Documentation: CS 401 Project

Problem Specification

There can be such situations that we need to arrange the data in ascending or descending manner or any other order according to our needs. We can have situations where we can interact with millions of data. So in that case we need an efficient method which can arrange the data in a faster manner.

There can be also some situations where we need to search the data from a bunch of data. We can have situations where we can interact with millions of data. So in that case we need an efficient method which can search the data in a faster manner.

Software Specification

The following application is able to perform Sorting and Searching operations on set of data of type String, Integer and Double. The application implements Bubble Sort & Quick Sort for Sorting operation and implements Linear Search and Binary Search for Searching operation.

The application has following functionalities:

- User can select the type of array (String, Integer, Double) to be sorted/searched from radio buttons
- Under Sorting:
    - User can enter list of values to be sorted in Text Field
    - Sort button is available to perform sorting operation
    - Sorted list is displayed in Text Field
    - Time consumed (nanosecond) by both the sorting algorithms (Bubble and Quick) is displayed
    - Total count of operations performed by both the sorting algorithms (Bubble and Quick) is also displayed
- Under Searching:
    - User can enter list of values to be searched within in Text Field
    - User can enter the value to be searched in Text Field
    - Search button is available to perform searching operation
    - Sorted list is displayed in Text Field
    - Time consumed (nanosecond) by both the searching algorithms (Linear and Binary) is displayed
    - Total count of operations performed by both the sorting algorithms (Linear and Binary) is also displayed
    - Searched result will be printed if the value is present in the user's list or not. If found then application also shows the position number of that value

Operational document

Application is responsible for sorting and searching the list of input provided by the user. Following steps should be followed to run the application successfully.

i. At first user has to select the type of array that will be sorted and or searched. Radio buttons are available to select between Integer array, Double array and String array. User's selection will be shown at right side of the radio button. Select any one of the radio button.
ii. Below that application has 2 sections – Sorting section and Searching section.
iii. After selecting the radio button, Text Fields will be enabled to enter the selected type of array for sorting and searching.
iv. Under Sorting section: Enter your input (list/array) in "," separated format in text field.
v. Application shows sorting algorithms (Bubble Sort and Quick Sort) on the basis of which sorting operation is performed.
vi. Click on Sort button
vii. Now application will show you following details under sorting section:
    - Time consumed for the operation in nanosecond by both the algorithms
    - Total count of operation by both the algorithms
    - Sorted list by both the algorithms
viii. Under Searching section: Enter your input (list/array) in "," separated format in text field.
ix. Enter a value to be searched in text field
x. Application shows searching algorithms (Linear Search and Binary Search) on the basis of which searching operation is performed.
xi. Click on Search button
xii. Now application will show you following details under searching section:
    - Sorted list
    - Time consumed for the operation in nanosecond by both the algorithms
    - Total count of operation by both the algorithms
    - Searched result by both the algorithms if the entered value is present in the list or not. If present then application also shows the position of the value

Project Management/schedule

Following time schedule has been followed to complete this project.

12th June – Project Planning (arranging project environment, drew blue print of the GUI, methods required to be implemented)

13th June – Wrote code for Bubble Sort

14th June – Wrote code for Quick Sort

15th June – Wrote code for Linear Search

16th June – Wrote code for Binary Search

17th June – GUI designing (JAVA Applet)

18th June – GUI designing (JAVA Applet)

19th June – GUI designing and optimizing the code to implement them for this GUI

20th June – Integrating sorting and searching codes with GUI

21st June – Optimizing the code for better readability and improving the GUI for a better user friendly interface

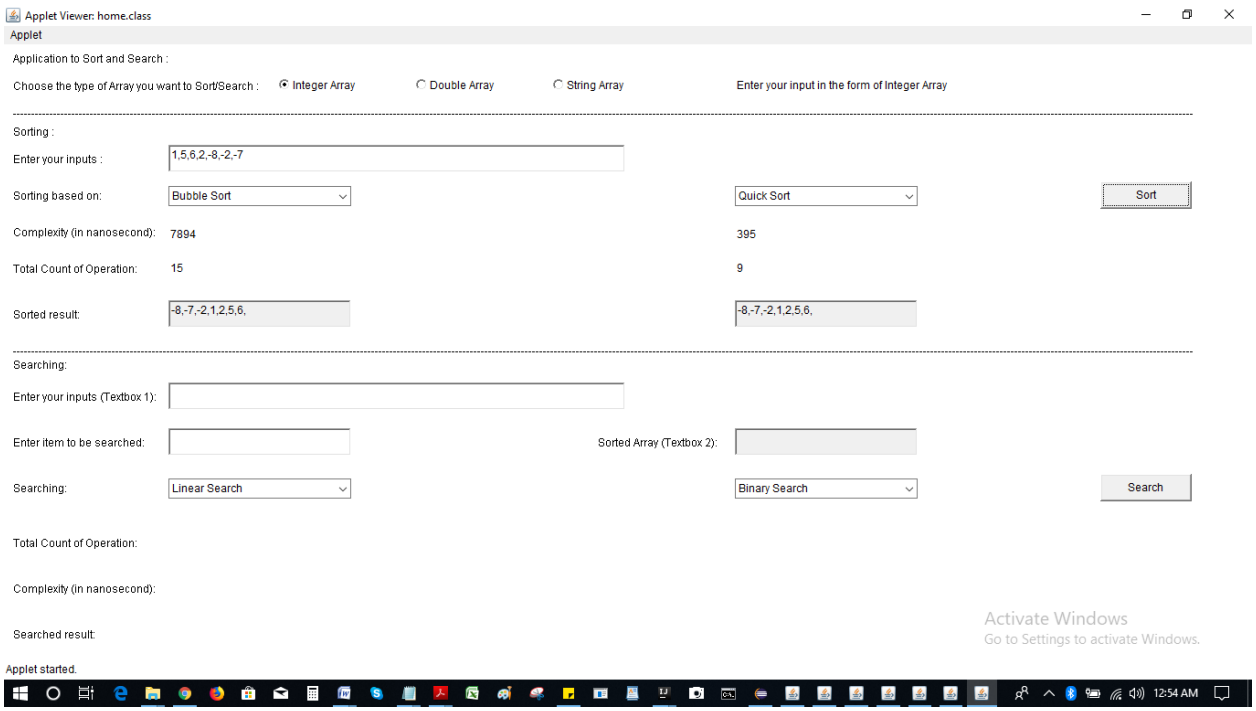
22nd June – Preparing the necessary documents for final submission

