



Design and Analysis of Algorithms

IT240

Assignment
2nd Year 2nd Semester

IT15127174 : Ranasinghe R.A.N.I.

This assignment is my own work.

.....

DESCRIPTION

The simulator can work as a e-learning tool for beginners to the algorithm field. User can upload input values to the system or allow system to generate random number sequence. Then it simulates each line execution of the algorithm.

This program simulates below algorithms,

- Bubble Sort
- Insertion Sort

LIMITATIONS

- This Algorithm Simulator was built using Java in NetBeans IDE. Older versions of Java might not support the program, Please use an updated version of Java to run the program.
- Can do sorting for only integer numbers.
- Can't do sorting with negative numbers.
- Can provide up to 12 numbers.
- Can have maximum of 4 digits for each integer

REFERENCES

<http://courses.cs.vt.edu/csonline/Algorithms/Lessons/InsertionSort/index.html>

<http://visualgo.net/sorting>

<http://stackoverflow.com/questions/16088994/sorting-an-array-of-int-using-bubblesort>

http://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms.htm

SAMPLE INPUTS AND OUTPUTS

BUBBLE SORT

The screenshot shows a web application titled "BUBBLE SORT". On the left, under the "Algorithm" tab, the following code is displayed:

```
1. int temp;  
2. for (int pass=1; pass<size; pass++)  
3.     for (int i=0; i<size - pass; i++)  
4.         if ( array [i] > array [i+1] ) TRUE  
5.             temp = array[i] ;  
6.             array[i] = array [i+1] ;  
7.             array [i+1] = temp ;
```

On the right, the "Enter Vales" input field is empty, with a note "*****Seperate values by spaces". Below it, "Array Size = 12" is shown. A row of 12 black boxes contains the numbers: 28, 34, 55, 68, 78, 10, 52, 91, 89, 70, 84, 50. Below this, "Sort in:" has two buttons: "Ascending Order" (selected) and "Descending Order". A "Generate Random Numbers" button is also present. Below the buttons, a row of 12 boxes shows the current state of the array: 10, 28, 34, 52, 50, 55, 68, 70, 78, 84, 89, 91. The boxes for 52 and 50 are black, while the others are red. A green double-headed arrow labeled "swap" connects the boxes for 52 and 50. Below the array, the text "if (52 > 50) then swap" is displayed. At the bottom right, it says "Array is Still Sorting ..".

Figure1 : Bubble sort , Ascending order (when user selects generate random numbers)

This screenshot shows the same application after the array has been sorted. The "Ascending Order" button remains selected. The array of 12 red boxes now contains the numbers: 10, 28, 34, 50, 52, 55, 68, 70, 78, 84, 89, 91. The text "Array is SORTED ✓" is displayed at the bottom right.

Figure2 : Bubble sort , Ascending order OUTPUT

INSERTION SORT

The screenshot shows the 'INSERTION SORT' application interface. On the left, the algorithm steps are listed. The main area displays the 'Enter Vales' input field with the values '85 12 456 9 41 60 12'. Below this, the 'Array Size = 7' is shown. The array elements are displayed in boxes: 85, 12, 456, 9, 41, 60, 12. The 'Sort in:' section has 'Ascending Order' selected and 'Descending Order' unselected. Below the array, the current state is shown with indices 0 to 6: 456, 85, 12, 9, 41, 60, 12. A green arrow points from the '41' at index 4 to the '9' at index 3, indicating the insertion point. The 'Key=' field shows '41'. Below this, the while loop condition is shown: 'while (3 >= 0 AND 9 < 41)' and the action 'do array[4] = array[3]'. The status 'Array is Still Sorting ...' is displayed at the bottom right.

Algorithm

1. for (int j=1; j<size; j++)
2. int key = array[j]
3. int i = j-1
4. while (i>=0 AND array[i] > key)
5. array[i+1] = array[i]
6. i = i-1
7. array[i+1] = key

Enter Vales: 85 12 456 9 41 60 12
*****Seperate values by spaces

Create Array Reset Array

Generate Random Numbers

Array Size = 7

85 12 456 9 41 60 12

Sort in: Ascending Order Descending Order

0 1 2 3 4 5 6
456 85 12 9 41 60 12

Key= 41

while (3 >= 0 AND 9 < 41)
do array[4] = array[3]

Array is Still Sorting ...

Figure3 : Insertion sort , Descending order (when user inputs values)

The screenshot shows the 'INSERTION SORT' application interface after the array has been sorted. The 'Enter Vales' input field still contains '85 12 456 9 41 60 12'. The 'Array Size = 7' is shown. The array elements are displayed in boxes: 456, 85, 60, 41, 12, 12, 9. The 'Sort in:' section has 'Ascending Order' selected and 'Descending Order' unselected. The status 'Array is SORTED ✓' is displayed at the bottom right.

Algorithm

1. for (int j=1; j<size; j++)
2. int key = array[j]
3. int i = j-1
4. while (i>=0 AND array[i] > key)
5. array[i+1] = array[i]
6. i = i-1
7. array[i+1] = key

