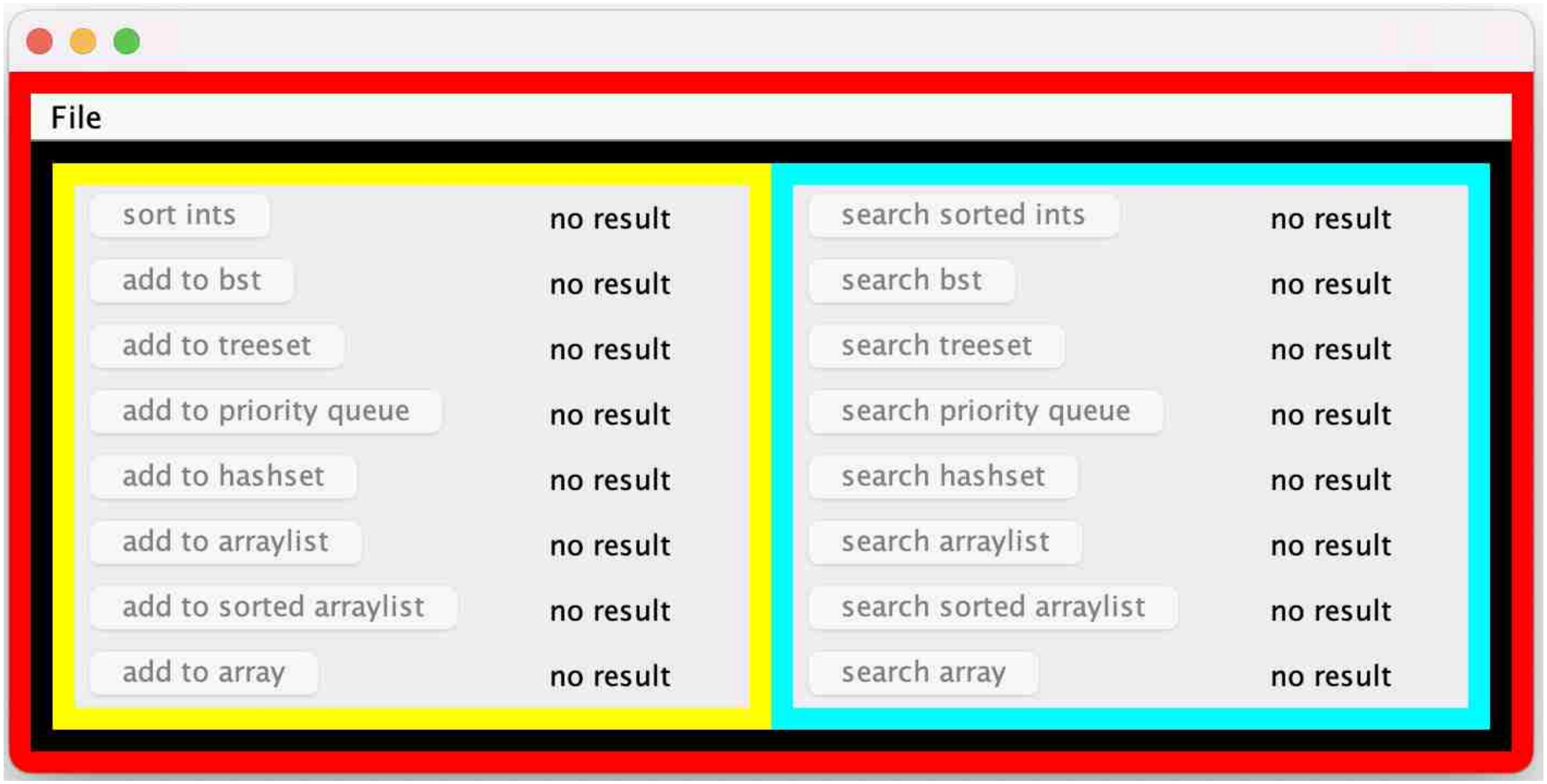
The GUI


JFrame f
JPanel mainPanel (BorderLayout)
JMenuBar menuBar (north in mainPanel)
JPanel mainButtonPanel (center in mainPanel, BoxLayout)
JPanel leftButtonPanel (GridBagLayout)
JButton
JLabel
JPanel rightButtonPanel (GridBagLayout)
JButton
JLabel

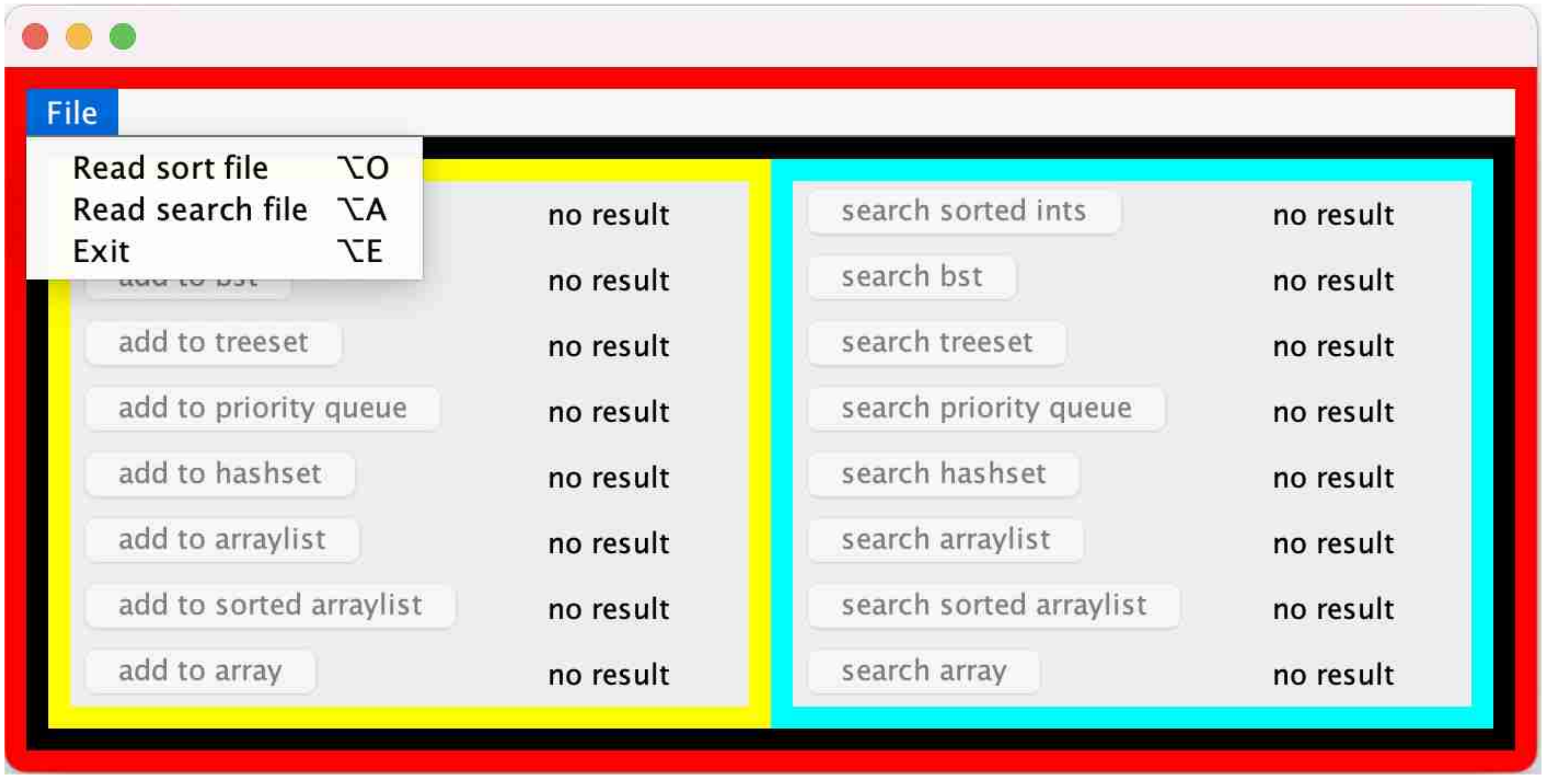
- The majority of the GUI functionality can be found in example2a.java, example4.java, and example4c.java
- Create the JFrame f
- Create the JPanel mainPanel
- Create the JMenuBar
  - Create the JMenu
  - Create the three JMenuItems
  - Add the three JMenuItems to the JMenu
  - Add the JMenu to the JMenuBar
- Create the JPanel mainButtonPanel
- Create the JPanels leftButtonPanel and rightButtonPanel

