Sri Lanka Institute of Information Technology

Design and Analysis of Algorithms -214

Assignment 1

Year 2, Semester II - 2015

Algorithm Simulator

For

Bubble Sort

Submitted by:

Registration No: IT15005786

Name: Thilina Loku Bogahawatte.

I hereby certify that this activity is furnished by me to the best of my knowledge.

Signature.

Abstract

Simulator is a numerical calculator with features such as creation of user-defined algorithms. Simulator presents here a new efficient simulation algorithm that is obtained as a modification of bubble sort algorithm.

This simulator program will be able to perform the Bubble sort with step by step execution.

We trace the history of bubble sort, its popularity, and its endurance in the face of pedagogical assertions that code and algorithmic examples used in early courses should be of high quality and adhere to established best practices. This paper is more an historical analysis than a philosophical treatise for the exclusion of bubble sort from books and courses. In short, the bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems." Although bubble sort may not be a best practice sort, perhaps the weight of history is more than enough to compensate and provide for its longevity

Contents

Introduction

Bubble Sort

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort. Although the algorithm is simple, most of the other sorting algorithms are more efficient for large lists

Bubble sort has worst-case and average complexity both O(n2), where n is the number of items being sorted. There exist many sorting algorithms with substantially better worst-case or average complexity of O(n log n). Even other O(n2) sorting algorithms, such as insertion sort, tend to have better performance than bubble sort. Therefore, bubble sort is not a practical sorting algorithm when n is large.

Bubble Sort

The bubble sort algorithm:

1. Compare adjacent elements. If the first is greater than the second, swap them.
2. Do this for each pair of adjacent elements, starting with the first two and ending with the last two. At this point the last element should be the greatest.
3. Repeat the steps for all elements except the last one.
4. Continue for one less element each time, until there are no more pairs to compare.

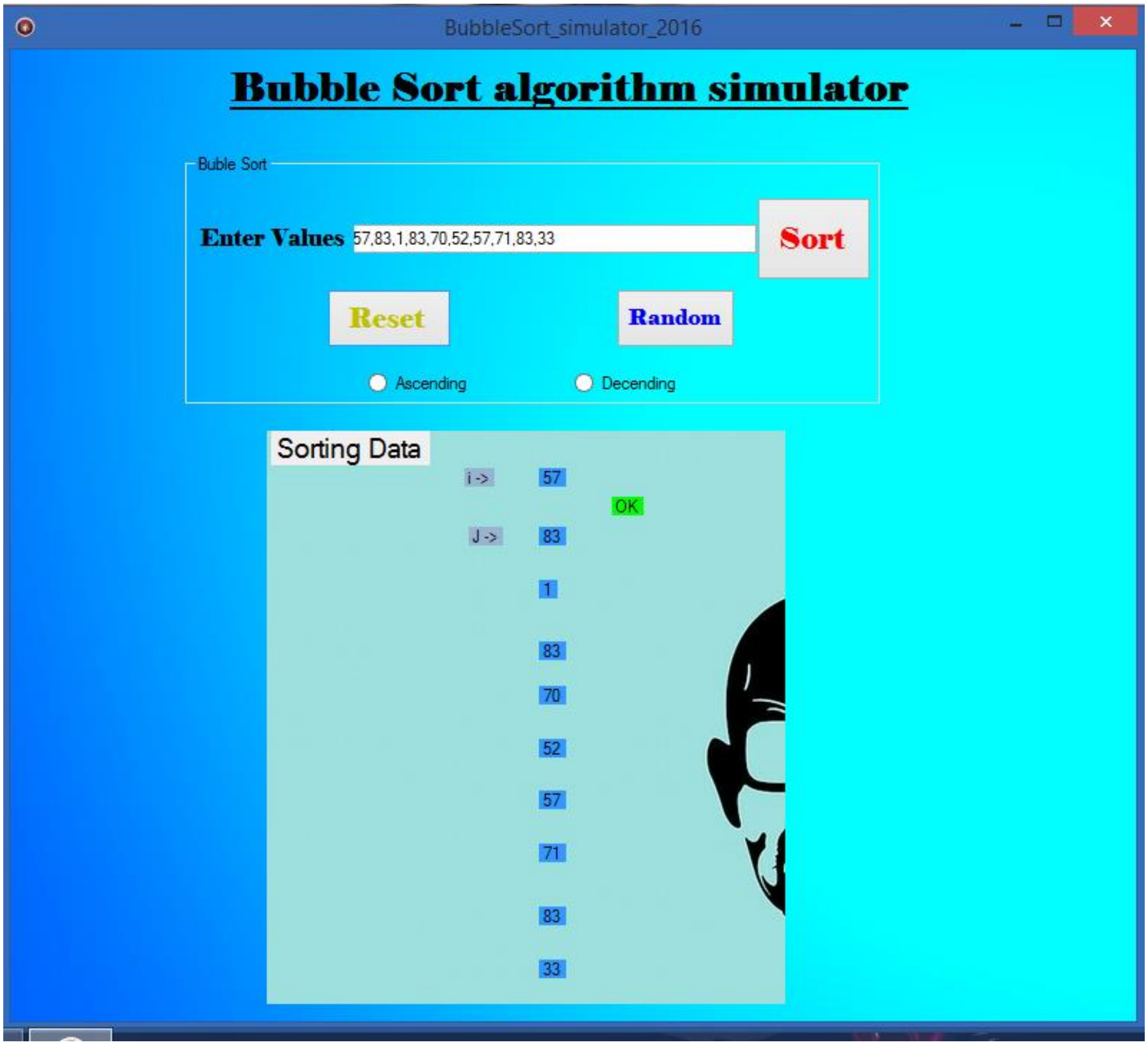
Psudocode for bubble sort

```
for i = 1:n,
    swapped = false
    for j = n:i+1,
        if a[j] < a[j-1],
            swap a[j,j-1]
            swapped = true
    → invariant: a[1..i] in final position
    break if not swapped
end
```