Testing Document

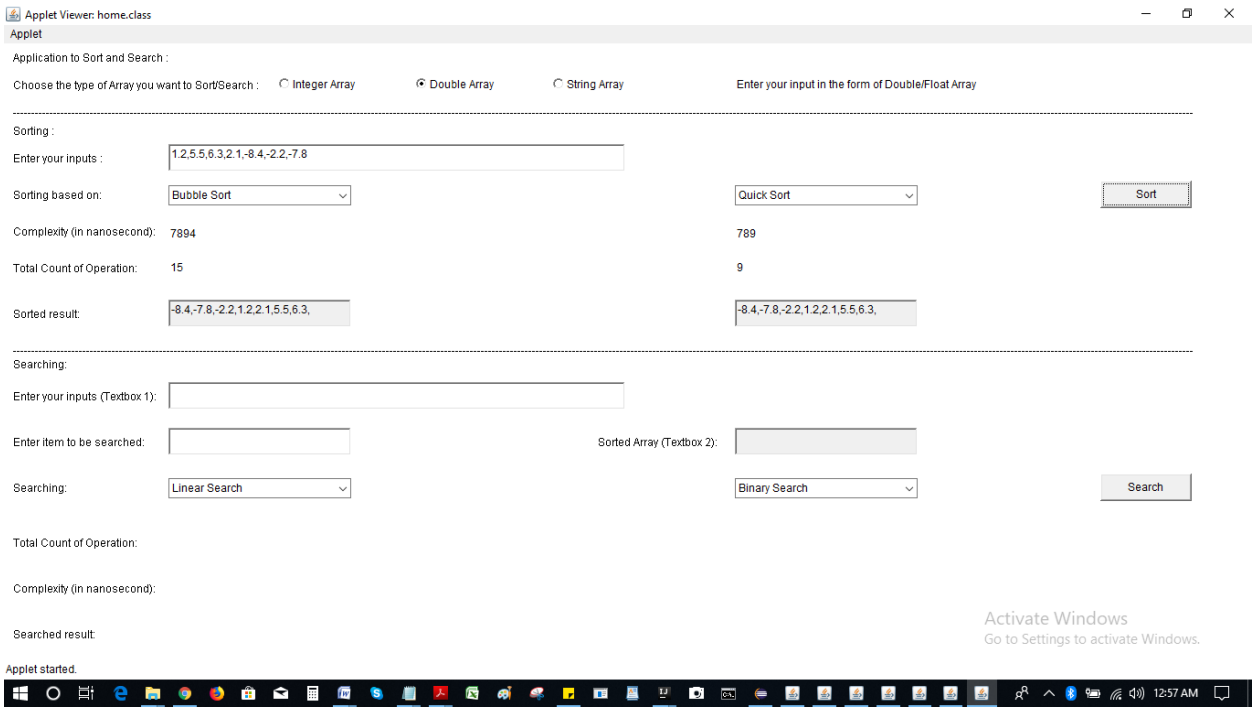
Request to enter following values in the input text field