- Define the LayoutManagers for the four JPanels
- Add the JMenuBar and mainButtonPanel to the mainPanel at NORTH and CENTER
  - Or add the JMenuBar as the menu bar component of the frame (f)
- Add the leftButtonPanel and rightButtonPanel to the mainButtonPanel
- Add the mainPanel to f
- Validate f
- Set f visible
- Ideally you will see something similar to what I have
  - After you have added the JButtons and JLabels

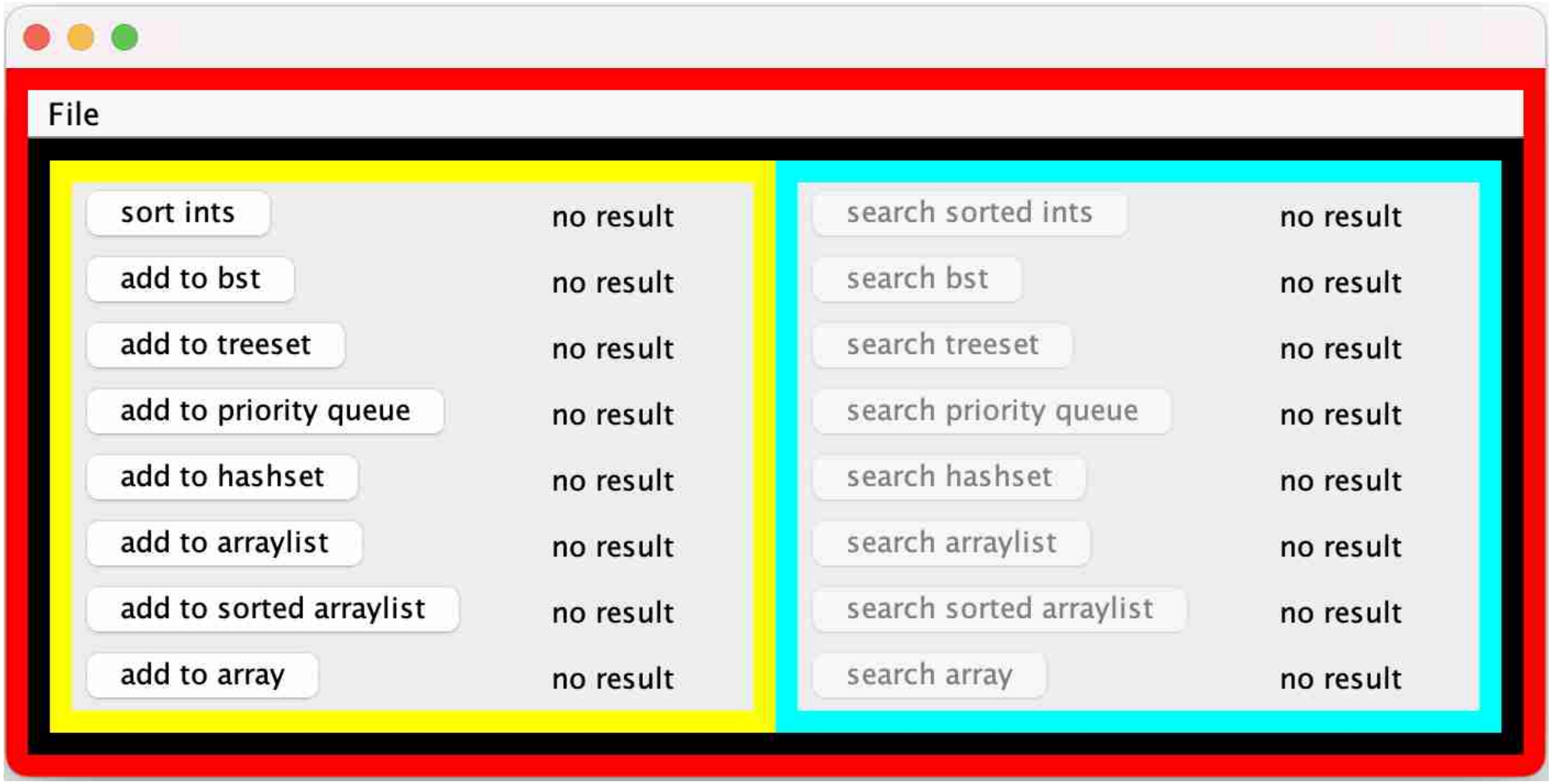
- Here's what mine looks like
  - The red border is the border for mainPanel
    - This won't contain the JMenuBar if you add it to the frame as the menu bar component
  - The black border is the border for mainButtonPanel
  - The yellow and turquoise borders are the borders for leftButtonPanel and rightButtonPanel respectively
  - The borders are there to show the four panels