Enter Vales: 85 12 456 9 41 60 12
*****Seperate values by spaces

Create Array Reset Array

Generate Random Numbers

Array Size = 7

85 12 456 9 41 60 12

Sort in: Ascending Order Descending Order

0 1 2 3 4 5 6
456 85 60 41 12 12 9

Array is SORTED ✓

Figure4 : Insertion sort , Descending order OUTPUT

SOURCE CODES

BUBBLE SORT

```
package daa;

import AppPackage.AnimationClass;
import java.awt.Color;
import java.awt.Font;
import java.util.StringTokenizer;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.UIManager;

public class BubbleSort extends javax.swing.JFrame {
    JLabel[] labelsIn = new JLabel[12]; //CREATE AN ARRAY OF LABLES TO STORE USER INPUT VALUES
    JLabel[] labelsOut = new JLabel[12]; //CREATE AN ARRAY OF LABLES TO STORE SORTED ARRAY VALUES
    JLabel[] labelsI = new JLabel[12]; //CREATE AN ARRAY OF LABLES TO STORE i VALUES
    int array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //INT ARRAY TO STORE USER INPUTS
    int noOfUserInputs = 0; //VARIABLE TO STORE NUMBER OF USER INPUT NUMBERS AT A TIME
    AnimationClass AC = new AnimationClass(); //MAKE A OBJECT FROM 'AnimationClass' CLASS

    //VARIABLES TO SORE USER SELECTED CHOICE WHETHER ASCENDING OD DESCENDING
    int choiceAscen = 0;
    int choiceDescen = 0;

    public BubbleSort() {
        initComponents();

        pnl_array.setVisible(false);
        clearArrayA();
        clearSortingA();

        //----INIATILZE INPUT LABEL ARRAY 'labelsIn'
        labelsIn[0] = lbl_n1;
        labelsIn[1] = lbl_n2;
        labelsIn[2] = lbl_n3;
        labelsIn[3] = lbl_n4;
        labelsIn[4] = lbl_n5;
        labelsIn[5] = lbl_n6;
        labelsIn[6] = lbl_n7;
        labelsIn[7] = lbl_n8;
        labelsIn[8] = lbl_n9;
        labelsIn[9] = lbl_n10;
        labelsIn[10] = lbl_n11;
        labelsIn[11] = lbl_n12;

        //----INIATILZE INPUT LABEL ARRAY 'labelsOut'
        labelsOut[0] = lbl_s1;
        labelsOut[1] = lbl_s2;
        labelsOut[2] = lbl_s3;
        labelsOut[3] = lbl_s4;
        labelsOut[4] = lbl_s5;
        labelsOut[5] = lbl_s6;
        labelsOut[6] = lbl_s7;
        labelsOut[7] = lbl_s8;
        labelsOut[8] = lbl_s9;
        labelsOut[9] = lbl_s10;
        labelsOut[10] = lbl_s11;
        labelsOut[11] = lbl_s12;

        //----INIATILZE INPUT LABEL ARRAY 'labelsI'
        labelsI[0] = lbl_i0;
        labelsI[1] = lbl_i1;
        labelsI[2] = lbl_i2;
        labelsI[3] = lbl_i3;
        labelsI[4] = lbl_i4;
        labelsI[5] = lbl_i5;
        labelsI[6] = lbl_i6;
        labelsI[7] = lbl_i7;
        labelsI[8] = lbl_i8;
```

```

        labelsI[9] = lbl_i9;
        labelsI[10] = lbl_i10;
        labelsI[11] = lbl_i11;
    }
//=====CLEAR INPUT ARRAY=====//
    public void clearArrayA() {
        pnl_order.setVisible(false);
        lbl_aSize.setVisible(false);

        lbl_n1.setVisible(false);
        lbl_n2.setVisible(false);
        lbl_n3.setVisible(false);
        lbl_n4.setVisible(false);
        lbl_n5.setVisible(false);
        lbl_n6.setVisible(false);
        lbl_n7.setVisible(false);
        lbl_n8.setVisible(false);
        lbl_n9.setVisible(false);
        lbl_n10.setVisible(false);
        lbl_n11.setVisible(false);
        lbl_n12.setVisible(false);

        btn_asc.setEnabled(true);
        btn_dsc.setEnabled(true);
    }

//=====CLEAR SORTED ARRAY=====//
    public void clearSortingA() {
        pnl_sorting.setVisible(false);

        lbl_s1.setVisible(false);
        lbl_s2.setVisible(false);
        lbl_s3.setVisible(false);
        lbl_s4.setVisible(false);
        lbl_s5.setVisible(false);
        lbl_s6.setVisible(false);
        lbl_s7.setVisible(false);
        lbl_s8.setVisible(false);
        lbl_s9.setVisible(false);
        lbl_s10.setVisible(false);
        lbl_s11.setVisible(false);
        lbl_s12.setVisible(false);

        lbl_i0.setVisible(false);
        lbl_i1.setVisible(false);
        lbl_i2.setVisible(false);
        lbl_i3.setVisible(false);
        lbl_i4.setVisible(false);
        lbl_i5.setVisible(false);
        lbl_i6.setVisible(false);
        lbl_i7.setVisible(false);
        lbl_i8.setVisible(false);
        lbl_i9.setVisible(false);
        lbl_i10.setVisible(false);
        lbl_i11.setVisible(false);

        clearDots();
        lbl_finish.setVisible(false);
        lbl_finishI.setVisible(false);
        lbl_arrow.setVisible(false);
        lbl_swap.setVisible(false);
        lbl_tf.setVisible(false);
        pnl_show.setVisible(false);
    }

//=====CLEAR PROGRASS DOTS=====//
    public void clearDots() {
        lbl_finish1.setVisible(false);
        lbl_finish2.setVisible(false);
        lbl_finish3.setVisible(false);
        lbl_finish4.setVisible(false);
        lbl_finish5.setVisible(false);
    }

```

```

//=====SORTING THE ARRAY=====//

public void sortingArray(int a, int b){
    int temp;
    try{
        lbl_line4.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM LINE 4
        lbl_algoSign.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM LINE 4

        lbl_tf.setBackground(new java.awt.Color(0,0,102));
        lbl_tf.setText("TRUE"); //DISPLAY WHETHER THE 5TH LINE IS TRUE OR FLASE
        lbl_tf.setVisible(true);
        sortThread.sleep(1500);
        lbl_line4.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE INTO PREVIOUS
        COLOR
        lbl_algoSign.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE INTO
        PREVIOUS COLOR
        lbl_tf.setBackground(new java.awt.Color(21,101,192));

        lbl_swap.setVisible(true); //HIGHLIGHT THE RELATED ALOGITHM LINE 5,6,7
        lbl_line5.setBackground(new java.awt.Color(0,0,102));
        lbl_line6.setBackground(new java.awt.Color(0,0,102));
        lbl_line7.setBackground(new java.awt.Color(0,0,102));
        sortThread.sleep(1500);

        //--SWAP THE ELEMENTS AND DISPLAY THEM ON LABLES
        temp = array[a];
        array[a] = array [b] ;
        labelsOut[a].setText(labelsOut[b].getText());
        array [b] = temp;
        labelsOut[b].setText(Integer.toString(temp));
    }

    catch(InterruptedException v){
        System.out.println(v);
    }
}

//=====THREAD TO BLINK DOTS=====//
Thread dots = new Thread() {
    public void run() {
        try{
            while(lbl_finish.getText().equals("Array is Still Sorting")){
                //DISPLAY DOT ONE AFTER THE OTHER
                lbl_finish1.setVisible(true);
                dots.sleep(500);
                lbl_finish2.setVisible(true);
                dots.sleep(500);
                lbl_finish3.setVisible(true);
                dots.sleep(500);
                lbl_finish4.setVisible(true);
                dots.sleep(500);
                lbl_finish5.setVisible(true);
                dots.sleep(500);

                clearDots();
                dots.sleep(200);
            }
        }
        catch(InterruptedException v){
            System.out.println(v);
        }
        catch(IllegalThreadStateException v){
            System.out.println("array is still sorting");
        }
    }
};

//=====THREAD TO SORT ELEMENTS=====//
Thread sortThread = new Thread() {
    public void run() {
        try {
            //=====DISPLAY USER INPUT ARRAY ELEMENT IN BELOW ARRAY WHICH IS UDER TO DISPLAY SORTING=====//