Sorting

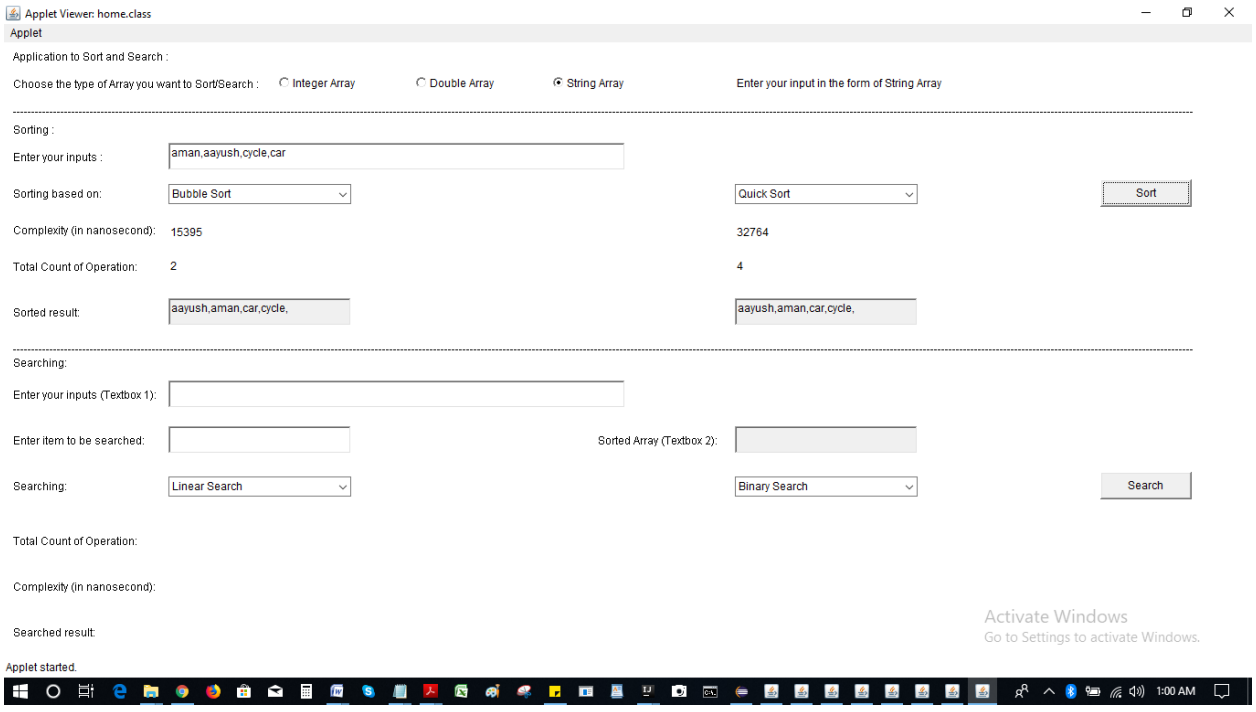
Integer Array: 1,5,6,2,-8,-2,-7



Double Array: 1.2,5.5,6.3,2.1,-8.4,-2.2,-7.8



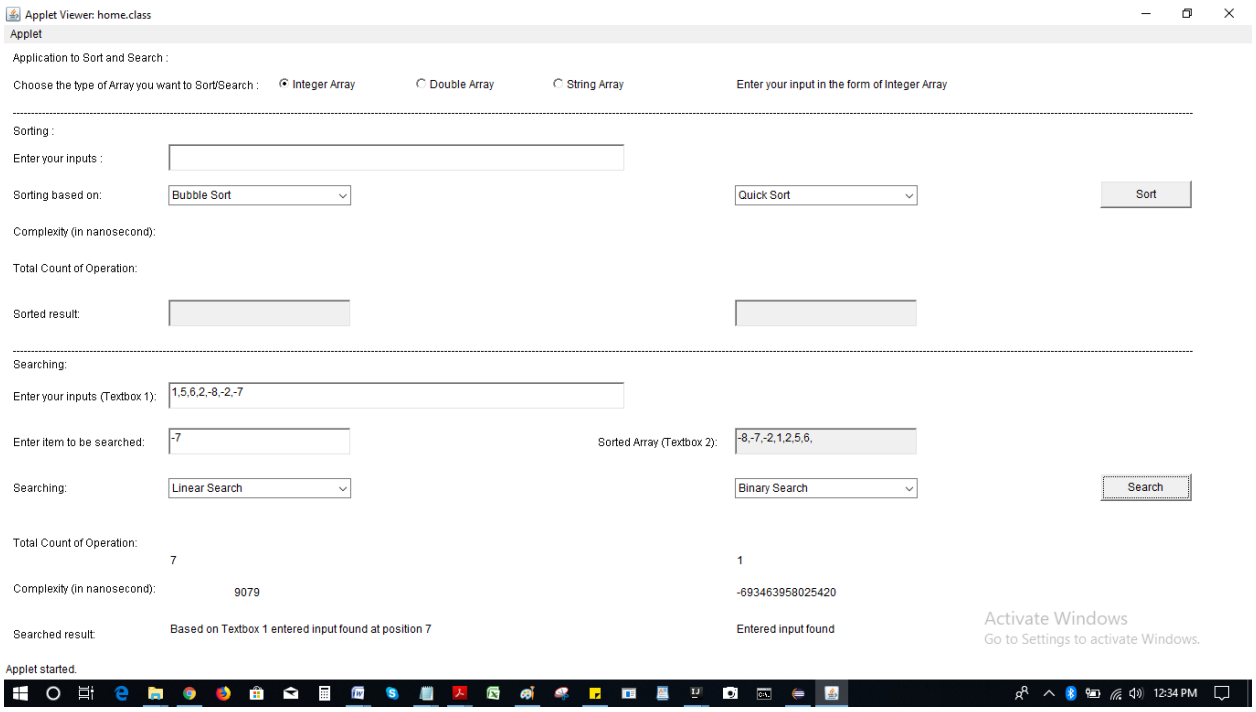
String Array: aman,aayush,cycle,car



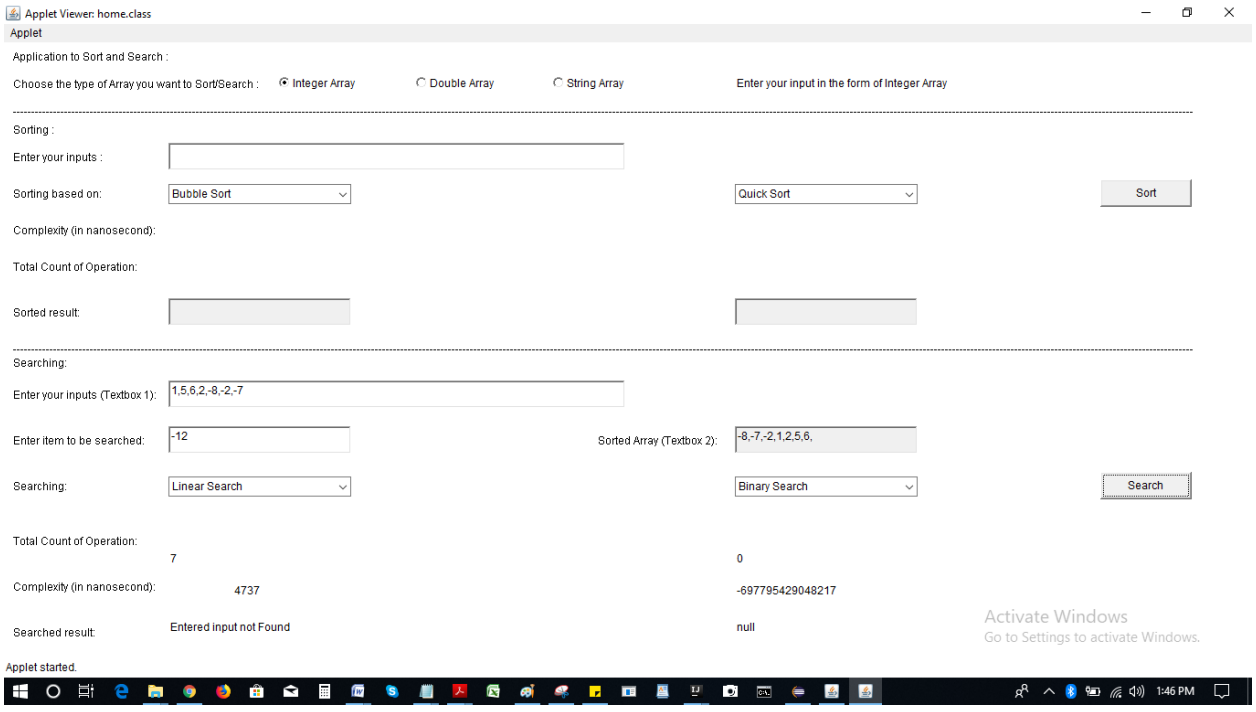
Searching

Integer Array: 1,5,6,2,-8,-2,-7

Search value for Integer Array: -7

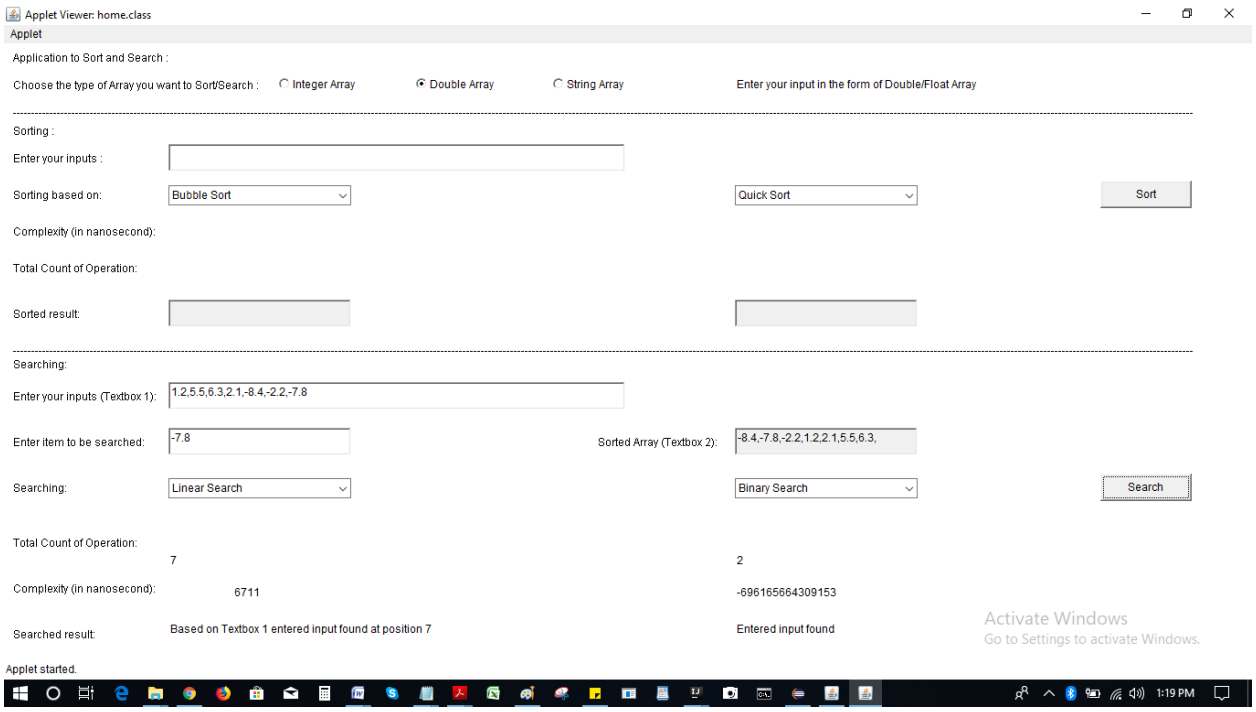


Searched value: -12

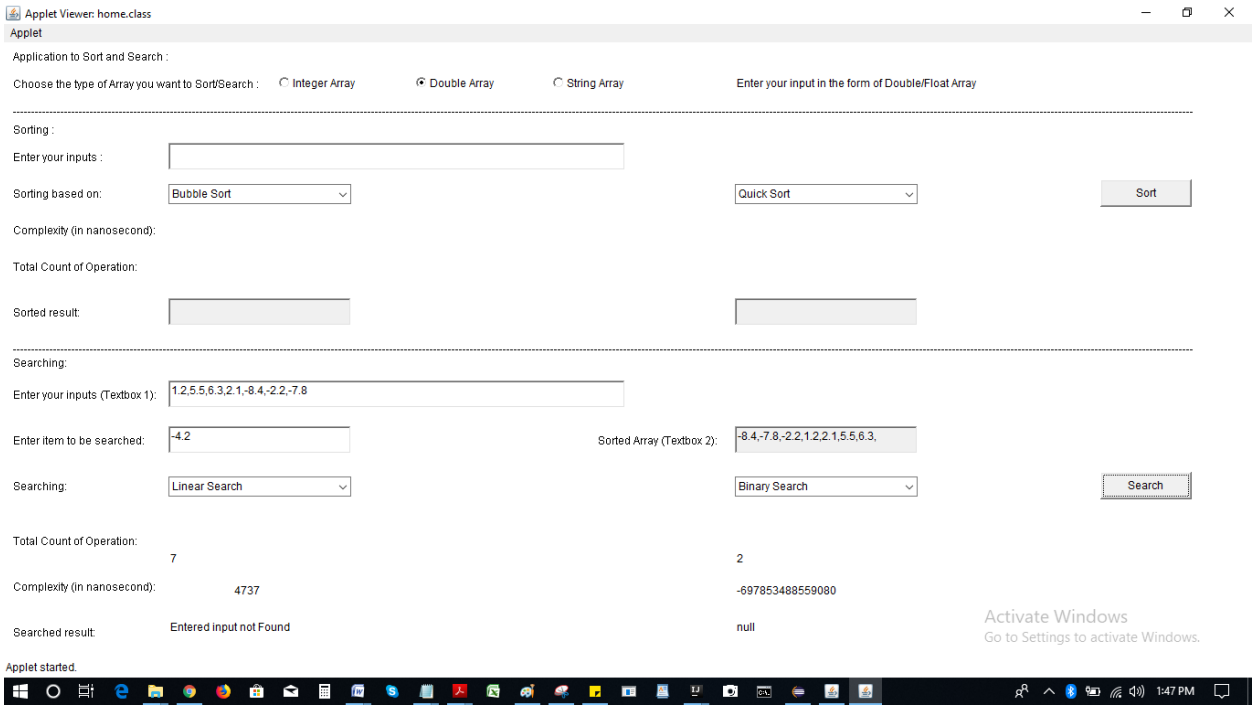


Double Array: 1.2,5.5,6.3,2.1,-8.4,-2.2,-7.8

Search value for Double Array: -7.8

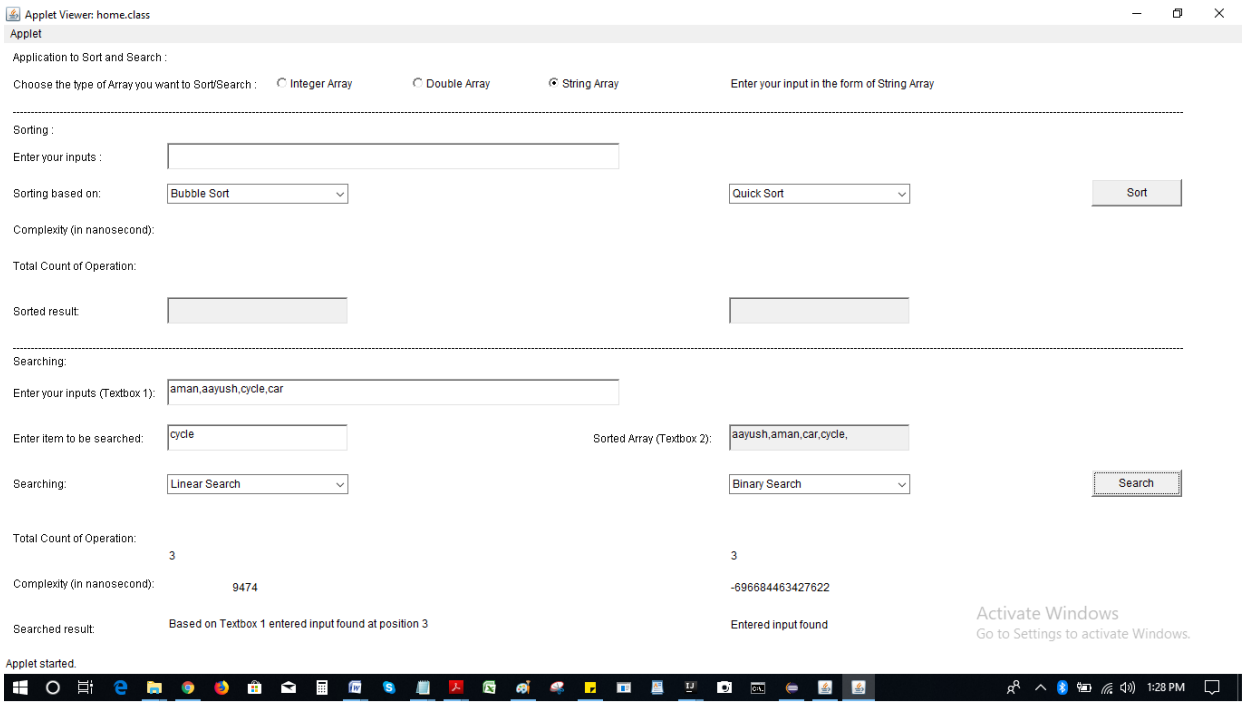


Searched value: -4.2

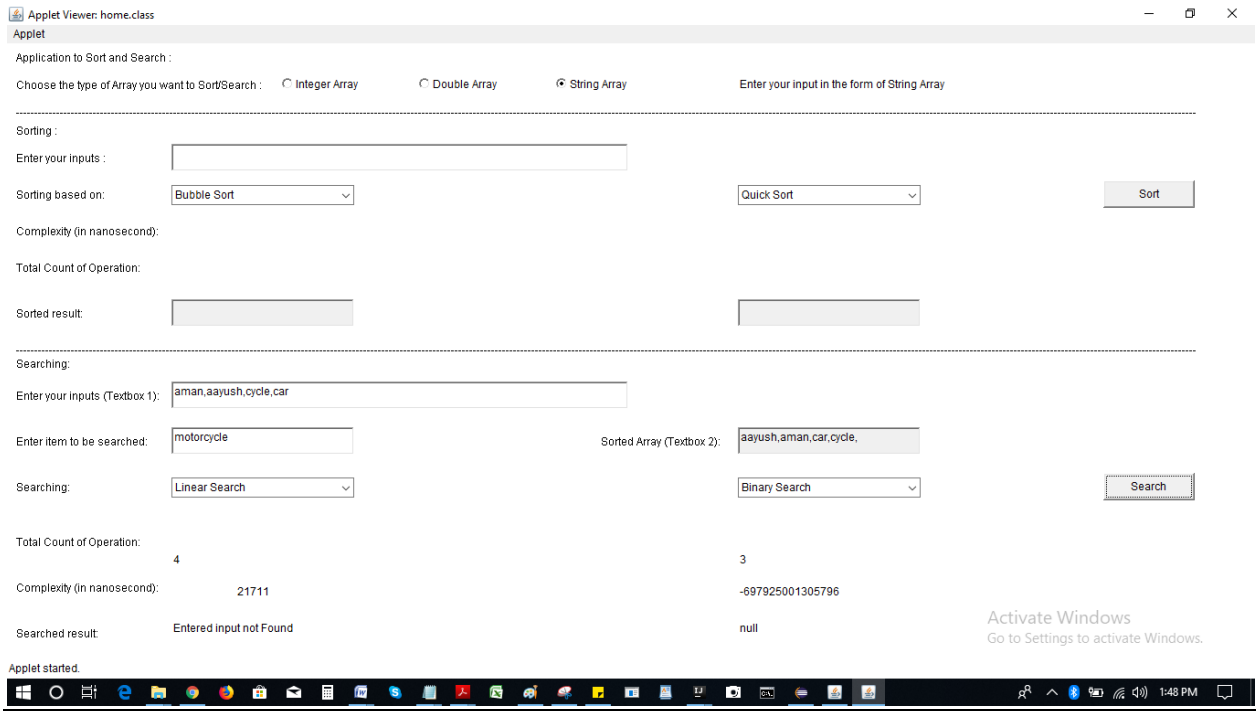


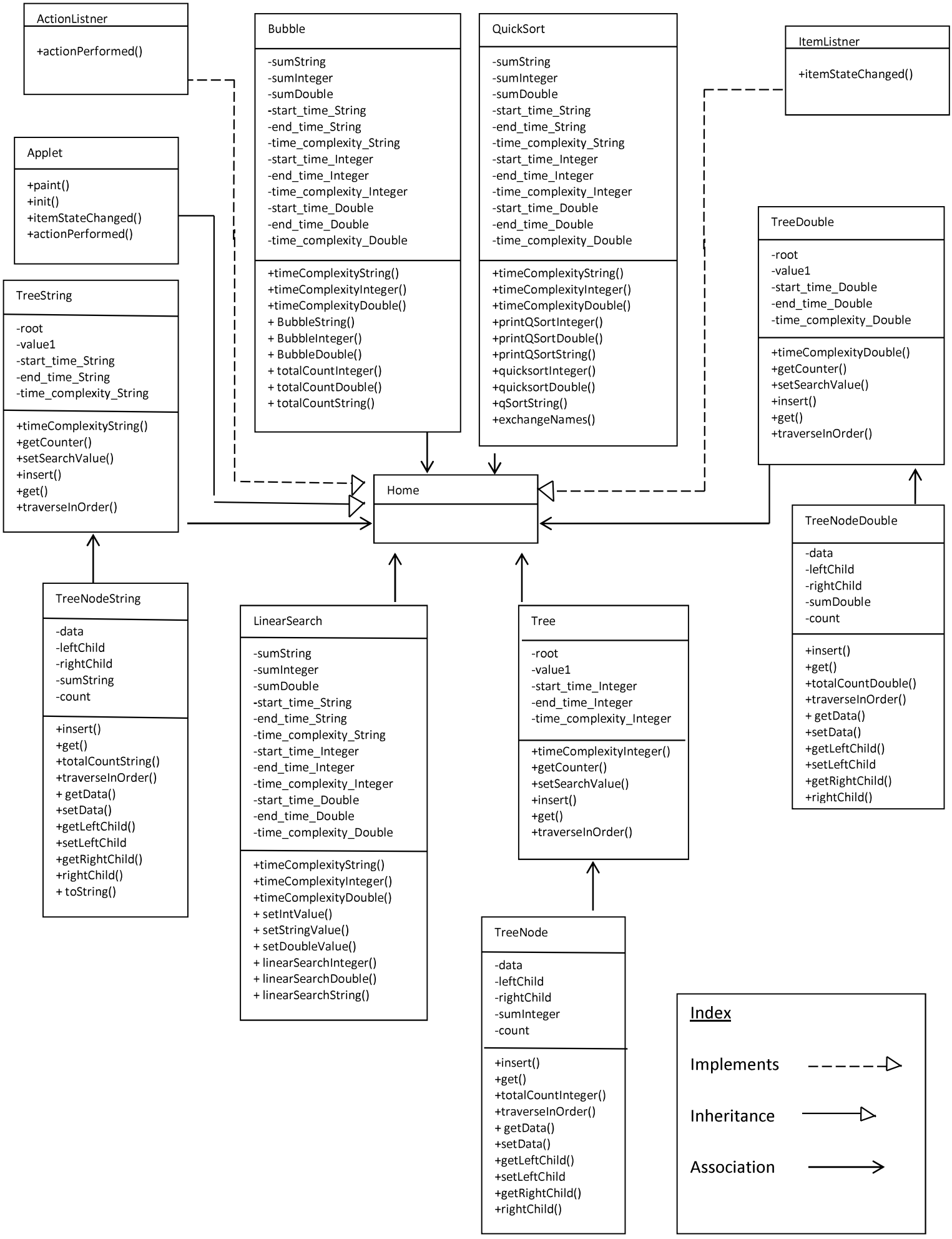
String Array: aman,aayush,cycle,car

Search value for String Array: cycle



Searched value: motorcycle



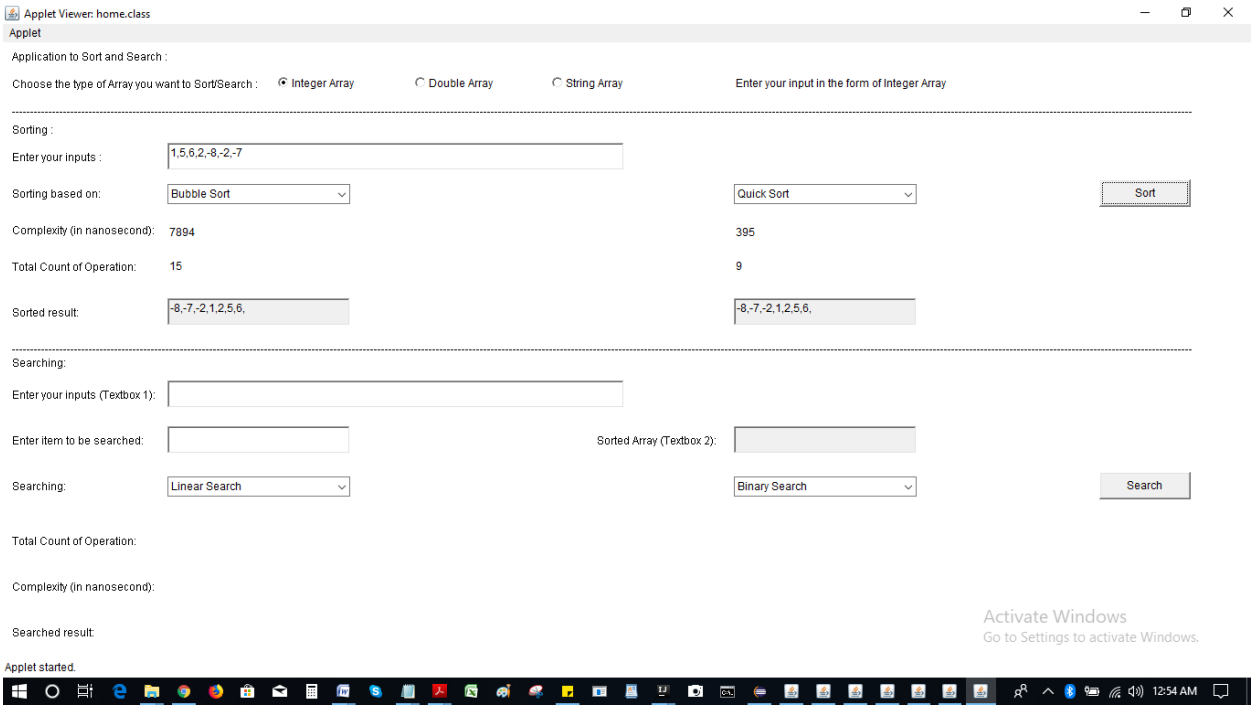

ActionListner
+actionPerformed()