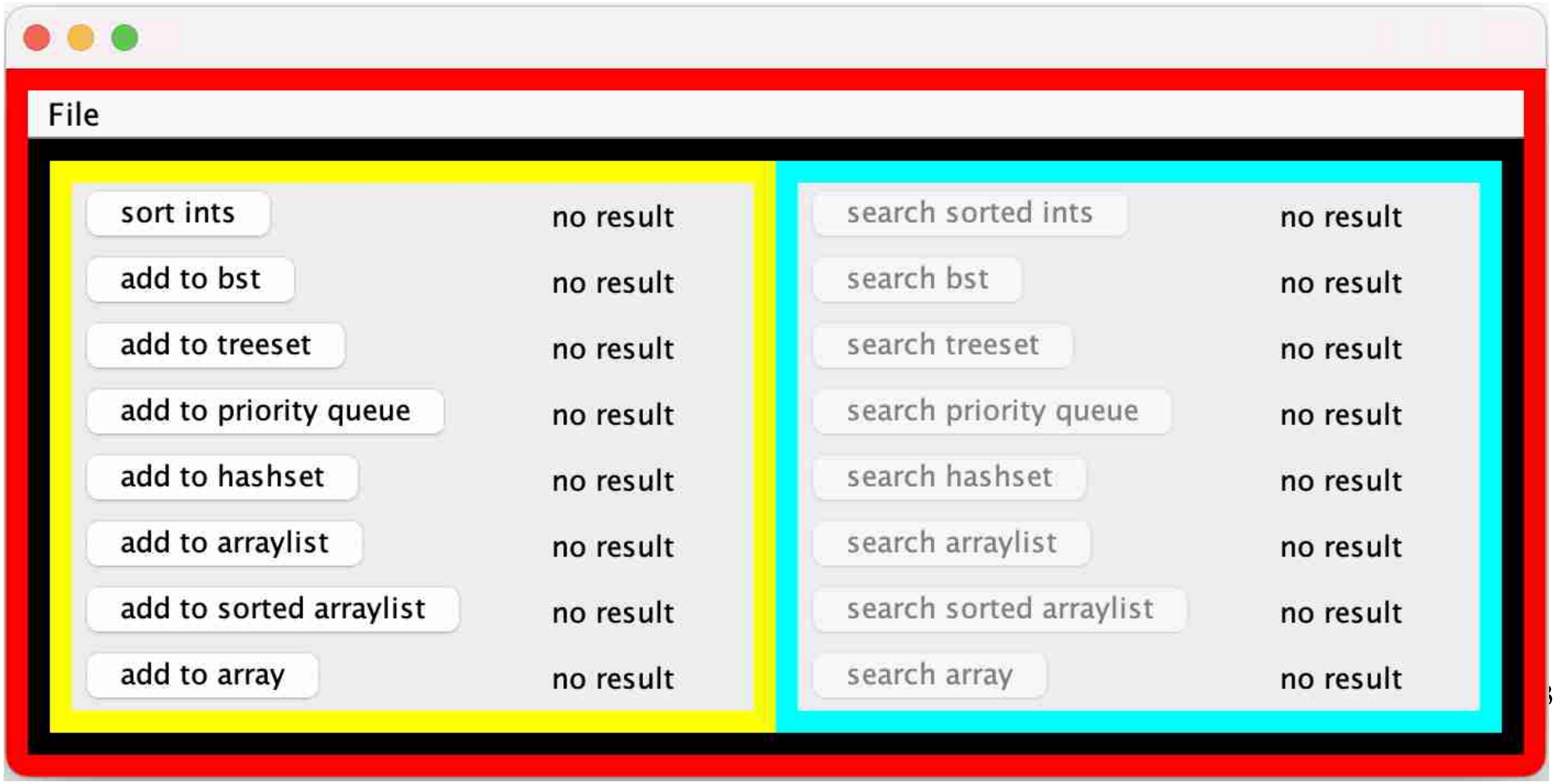
- Here's what mine looks like when the File menu is selected
  - Each of the three JMenuItems will need to have an ActionListener (see example2a.java)
  - The JButtons are not enabled until after the applicable file is read
    - For the search buttons, the search file must be read and the corresponding add/sort button must have already been selected



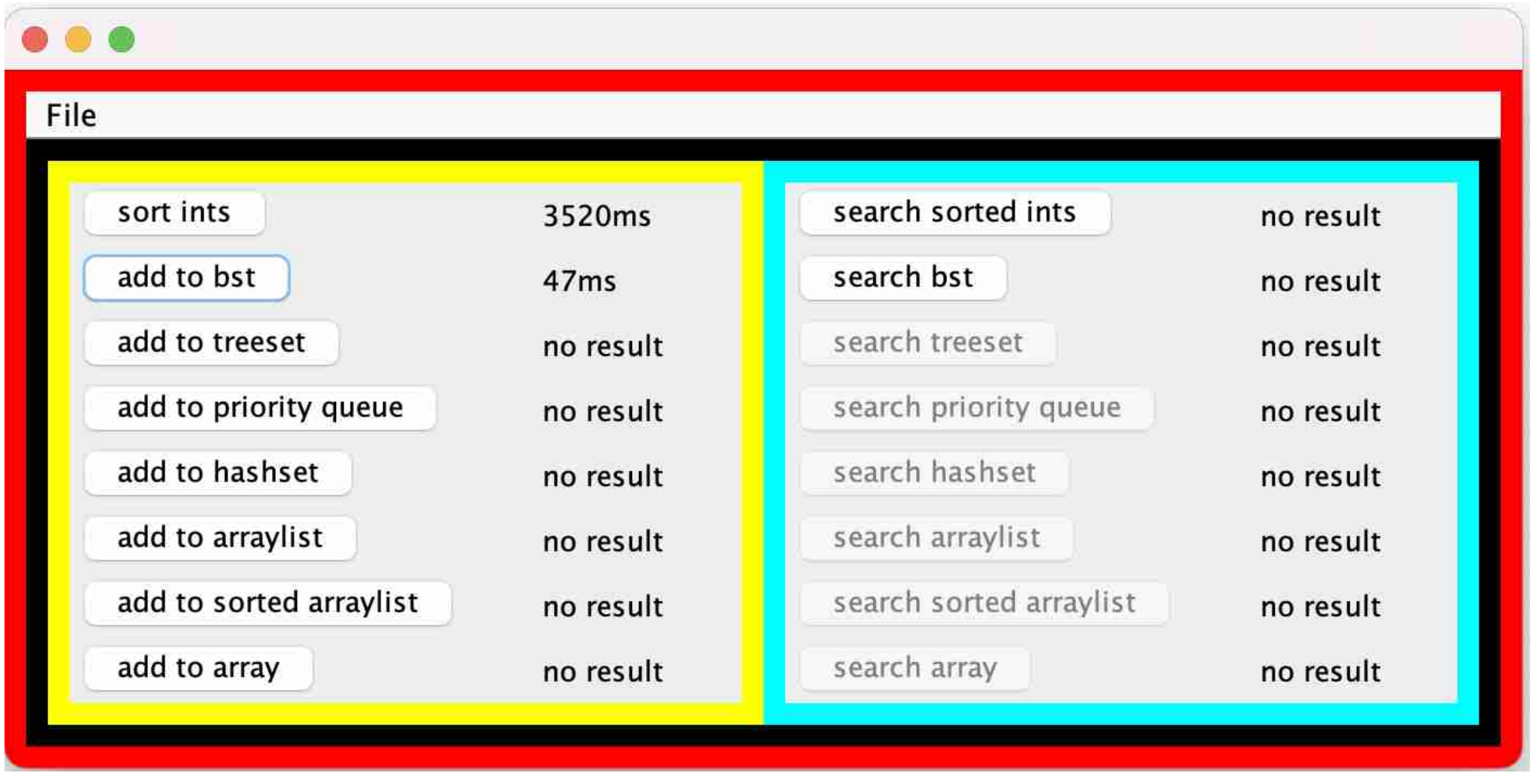
- Here's what mine looks like after the “read sort file” has been selected
  - The “sort”/”add” buttons are now enabled, while the “search” buttons are still not enabled