c# code for bubblesort

```
public static int[] SortArray(int[] array)
{
    int length = array.Length;

    int temp = array[0];

    for (int i = 0; i < length; i++)
    {
        for (int j = i+1; j < length; j++)
        {
            if (array[i] > array[j])
            {
                temp = array[i];

                array[i] = array[j];

                array[j] = temp;
            }
        }
    }

    return array;
}
```

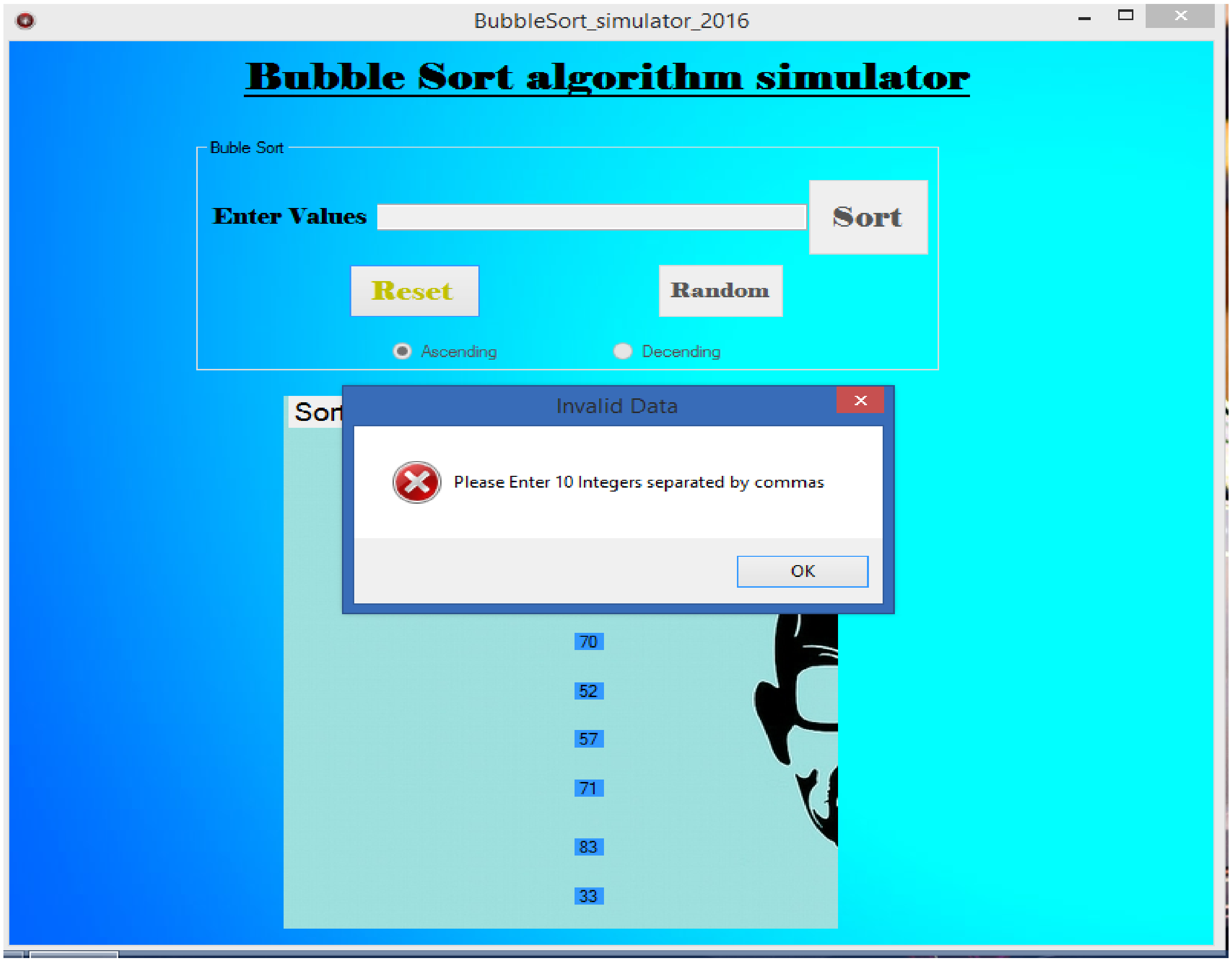
Interface



Enter Values 57,83,1,83,70,52,57,71,83,33

This is the home interface simiulator.to using this interface user can easily select the sorting algorithm and able to add integers separated by comma.

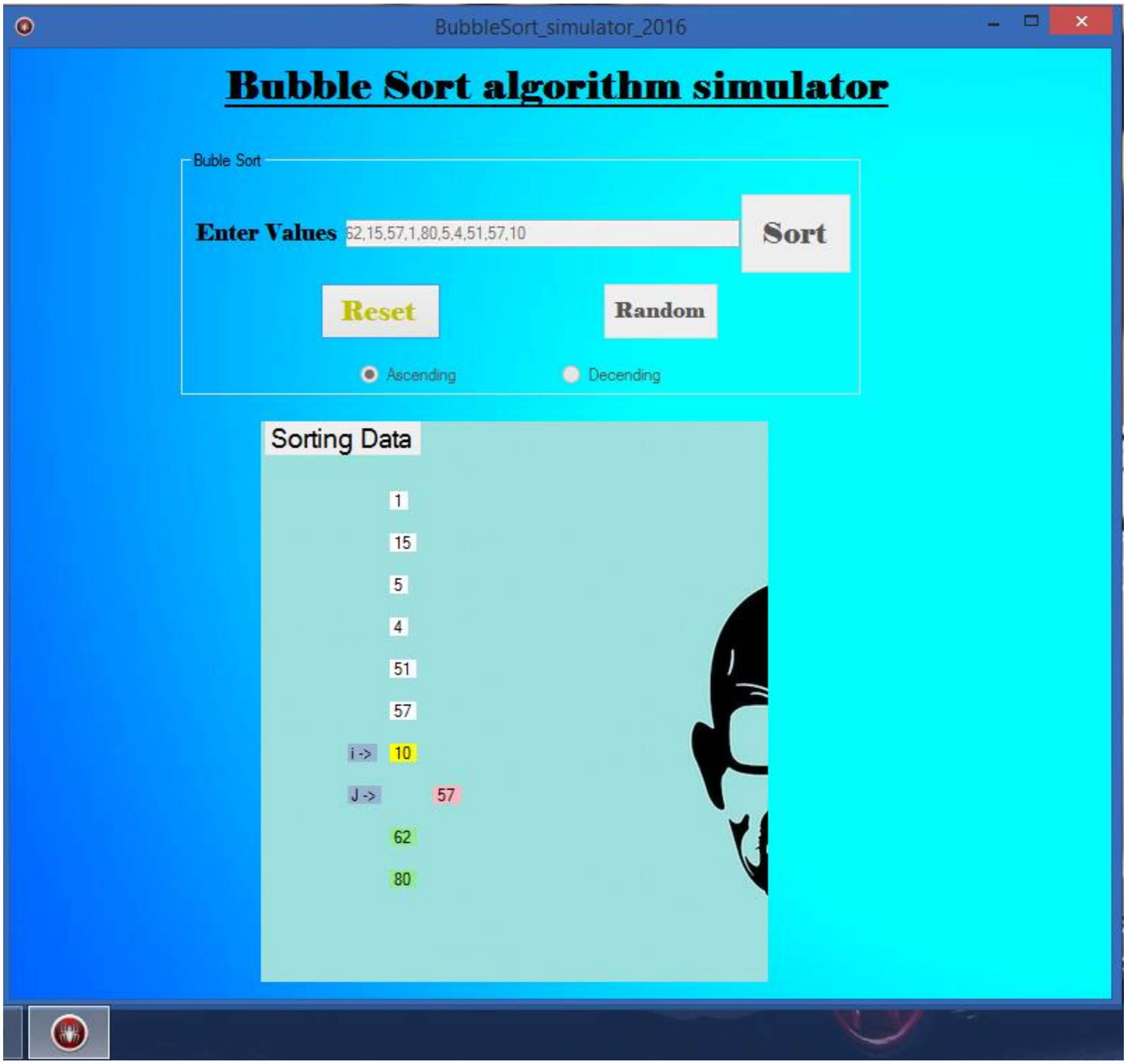
After adding number user can press sort button. Then sorting process will be show in step by animated labels holding input values of user.