Bubble
-sumString
-sumInteger
-sumDouble
-start_time_String
-end_time_String
-time_complexity_String
-start_time_Integer
-end_time_Integer
-time_complexity_Integer
-start_time_Double
-end_time_Double
-time_complexity_Double
+timeComplexityString()
+timeComplexityInteger()
+timeComplexityDouble()
+ BubbleString()
+ BubbleInteger()
+ BubbleDouble()
+ totalCountInteger()
+ totalCountDouble()
+ totalCountString()
QuickSort
-sumString
-sumInteger
-sumDouble
-start_time_String
-end_time_String
-time_complexity_String
-start_time_Integer
-end_time_Integer
-time_complexity_Integer
-start_time_Double
-end_time_Double
-time_complexity_Double
+timeComplexityString()
+timeComplexityInteger()
+timeComplexityDouble()
+printQSortInteger()
+printQSortDouble()
+printQSortString()
+quicksortInteger()
+quicksortDouble()
+qSortString()
+exchangeNames()
ItemListner
+itemStateChanged()
Applet
+paint()
+init()
+itemStateChanged()
+actionPerformed()
TreeDouble
-root
-value1
-start_time_Double
-end_time_Double
-time_complexity_Double
+timeComplexityDouble()
+getCounter()
+setSearchValue()
+insert()
+get()
+traverseInOrder()
TreeString
-root
-value1
-start_time_String
-end_time_String
-time_complexity_String
+timeComplexityString()
+getCounter()
+setSearchValue()
+insert()
+get()
+traverseInOrder()
Home
TreeNodeDouble
-data
-leftChild
-rightChild
-sumDouble
-count
+insert()
+get()
+totalCountDouble()
+traverseInOrder()
+ getData()
+setData()
+getLeftChild()
+setLeftChild
+getRightChild()
+rightChild()
TreeNodeString
-data
-leftChild
-rightChild
-sumString
-count
+insert()
+get()
+totalCountString()
+traverseInOrder()
+ getData()
+setData()
+getLeftChild()
+setLeftChild
+getRightChild()
+rightChild()
+ toString()
LinearSearch
-sumString
-sumInteger
-sumDouble
-start_time_String
-end_time_String
-time_complexity_String
-start_time_Integer
-end_time_Integer
-time_complexity_Integer
-start_time_Double
-end_time_Double
-time_complexity_Double
+timeComplexityString()
+timeComplexityInteger()
+timeComplexityDouble()
+ setIntValue()
+ setStringValue()
+ setDoubleValue()
+ linearSearchInteger()
+ linearSearchDouble()
+ linearSearchString()
Tree
-root
-value1
-start_time_Integer
-end_time_Integer
-time_complexity_Integer
+timeComplexityInteger()
+getCounter()
+setSearchValue()
+insert()
+get()
+traverseInOrder()
TreeNode
-data
-leftChild
-rightChild
-sumInteger
-count
+insert()
+get()
+totalCountInteger()
+traverseInOrder()
+ getData()
+setData()
+getLeftChild()
+setLeftChild
+getRightChild()
+rightChild()
Index
Implements
Inheritance
Association