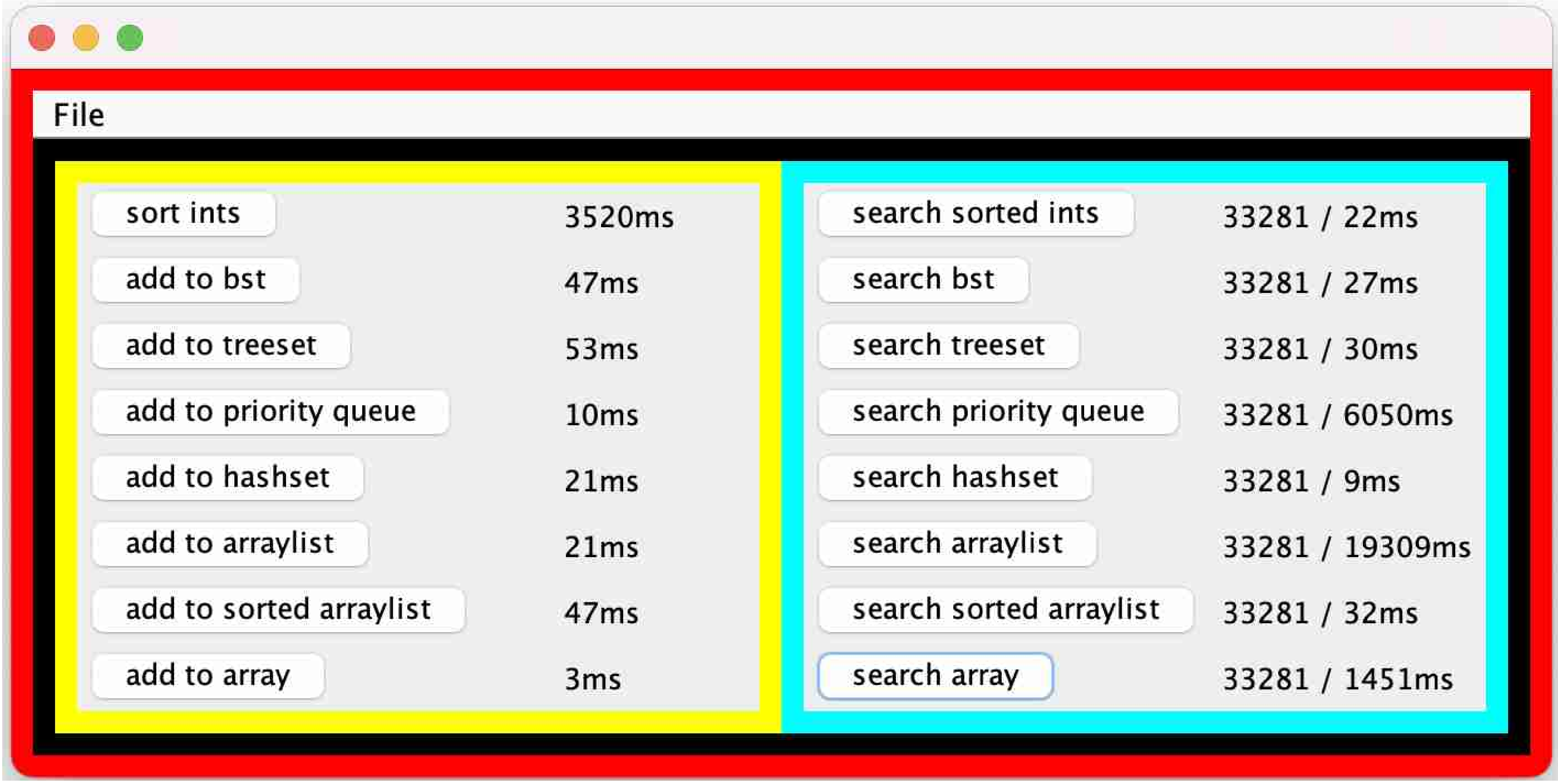
- Here's what mine looks like after the “read sort file” has been selected and the “read search file” has been selected
  - The “search” buttons are not enabled yet, since the corresponding “sort”/”add” buttons have not been selected



- Here's what mine looks like after the two “read ...” menu items have been selected and the first two “sort”/”add” buttons have been selected
  - Note that the first two corresponding “search” buttons are now enabled