● Ascending ○ Decending

Please Enter 10 Integers separated by commas

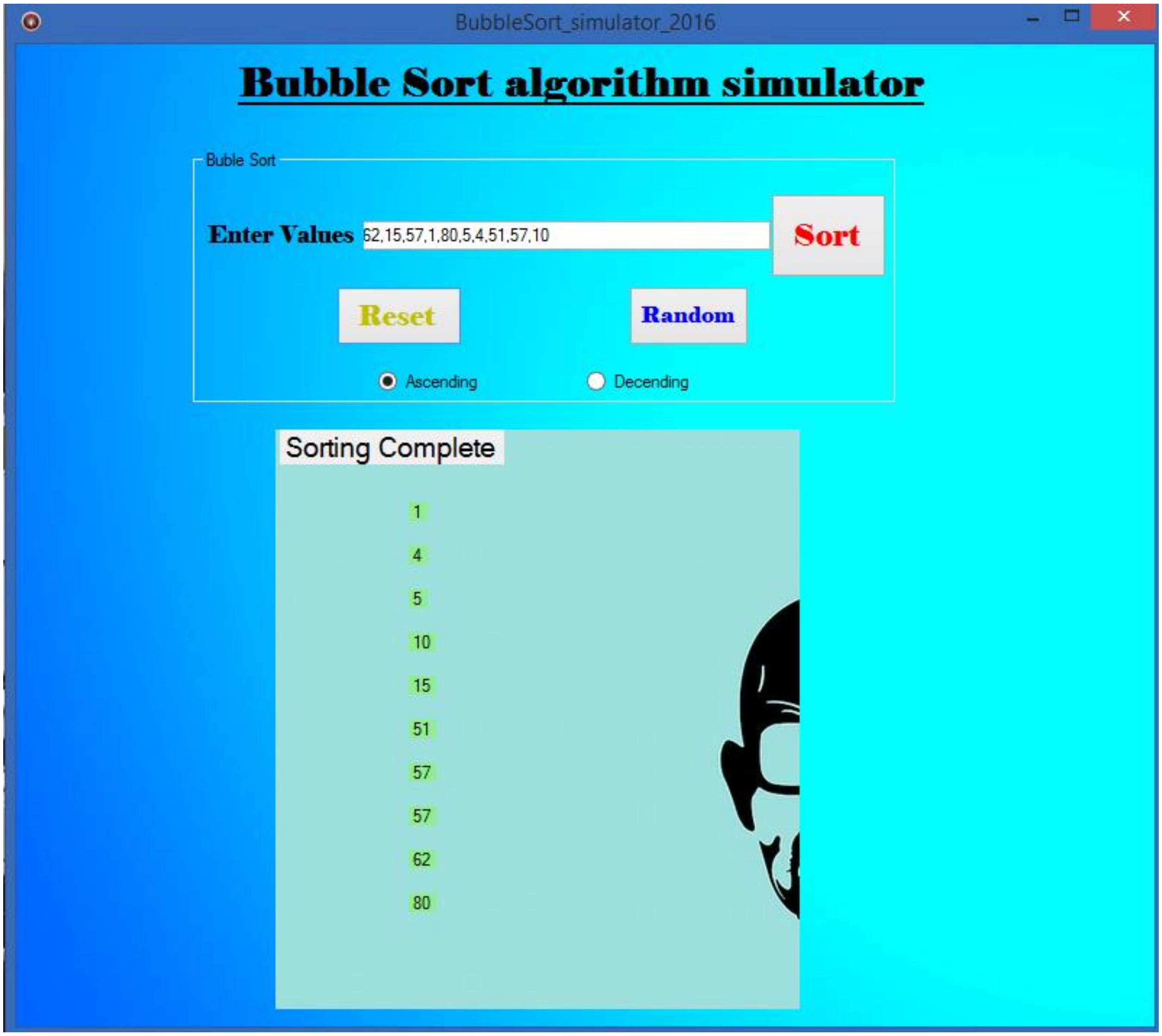
Verify the user entered numbers. In this case numbers can be possitive or negative. If not simulator display an error message.