```

```

int a = 0;
while(a<12 && a<noOfUserInputs) { //CHECK WHETHER THE ARRAY ELEMNT IS NOT 0
    for (int j=0; j<labelsOut.length; j++) {
        if (!labelsOut[j].isVisible()) { //CHECK WHETHER THE LABEL IS VISIBLE
            labelsI[j].setVisible(true);
            labelsOut[j].setVisible(true); //IF LABEL IS NOT VISIBLE MAKE IT VISIBLE
            labelsOut[j].setText(labelsIn[j].getText()); //SET USER INPUT VALUE TO THAT
        }
        break;
    }
    a++;
    sortThread.sleep(200);
}
sortThread.sleep(1500);

//=====START OF SORTING=====//
//int temp;
lbl_line1.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM LINE 1
sortThread.sleep(1000);
lbl_line1.setBackground(new java.awt.Color(21,101,192));

for (int pass = 1; pass < noOfUserInputs; pass++) {
    lbl_finish.setVisible(true); //MALE ARRAY PROGRESS LABEL VISIBLE
    lbl_finish.setText("Array is Still Sorting"); //SHOW MESSAGE THAT ARRAY IS STILL SORTING
    if(!dots.isAlive()) //STARTING POINT OF THE dots THREAD
        dots.start();

    lbl_line2.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM LINE
    sortThread.sleep(1300);
    lbl_line2.setBackground(new java.awt.Color(21,101,192));

    int current = 50; //VARIABLE TO KEEP TRACK OF CURRENT POISTION OF THE GREEN POINTER AND
    int next = 60; //DEFINES X DIRECTION ON EACH STEP

    lbl_arrow.setBounds(50, 90, 90, 60); //BRING GREEN ARROW TO THE BEGINING
    lbl_swap.setBounds(50, 140, 90, 35); //BRING SWAP MESSAGE TO THE BEGINING

    for (int i = 0; i < noOfUserInputs-pass; i++) {
        lbl_line3.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED
        lbl_arrow.setVisible(true); //MAKE GREEN POINTER VISIBLE
        sortThread.sleep(1300);
        lbl_line3.setBackground(new java.awt.Color(21,101,192));

        //MAKE SWAP PANEL DISPALY AND DISPALY RELEVANT VALUES
        pnl_show.setVisible(true);
        lbl_i.setText(labelsOut[i].getText());
        lbl_ii.setText(labelsOut[i+1].getText());

        //--SORT ARRAY IN ASCENDING ORDER-----//
        if (array[i] > array[i+1] && choiceAscen == 1) {
            sortingArray(i, i+1);
        }

        //--SORT ARRAY IN DESCENDING ORDER-----//
        else if (array[i] < array[i+1] && choiceDescen == 1) {
            sortingArray(i, i+1);
        }

        //--IF CONDITON/4th LINE IS FLASE DO THIS
        else {
            lbl_line4.setBackground(new java.awt.Color(0,0,102));
            lbl_algoSign.setBackground(new java.awt.Color(0,0,102));
            lbl_tf.setBackground(new java.awt.Color(0,0,102));
            lbl_tf.setText("FALSE");
            lbl_tf.setVisible(true);
            sortThread.sleep(1000);
            lbl_tf.setBackground(new java.awt.Color(21,101,192));
            lbl_line4.setBackground(new java.awt.Color(21,101,192));
            lbl_algoSign.setBackground(new java.awt.Color(21,101,192));
        }

        sortThread.sleep(1500);
        pnl_show.setVisible(false);

```



```

        lbl_swap.setVisible(false);

        //MOVING GREEN ARROW AND SWAP MESSAGE TO RIGHT
        if(i< noOfUserInputs-pass-1) {
            AC.jLabelXRight(current, current+next, 5, 2, lbl_arrow);
            AC.jLabelXRight(current, current+next, 5, 2, lbl_swap);
            current = current+next;
        }
        lbl_tf.setVisible(false);
        lbl_line5.setBackground(new java.awt.Color(21,101,192));           //CHANGE THE ALGORITHM
COLORS INTO PREVIOUS COLORS
        lbl_line6.setBackground(new java.awt.Color(21,101,192));
        lbl_line7.setBackground(new java.awt.Color(21,101,192));
        //END OF INNER LOOP
    }

    labelsOut[noOfUserInputs-pass].setBackground(new java.awt.Color(204,0,51));    //COLOR THE
LAST SORTED ELEMENT
    lbl_arrow.setVisible(false);
    sortThread.sleep(1500);
    //END OF OUTER LOOP
}

//-----THINGS TO DO WHEN SORTING IS COMPLETED
lbl_arrow.setVisible(false);           //MAKE GREEN ARROW INVISIBLE
lbl_swap.setVisible(false);           //MAKE GREEN ARROW INVISIBLE
labelsOut[0].setBackground(new java.awt.Color(204,0,51));

lbl_finish.setText("Array is SORTED");           //NOTIFY THAT ARRAY SORTING IS FINISH
lbl_finish.setForeground(new java.awt.Color(0,0,48));
Font f = lbl_finish.getFont();           //MAKE LABEL FONT TO BOLD
lbl_finish.setFont(f.deriveFont(f.getStyle() | Font.BOLD));
lbl_finishI.setVisible(true);

while(dots.isAlive())           //IF dots THREAD IS STILL ALIVE MAKE DOTS INVISIBLE
    clearDots();
}

catch(InterruptedException v){
    System.out.println(v);
}
catch(IllegalThreadStateException v){
    System.out.println("array is still sorting");
}
}
};

private void btn_createMouseEntered(java.awt.event.MouseEvent evt) {
    btn_create.setBackground(new java.awt.Color(5, 150, 160));
}

private void btn_createMouseExited(java.awt.event.MouseEvent evt) {
    btn_create.setBackground(new java.awt.Color(12, 173, 183));
}

private void btn_createActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String inNums = txt_inputs.getText();           //GET USER USER INPUT TO A VARIABLE

        String num;           //VARIABLE TO STORE ONE ELEMENT AT A TIME
        int i=0;           //VARIABLE TO KEEP TRACK OF ARRAY INDEX

        StringTokenizer token = new StringTokenizer(inNums, " ");

        if(inNums.matches("[\\d+\\s]+")) {           //VALIDATE USER INPUT
            pnl_array.setVisible(true);
            clearArrayA();           //CLEAR IF THERE IS ARRAY

            //===== display array size=====//
            int arraySize = token.countTokens();
            lbl_aSize.setText("Array Size = " + arraySize);
            lbl_aSize.setVisible(true);

            //=====set elements in array lables=====//
            while (token.hasMoreTokens()) {
                num = token.nextToken();           //GET VALUE AT A TIME
                array[i] = Integer.parseInt(num);           //CONVERT THAT VALUE TO INTEGER
            }
        }
    }
}

```

```

        //--SET VALUES TO LABELS
        for (int j=0; j<labelsIn.length; j++) {
            if (!labelsIn[j].isVisible()) {
                labelsIn[j].setVisible(true);
                labelsIn[j].setText(num);
                break;
            }
        }
        i++;
    }
    }

    noOfUserInputs = arraySize;
    pnl_order.setVisible(true);
    btn_asc.setEnabled(true);
    btn_dsc.setEnabled(true);
}

else{
    new Warning().setVisible(true);
    txt_inputs.setText("");
}

}
catch(NumberFormatException e){
    new Warning().setVisible(true);
    txt_inputs.setText("");
    clearArrayA();
}
//====to print array in output // error checking tool
for (int j = 0; j < array.length; j++) {
    System.out.print(array[j] + " ");
}
System.out.print("\n oOfUserInputs = " + noOfUserInputs);
}

private void btn_resetMouseEntered(java.awt.event.MouseEvent evt) {
    btn_reset.setBackground(new java.awt.Color(5, 150, 160));
}

private void btn_resetMouseExited(java.awt.event.MouseEvent evt) {
    btn_reset.setBackground(new java.awt.Color(12, 173, 183));
}

private void btn_resetActionPerformed(java.awt.event.ActionEvent evt) {
    if(!sortThread.isAlive()){
        txt_inputs.setText("");
        clearArrayA();
        clearSortingA();
    }
}

private void jLabel14MouseClicked(java.awt.event.MouseEvent evt) {
    System.exit(0);
}

private void btn_ascActionPerformed(java.awt.event.ActionEvent evt) {

    UIManager.getDefaults().put("Button.disabledText",Color.LIGHT_GRAY);
    btn_dsc.setEnabled(false);
    btn_reset.setEnabled(false);
    btn_create.setEnabled(false);
    btn_random.setEnabled(false);

    pnl_sorting.setVisible(true);

    choiceAscen = 1;
    choiceDescen = 0;
    lbl_algoSign.setText(">");
    lbl_sign.setText(">");

    sortThread.start();
    System.out.println("\n Main Thread exit");
}

private void btn_dscActionPerformed(java.awt.event.ActionEvent evt) {
    UIManager.getDefaults().put("Button.disabledText",Color.LIGHT_GRAY);
    btn_asc.setEnabled(false);
    btn_reset.setEnabled(false);
}