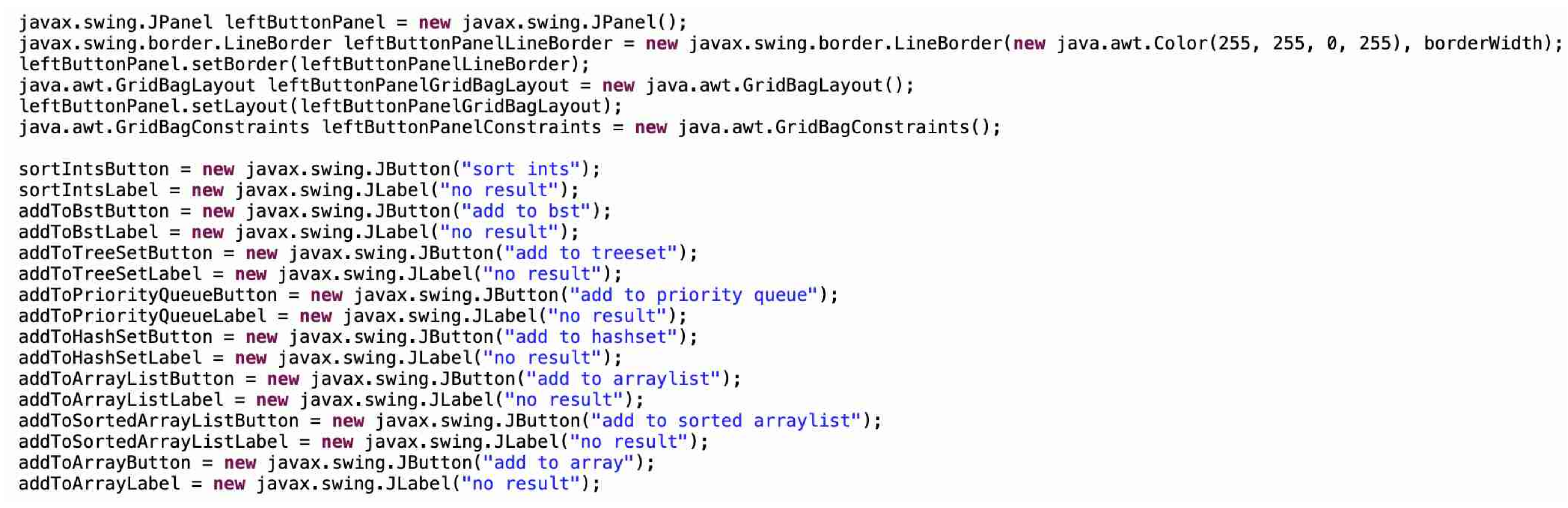
- Here's what mine looks like after the two “read ...” menu items have been selected and the left and right buttons have each been selected
  - This is for the 100,000 sort and search files



sort ints	3520ms	search sorted ints	33281 / 22ms
add to bst	47ms	search bst	33281 / 27ms
add to treeset	53ms	search treeset	33281 / 30ms
add to priority queue	10ms	search priority queue	33281 / 6050ms
add to hashset	21ms	search hashset	33281 / 9ms
add to arraylist	21ms	search arraylist	33281 / 19309ms
add to sorted arraylist	47ms	search sorted arraylist	33281 / 32ms
add to array	3ms	search array	33281 / 1451ms

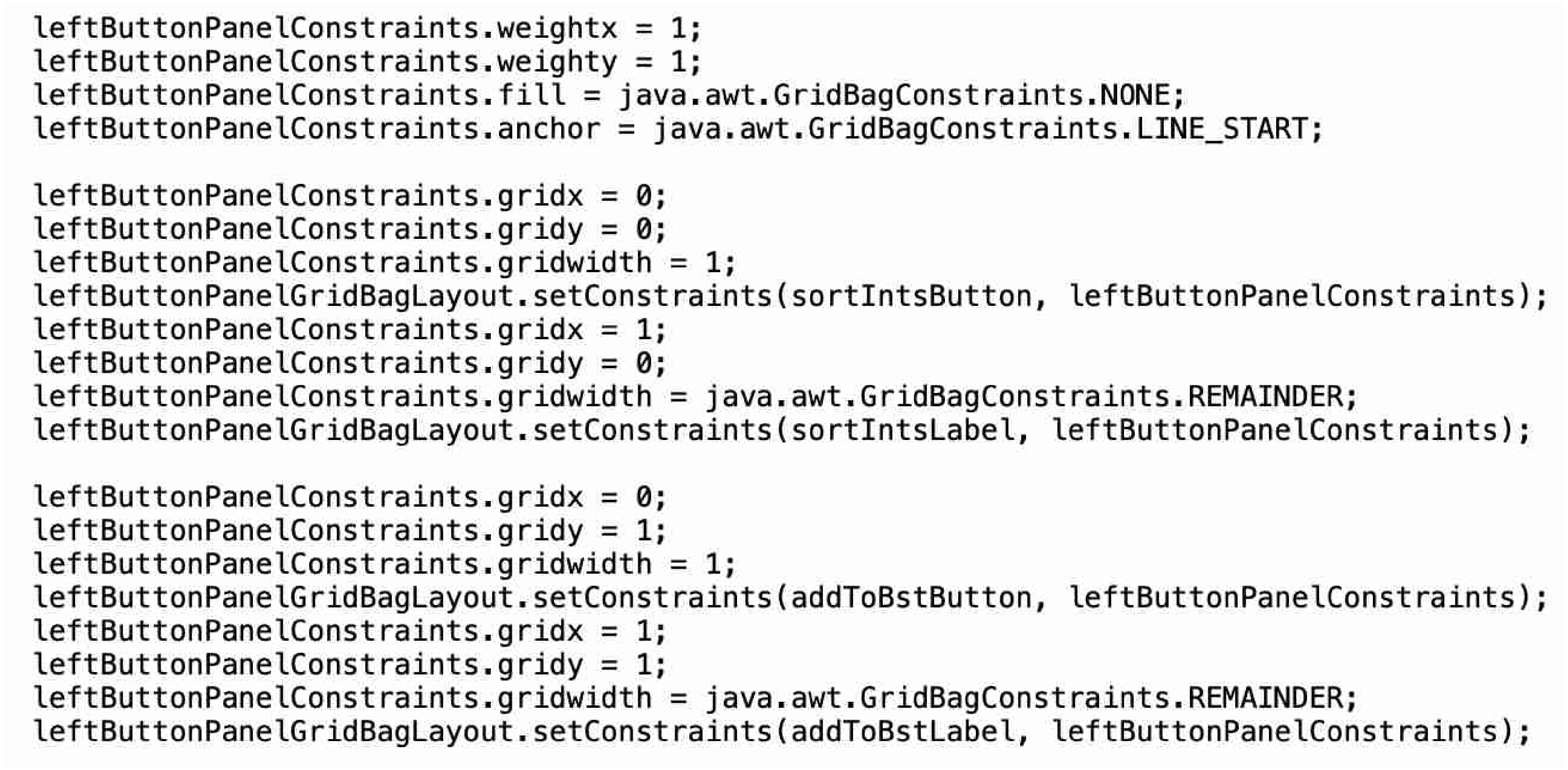
- Generally when you create a GUI, if any of the work behind the GUI is going to take time, the GUI and the “behind the scenes” computations are done in seperate threads
  - This stops the GUI from “freezing” while waiting for the “behind the scenes” computations to complete
  - I'm not a “human machine interface”, HMI, expert, but it seems like anything that takes more 10 – 20 ms is long enough that the delay might be noticable
- We aren't going to do that for this program
- When you select the “sort ints”, “search arraylist”, “search priority queue”, and “search array” for the larger test files, you will definitely notice that the GUI will “freeze” while waiting for the computations to complete
- Sorting the 100,000 value file with selection sort takes around 3.5 seconds on my laptop, and the 250,000 value file takes around 23 seconds