Enter Values 62,15,57,1,80,5,4,51,57,10

● Ascending ○ Decending

1
15
5
4
51
57
i -> 10
J -> 57
62
80



BubbleSort_simulator_2016

Bubble Sort algorithm simulator

Buble Sort

Enter Values 62,15,57,1,80,5,4,51,57,10

● Ascending ○ Decending

Sorting Complete

```
1
4
5
10
15
51
57
57
62
80
```

After sorting elements sorting complete label will be displayed with the sorted values

Interface Coding

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace T_BAlgorithm
{
    public partial class Form1 : Form
    {
        public Random rnd;

        double[] BdataArray;
        Label[] BmyLabels;
        int Blength = 10;
        int Btimer1count = 0;
        int Btimer2count = 0;
        int BEndingIndex = 9;
        int BcurrentIndex = 0;
        int BupElementInitialX;
        int BupElementInitialY;
        int BdownElementInitialX;
        int BdownElementInitialY;
        int BmovingLength = 0;



        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            rnd = new Random();

            Blength = 10;
            Btimer1count = 0;
            Btimer2count = 0;
            BEndingIndex = Blength - 9; //J
            BcurrentIndex = 0;          //I
            BmovingLength = 0;

            //+++++++++++++++++++++++++++++
            BmyLabels = new Label[] { Blabel1, Blabel2, Blabel3, Blabel4, Blabel5,
Blabel6, Blabel7, Blabel8, Blabel9, Blabel10 };
            Blength = 10;
            BdataArray = new double[Blength];

            this.textBox1.Text = "";//clear the textbox
            BmyLabels[0].Text = rnd.Next(0, 100).ToString();
            this.textBox1.AppendText(BmyLabels[0].Text);      //set label values to main
textbox
            for (int i = 1; i < Blength; i++)
            {
                BmyLabels[i].Text = rnd.Next(0, 100).ToString();        //append label
values to main text box
                this.textBox1.AppendText("," + BmyLabels[i].Text);
            }

        }

        private void Sort_Click(object sender, EventArgs e)
        {
            //------------------------
            //disabling buttons

            this.Random.Enabled = false;
            this.Sort.Enabled = false;
            this.textBox1.Enabled = false;
            this.rdbBAscending.Enabled = false;
            this.rdbBDescending.Enabled = false;

            //--------------------------
            //Resetting the variables for the new sorting
            Btimer1count = 0;
            Btimer2count = 0;
            BEndingIndex = Blength - 1;
            BcurrentIndex = 0;
            BmovingLength = 0;