Complexity analysis

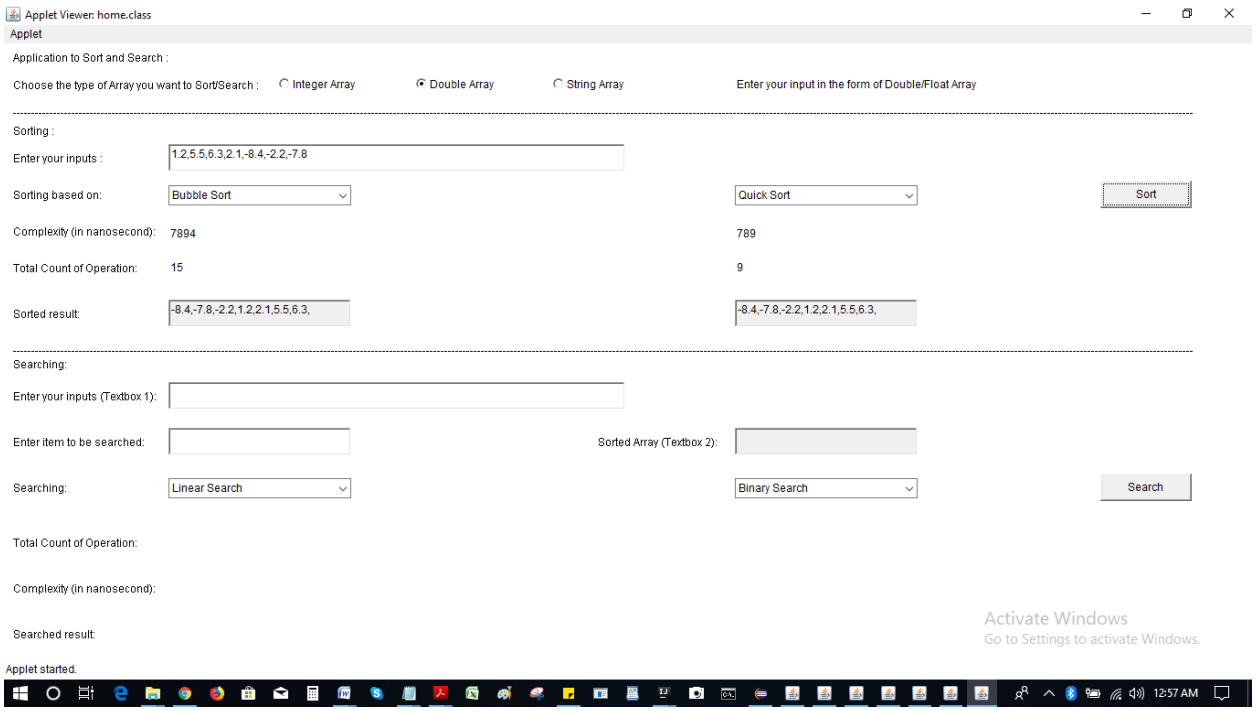
Sorting

Test Case Used and respective Screenshots for Actual results:

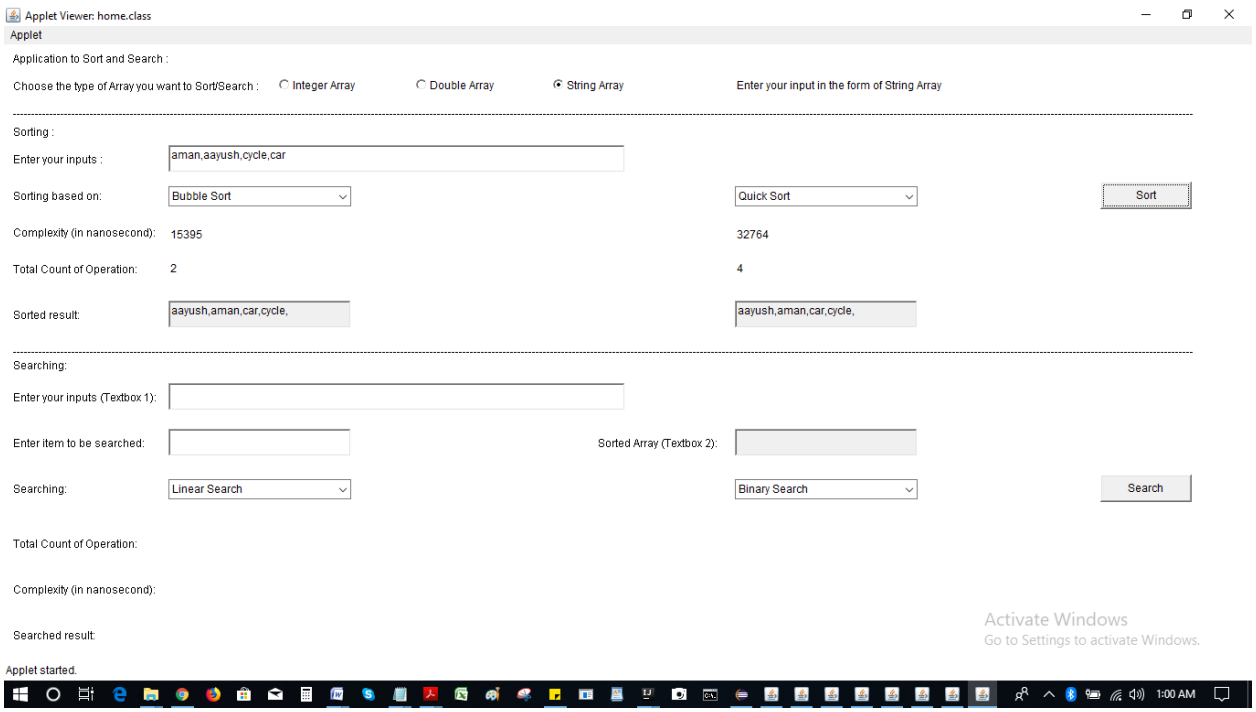
Integer Array: 1,5,6,2,-8,-2,-7



Double Array: 1.2,5.5,6.3,2.1,-8.4,-2.2,-7.8



String Array: aman,aayush,cycle,car



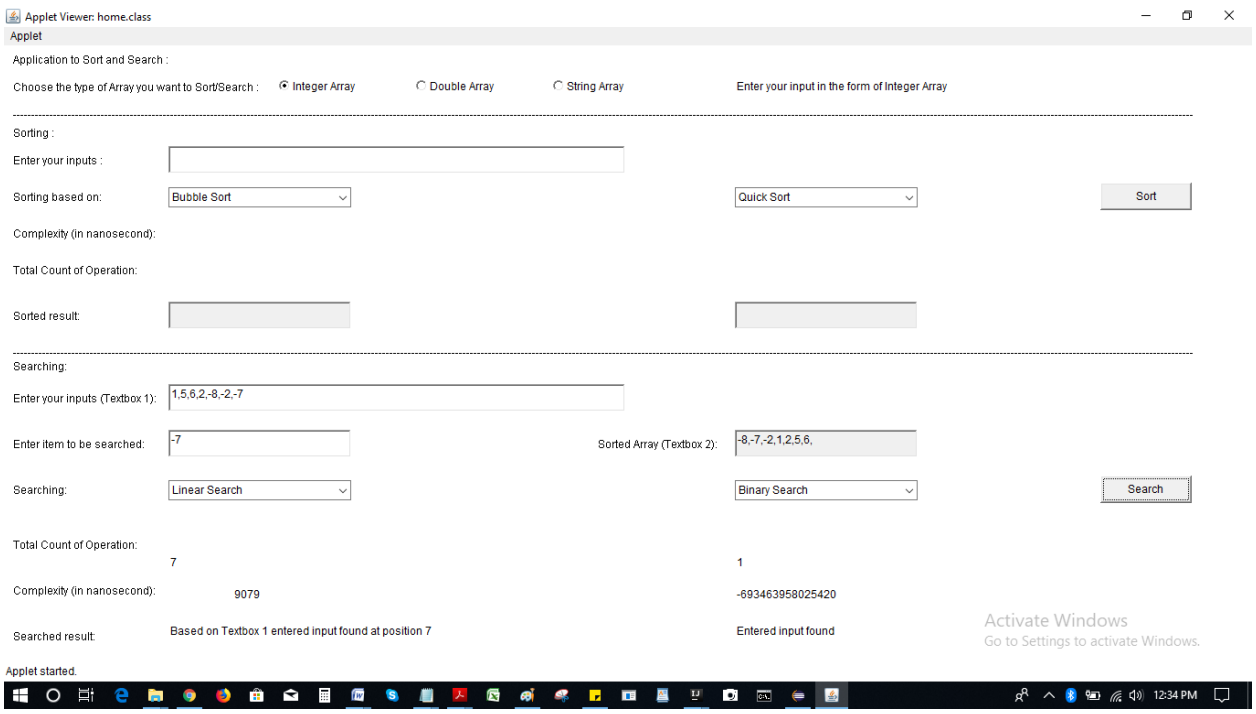
Based on above test cases and actual results following table has been drawn to compare the two sorting algorithms