- The GridBagLayout manager is fairly complex
- Here is what my code looks like for the leftButtonPanel
  - Instantiate the panel
  - Instantiate the gridbag layout
  - Set the layout of the panel
  - Instantiate the left buttons and labels



```
javax.swing.JPanel leftButtonPanel = new javax.swing.JPanel();
javax.swing.border.LineBorder leftButtonPanelLineBorder = new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 0, 255), borderWidth);
leftButtonPanel.setBorder(leftButtonPanelLineBorder);
java.awt.GridBagLayout leftButtonPanelGridBagLayout = new java.awt.GridBagLayout();
leftButtonPanel.setLayout(leftButtonPanelGridBagLayout);
java.awt.GridBagConstraints leftButtonPanelConstraints = new java.awt.GridBagConstraints();

sortIntsButton = new javax.swing.JButton("sort ints");
sortIntsLabel = new javax.swing.JLabel("no result");
addToBstButton = new javax.swing.JButton("add to bst");
addToBstLabel = new javax.swing.JLabel("no result");
addToTreeSetButton = new javax.swing.JButton("add to treeset");
addToTreeSetLabel = new javax.swing.JLabel("no result");
addToPriorityQueueButton = new javax.swing.JButton("add to priority queue");
addToPriorityQueueLabel = new javax.swing.JLabel("no result");
addToHashSetButton = new javax.swing.JButton("add to hashset");
addToHashSetLabel = new javax.swing.JLabel("no result");
addToArrayListButton = new javax.swing.JButton("add to arraylist");
addToArrayListLabel = new javax.swing.JLabel("no result");
addToSortedArrayListButton = new javax.swing.JButton("add to sorted arraylist");
addToSortedArrayListLabel = new javax.swing.JLabel("no result");
addToArrayButton = new javax.swing.JButton("add to array");
addToArrayLabel = new javax.swing.JLabel("no result");
```

- Here is what my code looks like for this first couple of buttons and labels being added to the leftButtonPanel

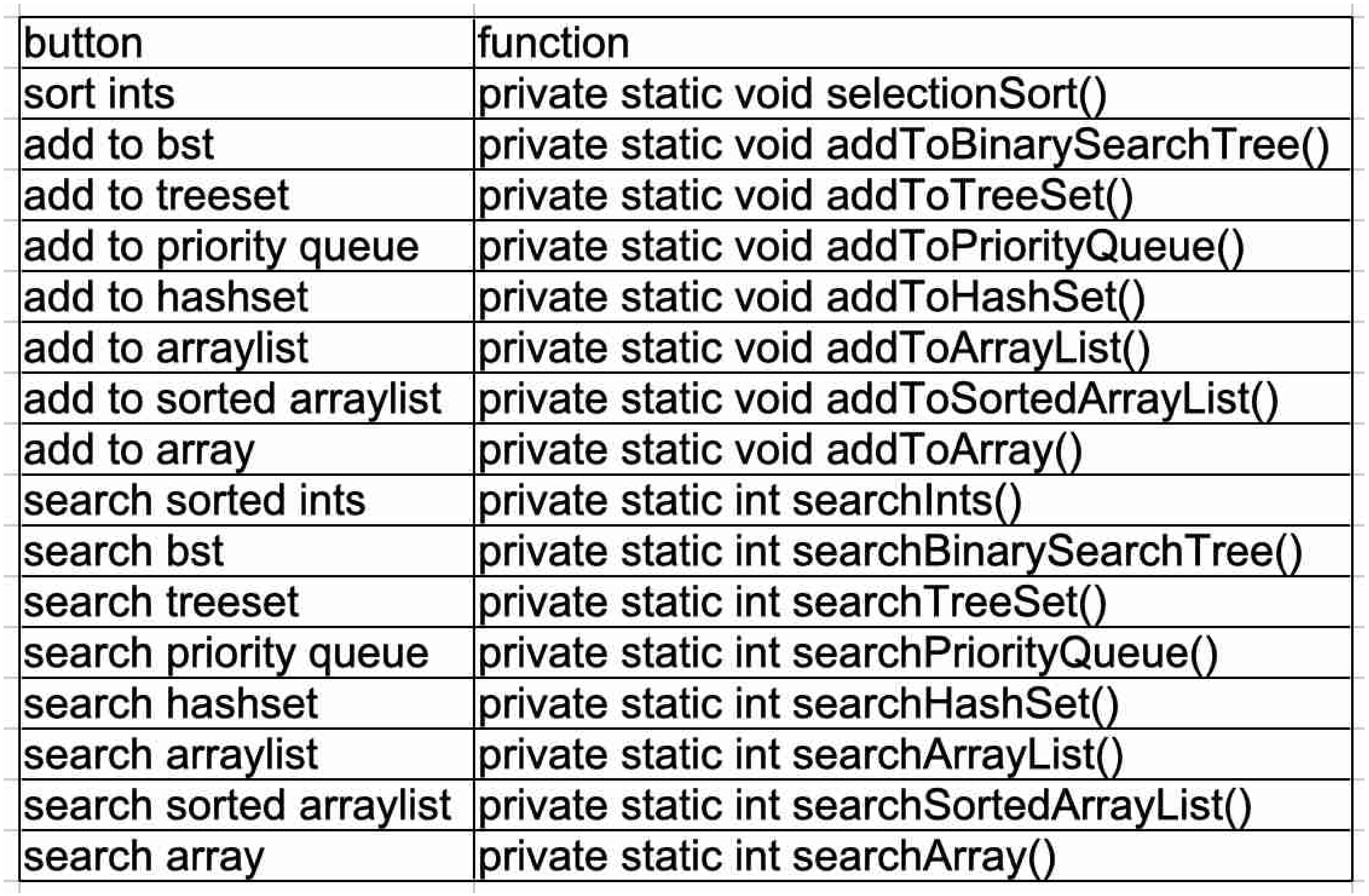


```
leftButtonPanelConstraints.weightx = 1;
leftButtonPanelConstraints.weighty = 1;
leftButtonPanelConstraints.fill = java.awt.GridBagConstraints.NONE;
leftButtonPanelConstraints.anchor = java.awt.GridBagConstraints.LINE_START;

leftButtonPanelConstraints.gridx = 0;
leftButtonPanelConstraints.gridy = 0;
leftButtonPanelConstraints.gridwidth = 1;
leftButtonPanelGridBagLayout.setConstraints(sortIntsButton, leftButtonPanelConstraints);
leftButtonPanelConstraints.gridx = 1;
leftButtonPanelConstraints.gridy = 0;
leftButtonPanelConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
leftButtonPanelGridBagLayout.setConstraints(sortIntsLabel, leftButtonPanelConstraints);

leftButtonPanelConstraints.gridx = 0;
leftButtonPanelConstraints.gridy = 1;
leftButtonPanelConstraints.gridwidth = 1;
leftButtonPanelGridBagLayout.setConstraints(addToBstButton, leftButtonPanelConstraints);
leftButtonPanelConstraints.gridx = 1;
leftButtonPanelConstraints.gridy = 1;
leftButtonPanelConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
leftButtonPanelGridBagLayout.setConstraints(addToBstLabel, leftButtonPanelConstraints);
```