```

```

        btn_create.setEnabled(false);
        btn_random.setEnabled(false);

        pnl_sorting.setVisible(true);

        choiceDescen = 1;
        choiceAscen = 0;
        lbl_algoSign.setText("<");
        lbl_sign.setText("<");
        sortThread.start(); //START THE THREAD

        System.out.println("\n Main Thread exit");
    }

    private void jPanel18MouseClicked(java.awt.event.MouseEvent evt) {
        System.exit(0);
    }

    private void jLabel15MouseClicked(java.awt.event.MouseEvent evt) {
        new Home().setVisible(true);
        this.dispose();
    }

    private void jPanel19MouseClicked(java.awt.event.MouseEvent evt) {
        new Home().setVisible(true);
        this.dispose();
    }

    private void btn_randomMouseClicked(java.awt.event.MouseEvent evt) {
        btn_random.setBackground(new java.awt.Color(5, 150, 160));
    }

    private void btn_randomMouseExited(java.awt.event.MouseEvent evt) {
        btn_random.setBackground(new java.awt.Color(12, 173, 183));
    }

    private void btn_randomActionPerformed(java.awt.event.ActionEvent evt) {

        pnl_array.setVisible(true);
        clearArrayA(); //CLEAR IF THERE IS ARRAY

        //===== display array size=====//
        int arraySize = 12;
        lbl_aSize.setText("Array Size = " + arraySize);
        lbl_aSize.setVisible(true);

        //--SET VALUES TO LABELS-----//
        for (int j=0; j<12; j++) {
            int random = 0;

            for (int k = 0; k < 12; k++) {
                random = (int) (Math.random()*100); //generate random number;
                if (array[k] != random){
                    break;
                }
            }

            array[j] = random;

            labelsIn[j].setVisible(true); //IF LABEL IS NOT VISIBLE MAKE IT VISIBLE
            labelsIn[j].setText(Integer.toString(random)); //SET USER INPUT VALUE TO THAT VARIABLE
        }

        noOfUserInputs = arraySize; //GET THE NUMBER OF USER INPUT NUMBERS TO VARIABLE

        pnl_order.setVisible(true);
        btn_asc.setEnabled(true);
        btn_dsc.setEnabled(true);
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new BubbleSort().setVisible(true);
            }
        });
    }
}

```

INSERTION SORT

```
package daa;

import AppPackage.AnimationClass;
import java.awt.Color;
import java.awt.Font;
import java.util.StringTokenizer;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.UIManager;
import java.lang.*;

public class InsertionSort extends javax.swing.JFrame {
    JLabel[] labelsIn = new JLabel[12]; //CREATE AN ARRAY OF LABELS TO STORE USER INPUT VALUES
    JLabel[] labelsOut = new JLabel[12]; //CREATE AN ARRAY OF LABELS TO STORE SORTED ARRAY VALUES
    JLabel[] labelsI = new JLabel[12]; //CREATE AN ARRAY OF LABELS TO STORE i VALUES
    int array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //INT ARRAY TO STORE USER INPUTS
    int noOfUserInputs = 0; //VARIABLE TO STORE NUMBER OF USER INPUT NUMBERS AT A TIME
    AnimationClass AC = new AnimationClass(); //MAKE A OBJECT FROM 'AnimationClass' CLASS

    //VARIABLES TO STORE USER SELECTED CHOICE WHETHER ASCENDING OR DESCENDING
    int choiceAscen = 0;
    int choiceDescen = 0;

    public InsertionSort() {
        initComponents();

        pnl_array.setVisible(false);
        clearArrayA();
        clearSortingA();

        //----INITIALIZE INPUT LABEL ARRAY 'labelsIn'
        labelsIn[0] = lbl_n1;
        labelsIn[1] = lbl_n2;
        labelsIn[2] = lbl_n3;
        labelsIn[3] = lbl_n4;
        labelsIn[4] = lbl_n5;
        labelsIn[5] = lbl_n6;
        labelsIn[6] = lbl_n7;
        labelsIn[7] = lbl_n8;
        labelsIn[8] = lbl_n9;
        labelsIn[9] = lbl_n10;
        labelsIn[10] = lbl_n11;
        labelsIn[11] = lbl_n12;

        //----INITIALIZE INPUT LABEL ARRAY 'labelsOut'
        labelsOut[0] = lbl_s1;
        labelsOut[1] = lbl_s2;
        labelsOut[2] = lbl_s3;
        labelsOut[3] = lbl_s4;
        labelsOut[4] = lbl_s5;
        labelsOut[5] = lbl_s6;
        labelsOut[6] = lbl_s7;
        labelsOut[7] = lbl_s8;
        labelsOut[8] = lbl_s9;
        labelsOut[9] = lbl_s10;
        labelsOut[10] = lbl_s11;
        labelsOut[11] = lbl_s12;

        //----INITIALIZE INPUT LABEL ARRAY 'labelsI'
        labelsI[0] = lbl_i0;
        labelsI[1] = lbl_i1;
        labelsI[2] = lbl_i2;
        labelsI[3] = lbl_i3;
        labelsI[4] = lbl_i4;
        labelsI[5] = lbl_i5;
        labelsI[6] = lbl_i6;
        labelsI[7] = lbl_i7;
        labelsI[8] = lbl_i8;
        labelsI[9] = lbl_i9;
        labelsI[10] = lbl_i10;
        labelsI[11] = lbl_i11;
    }
}
```

```
//=====CLEAR INPUT ARRAY=====//
public void clearArrayA() {
    pnl_order.setVisible(false);
    lbl_aSize.setVisible(false);

    lbl_n1.setVisible(false);
    lbl_n2.setVisible(false);
    lbl_n3.setVisible(false);
    lbl_n4.setVisible(false);
    lbl_n5.setVisible(false);
    lbl_n6.setVisible(false);
    lbl_n7.setVisible(false);
    lbl_n8.setVisible(false);
    lbl_n9.setVisible(false);
    lbl_n10.setVisible(false);
    lbl_n11.setVisible(false);
    lbl_n12.setVisible(false);

    btn_asc.setEnabled(true);
    btn_dsc.setEnabled(true);
}

//=====CLEAR SORTED ARRAY=====//
public void clearSortingA() {
    pnl_sorting.setVisible(false);

    lbl_s1.setVisible(false);
    lbl_s2.setVisible(false);
    lbl_s3.setVisible(false);
    lbl_s4.setVisible(false);
    lbl_s5.setVisible(false);
    lbl_s6.setVisible(false);
    lbl_s7.setVisible(false);
    lbl_s8.setVisible(false);
    lbl_s9.setVisible(false);
    lbl_s10.setVisible(false);
    lbl_s11.setVisible(false);
    lbl_s12.setVisible(false);

    lbl_i0.setVisible(false);
    lbl_i1.setVisible(false);
    lbl_i2.setVisible(false);
    lbl_i3.setVisible(false);
    lbl_i4.setVisible(false);
    lbl_i5.setVisible(false);
    lbl_i6.setVisible(false);
    lbl_i7.setVisible(false);
    lbl_i8.setVisible(false);
    lbl_i9.setVisible(false);
    lbl_i10.setVisible(false);
    lbl_i11.setVisible(false);

    clearDots();
    lbl_finish.setVisible(false);
    lbl_finishI.setVisible(false);
    lbl_arrow.setVisible(false);
    lbl_k.setVisible(false);
    lbl_key.setVisible(false);
    lbl_keyAssign.setVisible(false);
    lbl_tf.setVisible(false);
    pnl_show.setVisible(false);
}

//=====CLEAR PROGRASS DOTS=====//
public void clearDots() {
    lbl_finish1.setVisible(false);
    lbl_finish2.setVisible(false);
    lbl_finish3.setVisible(false);
    lbl_finish4.setVisible(false);
    lbl_finish5.setVisible(false);
}

//=====THREAD TO BLINK DOTS=====//
Thread dots = new Thread() {
    public void run() {
        try{
            while(lbl_finish.getText().equals("Array is Still Sorting")){ //DISPLAY DOT ONE AFTER THE
OTHER
                lbl_finish1.setVisible(true);