Parameter for comparison	Bubble Sort	Quick Sort
For Integer Array		
Time Consumed (nanosecond)	7894	395
Total number of operations performed	15	9
For Double Array		
Time Consumed (nanosecond)	7894	789
Total number of operations performed	15	9
For String Array		
Time Consumed (nanosecond)	15395	32764
Total number of operations performed	2	4

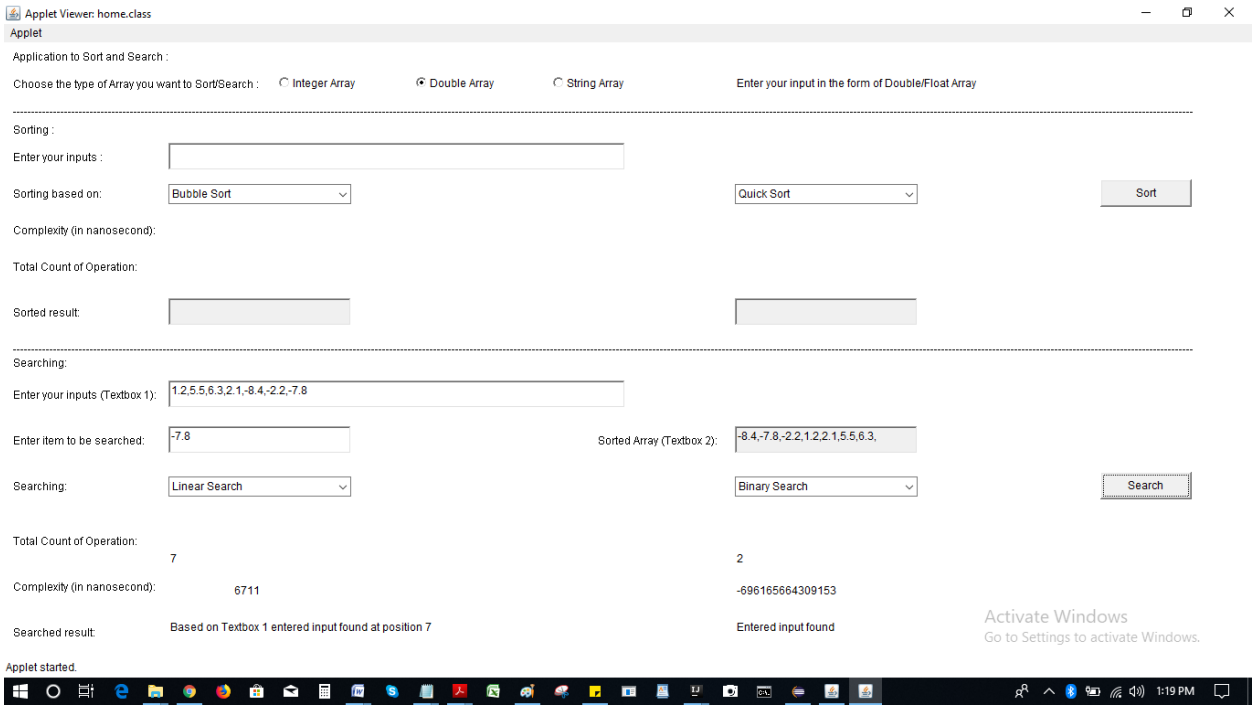
Searching

Test Cases Used:

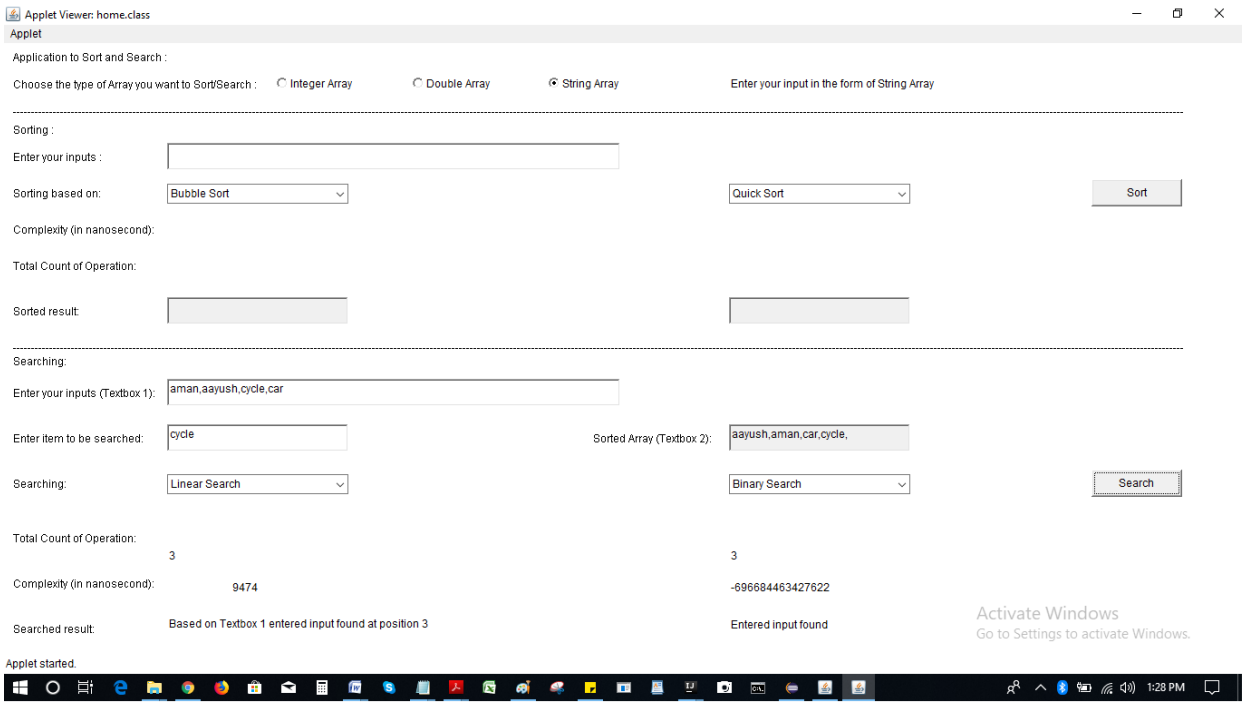
Integer Array: 1,5,6,2,-8,-2,-7



Double Array: 1.2,5.5,6.3,2.1,-8.4,-2.2,-7.8



String Array: aman,aayush,cycle,car



Based on above test cases and actual results following table has been drawn to compare the two searching algorithms

Parameter for comparison	Linear Search	Binary Search
For Integer Array		
Total number of operations performed	7	1
For Double Array		
Total number of operations performed	7	2
For String Array		
Total number of operations performed	3	3