- For my JButtons and JLabels, I declared them as static fields in the class, and when initially instantiated the JButtons are disabled
- When the input files are read, I enabled the appropriate JButtons
  - The “search” buttons are enabled when both the search file is read and the corresponding “sort”/”add” button is selected
    - Your code needs to handle both cases where the search file is read before or after the corresponding “sort”/”add” button is selected
- Each of the JButtons need to have an ActionListener defined for it, see example2a.java
- All of my code is in a single file, this is not required, but it does work
  - Except for the binary search tree code, which I re-used from program 5
    - If your program 5 is not functional, use a TreeSet

- Here is a list of the methods in my code
  - private static void selectionSort()
  - private static int searchInts()
  - private static void addToBinarySearchTree()
  - private static int searchBinarySearchTree()
  - private static void addToTreeSet()
  - private static int searchTreeSet()
  - private static void addToHashSet()
  - private static int searchHashSet()
  - private static void addToPriorityQueue()
  - private static int searchPriorityQueue()
  - private static void addToArrayList()
  - private static int searchArrayList()
  - private static void addToSortedArrayList()
  - private static int searchSortedArrayList()
  - private static void addToArray()
  - private static int searchArray()
  - private static void readData(String filename, boolean readSortValues)
- Here are the two inner classes that I use (see example2a.java)
  - static class MenuItemActionListener implements java.awt.event.ActionListener
  - static class ButtonActionListener implements java.awt.event.ActionListener

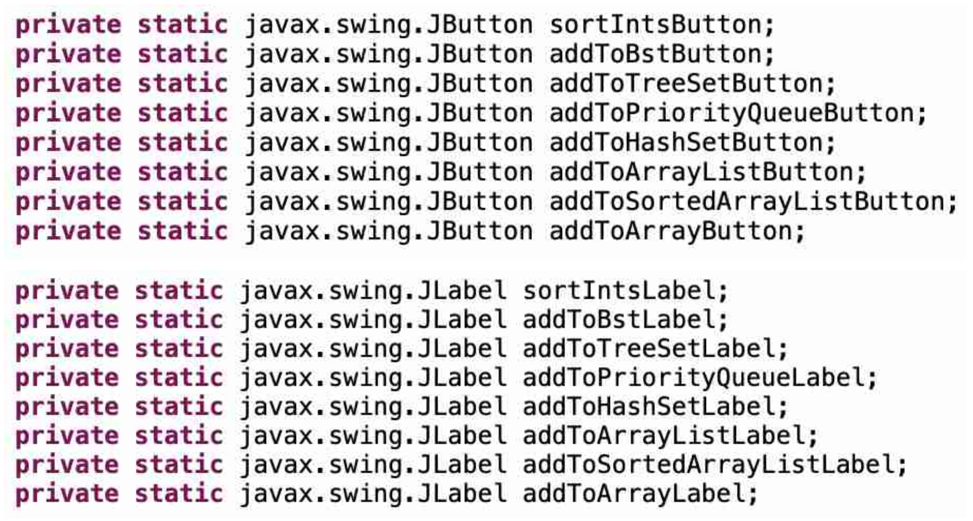
- Here is a cross reference between my buttons and the functions



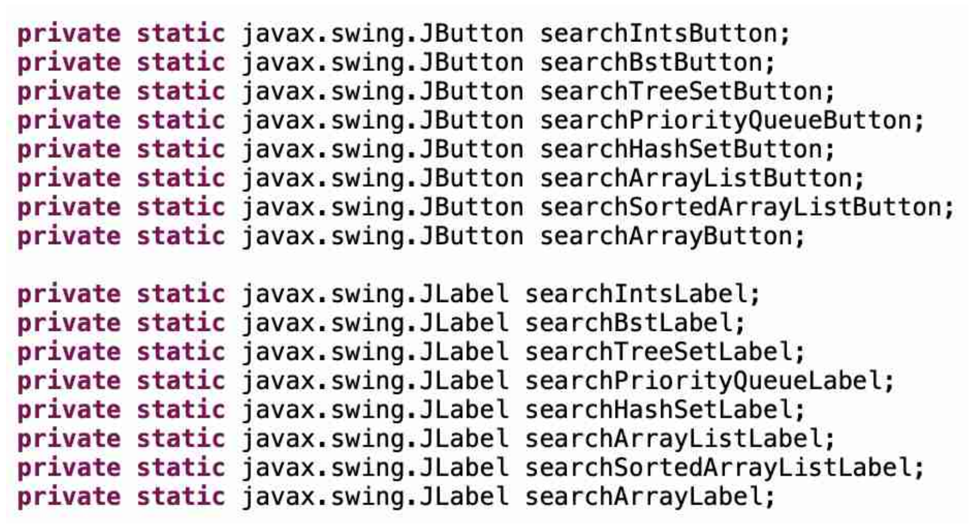
button	function
sort ints	private static void selectionSort()
add to bst	private static void addToBinarySearchTree()
add to treeset	private static void addToTreeSet()
add to priority queue	private static void addToPriorityQueue()
add to hashset	private static void addToHashSet()
add to arraylist	private static void addToArrayList()
add to sorted arraylist	private static void addToSortedArrayList()
add to array	private static void addToArray()
search sorted ints	private static int searchInts()
search bst	private static int searchBinarySearchTree()
search treeset	private static int searchTreeSet()
search priority queue	private static int searchPriorityQueue()
search hashset	private static int searchHashSet()
search arraylist	private static int searchArrayList()
search sorted arraylist	private static int searchSortedArrayList()
search array	private static int searchArray()

- And the two menu items both use “private static void readData(String filename, boolean readSortValues)”

- Here are my button and label declarations
  - Since they are fields of the class, they can be access directly by the methods defined in the class (to enable)
  - They can't be accessed directly by the inner classes of the main class/file, but they can be accessed if we pass them as a parameter of the inner class constructor (see example2a.java)
  - You don't need to use my variable names, they are just to show you an example of what I did