```

```

        dots.sleep(500);
        lbl_finish2.setVisible(true);
        dots.sleep(500);
        lbl_finish3.setVisible(true);
        dots.sleep(500);
        lbl_finish4.setVisible(true);
        dots.sleep(500);
        lbl_finish5.setVisible(true);
        dots.sleep(500);

        clearDots();
        dots.sleep(200);
    }
}
catch(InterruptedException v){
    System.out.println(v);
}
catch(IllegalThreadStateException v){
    System.out.println("array is still sorting");
}
}
};

//=====THREAD TO SORT ELEMENTS IN ASCENDING ORDER=====//
Thread sortThreadAscen = new Thread() {
    public void run() {
        try {
            //=====DISPLAY USER INPUT ARRAY ELEMENT IN BELOW ARRAY WHICH IS UDER TO DISPLAY SORTING=====//
            int a = 0;
            while( a<12 && a<noOfUserInputs) {                //CHECK WHETHER THE ARRAY ELEMNT IS NOT 0
                for (int j=0; j<labelsOut.length; j++) {
                    if (!labelsOut[j].isVisible()) {            //CHECK WHETHER THE LABEL IS VISIBLE
                        labelsI[j].setVisible(true);
                        labelsOut[j].setVisible(true);            //IF LABEL IS NOT VISIBLE MAKE IT VISIBLE
                        labelsOut[j].setText(labelsIn[j].getText()); //SET USER INPUT VALUE TO THAT
                        break;
                    }
                }
                a++;
                sortThreadAscen.sleep(200);
            }
            sortThreadAscen.sleep(1500);

            //=====START OF SORTING=====//

            int keyStart = 90;    //VARIABLE TO SOTRE STARTING POINT OF THE KEY LABEL AND KEY ASSIGN ARROW
            int arrowStart = 50;    //VARIABLE TO SOTRE STARTING POINT OF THE GREEN ARROW
            int next = 60;        //DEFINES X DIRECTION ON EACH STEP FOR GREEN ARROWS AND KEY LABEL

            for (int j=1; j<noOfUserInputs; j++) {
                lbl_finish.setVisible(true);                //MALE ARRAY PROGRESS LABEL VISIBLE
                lbl_finish.setText("Array is Still Sorting"); //SHOW MESSAGE THAT ARRAY IS STILL SORTING
                if(!dots.isAlive())                        //STARTING POINT OF THE dots THREAD
                    dots.start();

                //COLOR THE LAST SORTED ELEMENTS TO UNSORTED ELEMENTS' COLOR
                for (int k = 0; k <= j; k++) {
                    labelsOut[k].setBackground(new java.awt.Color(0,0,40));
                }

                //--BRING THE GREEN ARROW AND KEY LABEL TO THEIR STARTING POSITIONS
                lbl_key.setBounds(keyStart, 140, 55, 50);
                lbl_keyAssign.setBounds(keyStart, 85, 55, 50);
                lbl_arrow.setBounds(arrowStart, 80, 90, 50);

                lbl_line1.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED
                sortThreadAscen.sleep(2000);
                lbl_line1.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE
                INTO PREVIOUS COLOR

                //--DISPLAY KEY VALUE
                lbl_line2.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM
                LINE 2
                int key = array[j]; //GET THE KEY VALUE TO A VARIABLE
                lbl_key.setText(Integer.toString(key)); //ASSIGN KEY VALUE TO KEY VALUE LABEL
            }
        }
    }
};

```

```

        labelsOut[j].setBackground(new java.awt.Color(255,65,129));        //SHOW THE ELEMENT THAT
IS GOING TO BE THE KEY
        sortThreadAscen.sleep(1000);
        lbl_key.setVisible(true);        //MAKE KEY LABEL TO VISIBLE
        lbl_k.setVisible(true);
        labelsOut[j].setBackground(new java.awt.Color(0,0,40));        //REPAINT THE ELEMENT THAT IS
GOING TO BE THE KEY TO PREVIOUS COLOUR
        sortThreadAscen.sleep(1000);
        lbl_line2.setBackground(new java.awt.Color(21,101,192));        //CHANGE THE COLOR OF LINE INTO
PREVIOUS COLOR

        int i = j-1;
        lbl_line3.setBackground(new java.awt.Color(0,0,102));        //HIGHLIGHT THE RELATED ALOGITHM
LINE 3
        sortThreadAscen.sleep(1500);
        lbl_line3.setBackground(new java.awt.Color(21,101,192));        //CHANGE THE COLOR OF LINE INTO
PREVIOUS COLOR

        int keyGoFrom = keyStart;        //VARIABLE TO SOTRE STARTING POINT INSIDE WHILE OF THE
KEY LABEL AND KEY ASSIGN ARROW
        int arrowGoFrom = arrowStart;        //VARIABLE TO SOTRE STARTING POINT INSIDE WHILE OF THE
GREEN ARROW

        while( i>=0 && array[i] > key ) {
            lbl_line4.setBackground(new java.awt.Color(0,0,102));        //HIGHLIGHT THE RELATED
ALOGITHM LINE 4
            lbl_algoSign.setBackground(new java.awt.Color(0,0,102));
            lbl_tf.setBackground(new java.awt.Color(0,0,102));
            lbl_tf.setText("TRUE");        //DISPLAY WHETHER THE 4TH LINE IS TRUE OR FLASE
            lbl_tf.setVisible(true);

            //--MAKE SWAP PANEL VISIBLE AND DISPALY RELEVANT VALUES
            pnl_show.setVisible(true);
            lbl_iIndex.setText(Integer.toString(i));        //DISPLAY i TH INDEX
            lbl_iValue.setText(labelsOut[i].getText());        //DISPLAY array[i] VALUE
            lbl_keyValue.setText(Integer.toString(key));        //DISPLAY KEY VALUE
            lbl_ii.setText(Integer.toString(i+1));        //DISPLAY i+1 TH INDEX
            lbl_i.setText(Integer.toString(i));        //DISPLAY i TH INDEX
            sortThreadAscen.sleep(1800);

            lbl_line4.setBackground(new java.awt.Color(21,101,192));        //CHANGE THE COLOR OF
LINE INTO PREVIOUS COLOR
            lbl_tf.setBackground(new java.awt.Color(21,101,192));
            lbl_algoSign.setBackground(new java.awt.Color(21,101,192));

            lbl_line5.setBackground(new java.awt.Color(0,0,102));        //HIGHLIGHT THE RELATED
ALOGITHM LINE 5
            lbl_arrow.setVisible(true);        //MAKE GREEN ARROW VISIBLE
            sortThreadAscen.sleep(1000);
            array[i+1] = array[i];        //ASSIGN i INDEX VALUE TO i+1 INDEX VALUE
            labelsOut[i+1].setText(labelsOut[i].getText());
            sortThreadAscen.sleep(1500);
            lbl_line5.setBackground(new java.awt.Color(21,101,192));        //CHANGE THE COLOR OF
LINE INTO PREVIOUS COLOR
            lbl_arrow.setVisible(false);

            lbl_line6.setBackground(new java.awt.Color(0,0,102));        //HIGHLIGHT THE RELATED
ALOGITHM LINE 6
            sortThreadAscen.sleep(1300);
            lbl_line6.setBackground(new java.awt.Color(21,101,192));        //CHANGE THE COLOR OF LINE
INTO PREVIOUS COLOR

            lbl_tf.setVisible(false);

            i=i-1;        //DECREASE i VALUE
            //MOVE KEY LABEL AND KEY ASSIGN ARROW
            AC.jLabelXLeft(keyGoFrom, keyGoFrom-next, 5, 2, lbl_key);
            AC.jLabelXLeft(keyGoFrom, keyGoFrom-next, 5, 2, lbl_keyAssign);
            sortThreadAscen.sleep(2000);

            if (i>=0)        //MOVE GREEN ARROW
                AC.jLabelXLeft(arrowGoFrom, arrowGoFrom-next, 5, 2, lbl_arrow);

            //DECIDE KEY LABEL'S AND GREEN ARROW'S NEXT POSITION
            keyGoFrom = keyGoFrom-next;
            arrowGoFrom = arrowGoFrom-next;
            //END OF WHILE LOOP
        }