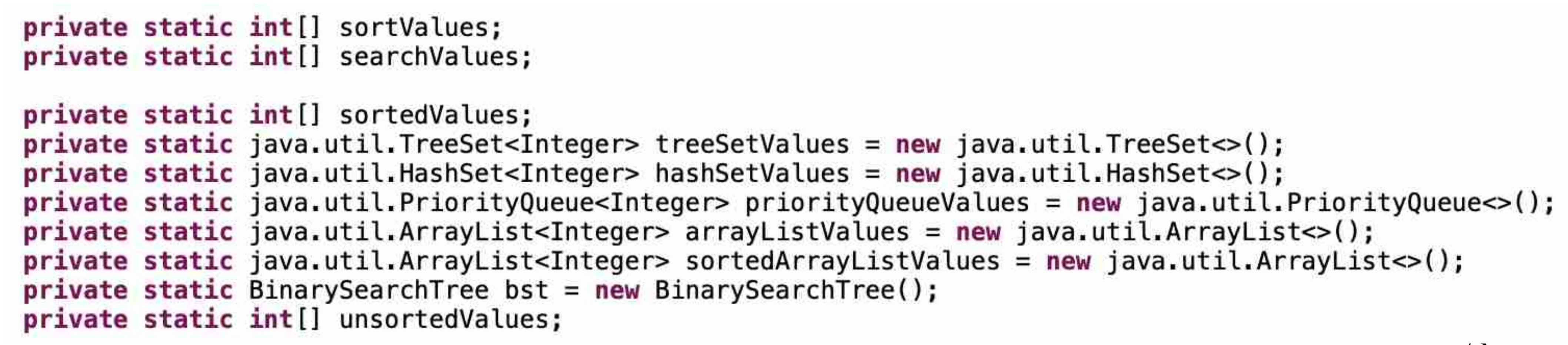


```java
private static javax.swing.JButton sortIntsButton;
private static javax.swing.JButton addToBstButton;
private static javax.swing.JButton addToTreeSetButton;
private static javax.swing.JButton addToPriorityQueueButton;
private static javax.swing.JButton addToHashSetButton;
private static javax.swing.JButton addToArrayListButton;
private static javax.swing.JButton addToSortedArrayListButton;
private static javax.swing.JButton addToArrayButton;

private static javax.swing.JLabel sortIntsLabel;
private static javax.swing.JLabel addToBstLabel;
private static javax.swing.JLabel addToTreeSetLabel;
private static javax.swing.JLabel addToPriorityQueueLabel;
private static javax.swing.JLabel addToHashSetLabel;
private static javax.swing.JLabel addToArrayListLabel;
private static javax.swing.JLabel addToSortedArrayListLabel;
private static javax.swing.JLabel addToArrayLabel;
```



```java
private static javax.swing.JButton searchIntsButton;
private static javax.swing.JButton searchBstButton;
private static javax.swing.JButton searchTreeSetButton;
private static javax.swing.JButton searchPriorityQueueButton;
private static javax.swing.JButton searchHashSetButton;
private static javax.swing.JButton searchArrayListButton;
private static javax.swing.JButton searchSortedArrayListButton;
private static javax.swing.JButton searchArrayButton;

private static javax.swing.JLabel searchIntsLabel;
private static javax.swing.JLabel searchBstLabel;
private static javax.swing.JLabel searchTreeSetLabel;
private static javax.swing.JLabel searchPriorityQueueLabel;
private static javax.swing.JLabel searchHashSetLabel;
private static javax.swing.JLabel searchArrayListLabel;
private static javax.swing.JLabel searchSortedArrayListLabel;
private static javax.swing.JLabel searchArrayLabel;
```

- Here are my data storage declarations
  - int[] sortValues contains the list of values read from the “sort file”
  - int[] searchValues contains the list of values read from the “search file”
  - int[] sortedValues is the array that is sorted in selectionSort()
  - treeSetValues<Integer> is the tree set used for adding to and searching
  - hashSetValues<Integer> is the hash set used for adding to and searching
  - priorityQueueValues<Integer> is the priority queue used for adding to and searching
  - arrayListValues<Integer> is the unsorted ArrayList used for adding to and searching
  - sortedArrayListValues<Integer> is the sorted ArrayList used for adding to and searching
  - int[] unsortValues is the unsorted int[] used for adding to and searching
  - bst is the binary search tree used for adding to and searching



```java
private static int[] sortValues;
private static int[] searchValues;

private static int[] sortedValues;
private static java.util.TreeSet<Integer> treeSetValues = new java.util.TreeSet<>();
private static java.util.HashSet<Integer> hashSetValues = new java.util.HashSet<>();
private static java.util.PriorityQueue<Integer> priorityQueueValues = new java.util.PriorityQueue<>();
private static java.util.ArrayList<Integer> arrayListValues = new java.util.ArrayList<>();
private static java.util.ArrayList<Integer> sortedArrayListValues = new java.util.ArrayList<>();
private static BinarySearchTree bst = new BinarySearchTree();
private static int[] unsortedValues;
```

- This is not required, I added keyboard short cuts
- The “File” menu
  - `javax.swing.JMenu fileMenu = new javax.swing.JMenu("File");`
  - `fileMenu.setMnemonic(java.awt.event.KeyEvent.VK_F); // ctrl-option f on mac os`
- The “Read sort file” menu item
  - `javax.swing.JMenuItem fileReadSortFile = new javax.swing.JMenuItem("Read sort file");`
  - `fileReadSortFile.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_O, java.awt.event.ActionEvent.ALT_MASK));`
- The “Read search file” menu item
  - `javax.swing.JMenuItem fileReadSearchFile = new javax.swing.JMenuItem("Read search file");`
  - `fileReadSearchFile.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_A, java.awt.event.ActionEvent.ALT_MASK));`
- The “Exit” menu item
  - `javax.swing.JMenuItem fileExit = new javax.swing.JMenuItem("Exit");`
  - `fileExit.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_E, java.awt.event.ActionEvent.ALT_MASK));`

- To get the time estimates, simply capture the time just before and just after calling the various methods
  - long t0 = System.currentTimeMillis()
  - long t1 = System.currentTimeMillis()
  - Then set the text to (t1-t0)+"ms" for the appropriate label
- For the two ActionListeners, in their actionPerformed() methods, I have a collection of "if" statements to determine which button or menu item was selected, and then call the appropriate method to perform the required action
  - See example2a.java
- The vast majority of my code is related to the GUI, the other code, reading the input files, and adding/sorting/searching for values is quite small
  - For the TreeSet, HashSet, unsorted ArrayList, and PriorityQueue we are just using the add() and contains() methods
  - For the binary search tree, we are just doing the insertNode() and getNode()

- The search methods return the number of values found
- The label associated with the “search” buttons are updated with the number of search values found and the time spent searching for them
  - The labels should have “X / Yms” after performing the search
    - Where X is the number of values found in the search and Y is the number of milliseconds to perform the search

- Binary search pseudo code

```
int bottom = 0;
int top = values.length-1;
while( bottom <= top )
{
        int middle = (bottom+top)/2;
        if( searchValue < values[middle] )
        {
                top = middle-1;
        }
        else
        {
                if( searchValue > values[middle] )
                {
                        bottom = middle+1;
                }
                else
                {
                        // searchValue is found
                }
        }
}
```

- Only works with sorted arrays