```

```

//--WHAT TO DO IF THE WHILE CONDITION IS FALSE
LINE 4    lbl_line4.setBackground(new java.awt.Color(0,0,102));    //HIGHLIGHT THE RELATED ALGORITHM

    lbl_algoSign.setBackground(new java.awt.Color(0,0,102));
    lbl_tf.setBackground(new java.awt.Color(0,0,102));
    lbl_tf.setText("FALSE");    //DISPLAY WHETHER THE 4TH LINE IS TRUE OR FLASE
    lbl_tf.setVisible(true);

    //--MAKE SWAP PANEL VISIBLE AND DISPALY RELEVANT VALUES
    pnl_show.setVisible(true);
    lbl_iIndex.setText(Integer.toString(i));    //DISPLAY i TH INDEX
    lbl_iValue.setText(labelsOut[i+1].getText());    //DISPLAY array[i] VALUE
    lbl_keyValue.setText(Integer.toString(key));    //DISPLAY KEY VALUE
    lbl_ii.setText(Integer.toString(i+1));    //DISPLAY i+1 TH INDEX
    lbl_i.setText(Integer.toString(i));    //DISPLAY i TH INDEX
    sortThreadAscen.sleep(1800);

    lbl_line4.setBackground(new java.awt.Color(21,101,192));    //CHANGE THE COLOR OF LINE
    INTO PREVIOUS COLOR
    lbl_tf.setBackground(new java.awt.Color(21,101,192));
    lbl_algoSign.setBackground(new java.awt.Color(21,101,192));

    //--ASSIGN KEY TO i+1 TH INDEX
    lbl_keyAssign.setVisible(true);    //MALE KEY ASSIGN ARROW VALUE VISIBLE
    array[i+1] = key;
    sortThreadAscen.sleep(1000);
    labelsOut[i+1].setText(Integer.toString(key));
    sortThreadAscen.sleep(1800);

    lbl_keyAssign.setVisible(false);    //DISAPERE KEY VALUE LABEL
    lbl_key.setVisible(false);
    lbl_k.setVisible(false);
    lbl_tf.setVisible(false);
    pnl_show.setVisible(false);    //MAKE pnl_show PANEL INVISIBLE

    keyStart = keyStart + next;    //DECIDE FROM WHERE KEY LABEL START FROM NEXT for LOOP
    arrowStart = arrowStart + next;    //DECIDE FROM WHERE GREEN ARROW START FROM NEXT for LOOP

    //COLOR THE LAST SORTED ELEMENTS
    for (int k = 0; k <= j; k++) {
        labelsOut[k].setBackground(new java.awt.Color(204,0,51));
    }
    sortThreadAscen.sleep(1500);

    //END OF OUTER FOR LOOP
}

//-----THINGS TO DO WHEN SOTRING IS COMPLETED-----//
    lbl_finish.setText("Array is SORTED");    //NOTIFY THAT ARRAY SORTING IS FINISH
    lbl_finish.setForeground(new java.awt.Color(0,0,48));
    Font f = lbl_finish.getFont();    //MAKE LABEL FONT TO BOLD
    lbl_finish.setFont(f.deriveFont(f.getStyle() | Font.BOLD));
    lbl_finishI.setVisible(true);

    while(dots.isAlive())    //IF dots THREAD IS STILL ALIVE MAKE DOTS INVISIBLE
        clearDots();
}

catch(InterruptedException v){
    System.out.println(v);
}
catch(IllegalThreadStateException v){
    System.out.println("array is still sorting");
}
}
};

//=====THREAD TO SORT ELEMENTS IN DESCENDING ORDER=====//
Thread sortThreadDescen = new Thread() {
    public void run() {
        try {
            //====DISPLAY USER INPUT ARRAY ELEMENT IN BELOW ARRAY WHICH IS UDER TO DISPLAY SORTING====//
            int a = 0;
            while(a<12 && a<noOfUserInputs) {    //CHECK WHETHER THE ARRAY ELEMNT IS NOT 0
                for (int j=0; j<labelsOut.length; j++) {
                    if (!labelsOut[j].isVisible()) {    //CHECK WHETHER THE LABLE IS VISIBLE

```



```

        labelsI[j].setVisible(true);
        labelsOut[j].setVisible(true);           //IF LABEL IS NOT VISIBLE MAKE IT VISIBLE
        labelsOut[j].setText(labelsIn[j].getText()); //SET USER INPUT VALUE TO THAT
VARIABLE
    }
    break;
}
}
a++;
sortThreadDescen.sleep(200);
}
sortThreadDescen.sleep(1500);

//=====START OF SORTING=====//

int keyStart = 90; //VARIABLE TO SOTRE STARTING POINT OF THE KEY LABEL AND KEY ASSIGN ARROW
int arrowStart = 50; //VARIABLE TO SOTRE STARTING POINT OF THE GREEN ARROW
int next = 60; //DEFINES X DIRECTION ON EACH STEP FOR GREEN ARROWS AND KEY LABEL

for (int j=1; j<noOfUserInputs; j++){
    lbl_finish.setVisible(true); //MALE ARRAY PROGRESS LABEL VISIBLE
    lbl_finish.setText("Array is Still Sorting"); //SHOW MESSAGE THAT ARRAY IS STILL SORTING
    if(!dots.isAlive()) //STARTING POINT OF THE dots THREAD
        dots.start();

    //COLOR THE LAST SORTED ELEMENTS TO UNSORTED ELEMENTS' COLOR
    for (int k = 0; k <= j; k++) {
        labelsOut[k].setBackground(new java.awt.Color(0,0,40));
    }

    //--BRING THE GREEN ARROW AND KEY LABEL TO THEIR STARTING POSITIONS
    lbl_key.setBounds(keyStart, 140, 55, 50);
    lbl_keyAssign.setBounds(keyStart, 85, 55, 50);
    lbl_arrow.setBounds(arrowStart, 80, 90, 50);

    lbl_line1.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM
LINE 1
    sortThreadDescen.sleep(2000);
    lbl_line1.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE INTO
PREVIOUS COLOR

    //--DISPLAY KEY VALUE
    lbl_line2.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM
LINE 2
    int key = array[j]; //GET THE KEY VALUE TO A VARIABLE
    lbl_key.setText(Integer.toString(key)); //ASSIGN KEY VALUE TO KEY VALUE LABEL
    labelsOut[j].setBackground(new java.awt.Color(255,65,129)); //SHOW THE ELEMENT THAT IS
GOING TO BE THE KEY
    sortThreadDescen.sleep(1000);
    lbl_key.setVisible(true); //MAKE KEY LABEL TO VISIBLE
    lbl_k.setVisible(true);
    labelsOut[j].setBackground(new java.awt.Color(0,0,40)); //REPAINT THE ELEMENT THAT IS
GOING TO BE THE KEY TO PREVIOUS COLOUR
    sortThreadDescen.sleep(1000);
    lbl_line2.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE INTO
PREVIOUS COLOR

    int i = j-1;
    lbl_line3.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM
LINE 3
    sortThreadDescen.sleep(1500);
    lbl_line3.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE INTO
PREVIOUS COLOR

    int keyGoFrom = keyStart; //VARIABLE TO SOTRE STARTING POINT INSIDE WHILE OF THE KEY
LABEL AND KEY ASSIGN ARROW
    int arrowGoFrom = arrowStart; //VARIABLE TO SOTRE STARTING POINT INSIDE WHILE OF THE
GREEN ARROW

    while( i>=0 && array[i] < key ) {
        lbl_line4.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED
ALOGITHM LINE 4
        lbl_algoSign.setBackground(new java.awt.Color(0,0,102));
        lbl_tf.setBackground(new java.awt.Color(0,0,102));
        lbl_tf.setText("TRUE"); //DISPLAY WHETHER THE 4TH LINE IS TRUE OR FLASE
        lbl_tf.setVisible(true);

        //--MAKE SWAP PANEL VISIBLE AND DISPALY RELEVANT VALUES
        pnl_show.setVisible(true);

```

```

        lbl_iIndex.setText(Integer.toString(i)); //DISPLAY i TH INDEX
        lbl_iValue.setText(labelsOut[i].getText()); //DISPLAY array[i] VALUE
        lbl_keyValue.setText(Integer.toString(key)); //DISPLAY KEY VALUE
        lbl_ii.setText(Integer.toString(i+1)); //DISPLAY i+1 TH INDEX
        lbl_i.setText(Integer.toString(i)); //DISPLAY i TH INDEX
        sortThreadDescen.sleep(1800);

        lbl_line4.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE
        INTO PREVIOUS COLOR

        lbl_tf.setBackground(new java.awt.Color(21,101,192));
        lbl_algoSign.setBackground(new java.awt.Color(21,101,192));

        lbl_line5.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED
        ALOGITHM LINE 5

        lbl_arrow.setVisible(true); //MAKE GREEN ARROW VISIBLE
        sortThreadDescen.sleep(1000);
        array[i+1] = array[i]; //ASSIGN i INDEX VALUE TO i+1 INDEX VALUE
        labelsOut[i+1].setText(labelsOut[i].getText());
        sortThreadDescen.sleep(1500);
        INTO PREVIOUS COLOR
        lbl_line5.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE

        lbl_arrow.setVisible(false);

        lbl_line6.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED
        ALOGITHM LINE 6

        sortThreadDescen.sleep(1300);
        INTO PREVIOUS COLOR
        lbl_line6.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE

        lbl_tf.setVisible(false);

        i=i-1; //DECREASE i VALUE
        //MOVE KEY LABEL AND KEY ASSIGN ARROW
        AC.jLabelXLeft(keyGoFrom, keyGoFrom-next, 5, 2, lbl_key);
        AC.jLabelXLeft(keyGoFrom, keyGoFrom-next, 5, 2, lbl_keyAssign);

        sortThreadDescen.sleep(2000);

        if (i>=0) //MOVE GREEN ARROW
            AC.jLabelXLeft(arrowGoFrom, arrowGoFrom-next, 5, 2, lbl_arrow);

        //DECIDE KEY LABEL'S AND GREEN ARROW'S NEXT POSITION
        keyGoFrom = keyGoFrom-next;
        arrowGoFrom = arrowGoFrom-next;

        //END OF WHILE LOOP
    }

    //--WHAT TO DO IF THE WHILE CONDITION IS FALSE
    lbl_line4.setBackground(new java.awt.Color(0,0,102)); //HIGHLIGHT THE RELATED ALOGITHM
    LINE 4

    lbl_algoSign.setBackground(new java.awt.Color(0,0,102));
    lbl_tf.setBackground(new java.awt.Color(0,0,102));
    lbl_tf.setText("FALSE"); //DISPLAY WHETHER THE 4TH LINE IS TRUE OR FLASE
    lbl_tf.setVisible(true);

    //--MAKE SWAP PANEL VISIBLE AND DISPALY RELEVANT VALUES
    pnl_show.setVisible(true);
    lbl_iIndex.setText(Integer.toString(i)); //DISPLAY i TH INDEX
    lbl_iValue.setText(labelsOut[i+1].getText()); //DISPLAY array[i] VALUE
    lbl_keyValue.setText(Integer.toString(key)); //DISPLAY KEY VALUE
    lbl_ii.setText(Integer.toString(i+1)); //DISPLAY i+1 TH INDEX
    lbl_i.setText(Integer.toString(i)); //DISPLAY i TH INDEX
    sortThreadDescen.sleep(1800);

    lbl_line4.setBackground(new java.awt.Color(21,101,192)); //CHANGE THE COLOR OF LINE INTO
    PREVIOUS COLOR

    lbl_tf.setBackground(new java.awt.Color(21,101,192));
    lbl_algoSign.setBackground(new java.awt.Color(21,101,192));

    //--ASSIGN KEY TO i+1 TH INDEX
    lbl_keyAssign.setVisible(true); //MALE KEY ASSIGN ARROW VALUE VISIBLE
    array[i+1] = key;
    sortThreadDescen.sleep(1000);
    labelsOut[i+1].setText(Integer.toString(key));
    sortThreadDescen.sleep(1800);

    lbl_keyAssign.setVisible(false); //DISAPERE KEY VALUE LABEL
    lbl_key.setVisible(false);

```

```

        lbl_k.setVisible(false);
        lbl_tf.setVisible(false);
        pnl_show.setVisible(false); //MAKE pnl_show PANEL INVISIBLE

        keyStart = keyStart + next; //DECIDE FROM WHERE KEY LABEL START FROM NEXT for LOOP
        arrowStart = arrowStart + next; //DECIDE FROM WHERE GREEN ARROW START FROM NEXT for LOOP

        //COLOR THE LAST SORTED ELEMENTS
        for (int k = 0; k <= j; k++) {
            labelsOut[k].setBackground(new java.awt.Color(204,0,51));
        }
        sortThreadDescen.sleep(1500);

        //END OF OUTER FOR LOOP
    }

//-----THINGS TO DO WHEN SORTING IS COMPLETED

        lbl_finish.setText("Array is SORTED"); //NOTIFY THAT ARRAY SORTING IS FINISH
        lbl_finish.setForeground(new java.awt.Color(0,0,48));
        Font f = lbl_finish.getFont(); //MAKE LABEL FONT TO BOLD
        lbl_finish.setFont(f.deriveFont(f.getStyle() | Font.BOLD));
        lbl_finishI.setVisible(true);

        while(dots.isAlive()) //IF dots THREAD IS STILL ALIVE MAKE DOTS INVISIBLE
            clearDots();
    }

    catch(InterruptedException v){
        System.out.println(v);
    }
    catch(IllegalThreadStateException v){
        System.out.println("array is still sorting");
    }
}

};

private void btn_createMouseEntered(java.awt.event.MouseEvent evt) {
    btn_create.setBackground(new java.awt.Color(5, 150, 160));
}

private void btn_createMouseExited(java.awt.event.MouseEvent evt) {
    btn_create.setBackground(new java.awt.Color(12, 173, 183));
}

private void btn_createActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String inNums = txt_inputs.getText(); //GET USER USER INPUT TO A VARIABLE
        String num; //VARIABLE TO STORE ONE ELEMENT AT A TIME
        int i=0; //VARIABLE TO KEEP TRACK OF ARRAY INDEX
        StringTokenizer token = new StringTokenizer( inNums, " ");

        if(inNums.matches("[\\d+\\s]+")) { //VALIDATE USER INPUT
            pnl_array.setVisible(true);
            clearArrayA(); //CLEAR IF THERE IS ARRAY

            //===== display array size=====//
            int arraySize = token.countTokens();
            lbl_aSize.setText("Array Size = " + arraySize);
            lbl_aSize.setVisible(true);

            //=====set elements in array lables=====//
            while (token.hasMoreTokens()) {
                num = token.nextToken(); //GET VALUE AT A TIME
                array[i] = Integer.parseInt(num); //CONVERT THAT VALUE TO INTEGER

                //--SET VALUES TO LABELS-----//
                for (int j=0; j<labelsIn.length; j++) {
                    if (!labelsIn[j].isVisible()) {
                        labelsIn[j].setVisible(true); //CHECK WHETHER THE LABEL IS VISIBLE
                        labelsIn[j].setText(num); //IF LABEL IS NOT VISIBLE MAKE IT VISIBLE
                        break; //SET USER INPUT VALUE TO THAT VARIABLE
                    }
                }
                i++; //INCREASE ARRAY INDEX
            }
        }
    }
}

```

```

        noOfUserInputs = arraySize;                //GET THE NUMBER OF USER INPUT NUMBERS TO VARIABLE

        pnl_order.setVisible(true);
        btn_asc.setEnabled(true);
        btn_dsc.setEnabled(true);
    }

    else{
        new Warning().setVisible(true);
        txt_inputs.setText("");
    }
}
catch(NumberFormatException e){
    new Warning().setVisible(true);
    txt_inputs.setText("");
    clearArrayA();                                //CLEAR IF THERE IS ARRAY
}

//====to print array in output // error checking tool
for (int j = 0; j < array.length; j++) {
    System.out.print(array[j] + " ");
}
System.out.print("\n oOfUserInputs = " + noOfUserInputs);
}

private void btn_resetMouseEntered(java.awt.event.MouseEvent evt) {
    btn_reset.setBackground(new java.awt.Color(5, 150, 160));
}

private void btn_resetMouseExited(java.awt.event.MouseEvent evt) {
    btn_reset.setBackground(new java.awt.Color(12, 173, 183));
}

private void btn_resetActionPerformed(java.awt.event.ActionEvent evt) {
    if(!sortThreadAscen.isAlive()){
        txt_inputs.setText("");
        clearArrayA();
        clearSortingA();
    }
}

private void jLabel14MouseClicked(java.awt.event.MouseEvent evt) {
    System.exit(0);
}

private void btn_ascActionPerformed(java.awt.event.ActionEvent evt) {

    UIManager.getDefaults().put("Button.disabledText",Color.LIGHT_GRAY);
    btn_dsc.setEnabled(false);
    btn_reset.setEnabled(false);
    btn_create.setEnabled(false);
    btn_random.setEnabled(false);
    pnl_sorting.setVisible(true);

    choiceAscen = 1;
    choiceDescen = 0;
    lbl_algoSign.setText(">");
    lbl_sign.setText(">");

    sortThreadAscen.start();                        //START THE THREAD
}

private void btn_dscActionPerformed(java.awt.event.ActionEvent evt) {
    UIManager.getDefaults().put("Button.disabledText",Color.LIGHT_GRAY);
    btn_asc.setEnabled(false);
    btn_reset.setEnabled(false);
    btn_create.setEnabled(false);
    btn_random.setEnabled(false);
    pnl_sorting.setVisible(true);

    choiceDescen = 1;
    choiceAscen = 0;
    lbl_algoSign.setText("<");
    lbl_sign.setText("<");
    sortThreadDescen.start();                        //START THE THREAD
}
}

```

```

private void jPanel18MouseClicked(java.awt.event.MouseEvent evt) {
    System.exit(0);
}

private void jLabel15MouseClicked(java.awt.event.MouseEvent evt) {
    new Home().setVisible(true);
    this.dispose();
}

private void jPanel19MouseClicked(java.awt.event.MouseEvent evt) {
    new Home().setVisible(true);
    this.dispose();
}

private void btn_randomMouseClicked(java.awt.event.MouseEvent evt) {
    btn_random.setBackground(new java.awt.Color(5, 150, 160));
}

private void btn_randomMouseExited(java.awt.event.MouseEvent evt) {
    btn_random.setBackground(new java.awt.Color(12, 173, 183));
}

private void btn_randomActionPerformed(java.awt.event.ActionEvent evt) {

    pnl_array.setVisible(true);
    clearArrayA(); //CLEAR IF THERE IS ARRAY

    //===== display array size=====//
    int arraySize = 12;
    lbl_aSize.setText("Array Size = " + arraySize);
    lbl_aSize.setVisible(true);

    //--SET VALUES TO LABELS-----//
    for (int j=0; j<12; j++) {
        int random = 0;

        for (int k = 0; k < 12; k++) {
            random = (int) (Math.random()*100); //generate random number;
            if (array[k] != random){
                break;
            }
        }

        array[j] = random;

        labelsIn[j].setVisible(true); //IF LABEL IS NOT VISIBLE MAKE IT VISIBLE
        labelsIn[j].setText(Integer.toString(random)); //SET USER INPUT VALUE TO THAT VARIABLE
        //break;
    }

    noOfUserInputs = arraySize; //GET THE NUMBER OF USER INPUT NUMBERS TO VARIABLE

    pnl_order.setVisible(true);
    btn_asc.setEnabled(true);
    btn_dsc.setEnabled(true);
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new InsertionSort().setVisible(true);
        }
    });
}

```

OTHER INTERFACES