            //---------------------------------------
            string str = this.textBox1.Text;  //main text box values to string
            string[] strArr = null;


            char[] splitchar = { ',' };
            strArr = str.Split(splitchar);  //split elements of string array by ','
            Blength = strArr.Length;        //get number of splited elements

            if (Blength != 10)
            {
                //Terminate Program
                MessageBox.Show("Please Enter 10 Integers separated by commas", "Invalid
Data", MessageBoxButtons.OK, MessageBoxIcon.Error);
                this.Random.Enabled = true;
                this.Sort.Enabled = true;
                this.textBox1.Enabled = true;
                this.rdbBAscending.Enabled = true;
                this.rdbBDescending.Enabled = true;
                Blength = 10;
                this.textBox1.Focus();
                return;
            }
            BdataArray = new double[Blength];
            BdataArray = new double[Blength];
            for (int i = 0; i < strArr.Length; i++)
            {
                try
                {
                    BdataArray[i] = Int32.Parse(strArr[i].ToString()); //get main textbox
value to a array

                }
                catch (Exception esw)
                {
                    MessageBox.Show("Please Enter 10 Integers separated by commas",
"Invalid Data", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    Sort.Enabled = true;
                    Random.Enabled = true;
                    textBox1.Enabled = true;
                    this.rdbBAscending.Enabled = true;
                    this.rdbBDescending.Enabled = true;
                    this.textBox1.Focus();
                    return; //if the data are not according to the given format exit

                }
            }


            //----------------------------------------
            //setting the speed of animation
            //if (timerB2Speeder.Value == 0)
            //    timerB2.Interval = 50;
            //else if (timerB2Speeder.Value == 10)
            //    timerB2.Interval = 1;
            //else
            //    timerB2.Interval = 20 / timerB2Speeder.Value;
            //--------------------------------------------------
            this.lblBStatus.Text = "Sorting Data";


            for (int i = 0; i < Blength; i++)
            {
                BmyLabels[i].Text = BdataArray[i].ToString(); //passing values to labels
                BmyLabels[i].BackColor = Color.White;
            }

            lblBJ.Location = new Point(BmyLabels[BEndingIndex].Location.X - 30,
BmyLabels[BEndingIndex].Location.Y);
            lblBI.Location = new Point(BmyLabels[BcurrentIndex].Location.X - 30,
BmyLabels[BcurrentIndex].Location.Y);
            this.lblBJ.Visible = true;
            this.lblBI.Visible = true;
            timer1.Start();
        }
        private void btnBRandom_Click(object sender, EventArgs e)
        {
            this.textBox1.Text = "";//clear the textbox


            BmyLabels[0].Text = rnd.Next(0, 100).ToString();
            BmyLabels[0].BackColor = Color.White;
            this.textBox1.AppendText(BmyLabels[0].Text);
            for (int i = 1; i < Blength; i++)
            {
                BmyLabels[i].Text = rnd.Next(0, 100).ToString();
                BmyLabels[i].BackColor = Color.White;
                this.textBox1.AppendText("," + BmyLabels[i].Text);
            }
            textBox1.Focus();
        }

        private void btnBClear_Click(object sender, EventArgs e)
        {
            timer1.Stop();
            timer2.Stop();

            this.textBox1.Enabled = true;
            this.textBox1.Text = "";
            BmyLabels = new Label[] { Blabel1, Blabel2, Blabel3, Blabel4, Blabel5,
Blabel6, Blabel7, Blabel8, Blabel9, Blabel10 };
            BmyLabels[0].Location = new Point(92, 50);
            BmyLabels[0].Text = "?";
            BmyLabels[0].BackColor = Color.White;

            for (int i = 1; i < Blength; i++)
            {
                BmyLabels[i].Text = "?";
                BmyLabels[i].Location = new Point(BmyLabels[i - 1].Location.X,
BmyLabels[i - 1].Location.Y + 30);
                BmyLabels[i].BackColor = Color.White;
            }
            lblBStatus.Text = "Sorting";
            lblBI.Visible = false;
            lblBJ.Visible = false;
            lblBOK.Visible = false;

             Random.Enabled = true;
             Sort.Enabled = true;
            this.rdbBAscending.Enabled = true;
            this.rdbBDescending.Enabled = true;

            //Resetting the variables for the new sorting
            Btimer1count = 0;
            Btimer2count = 0;
            BEndingIndex = Blength - 1;
            BcurrentIndex = 0;
            BmovingLength = 0;
            Blength = 10;
            textBox1.Focus();

        }

        private void Random_Click(object sender, EventArgs e)
        {
            this.textBox1.Text = "";//clear the textbox


            BmyLabels[0].Text = rnd.Next(0, 100).ToString();
            BmyLabels[0].BackColor = Color.White;
            this.textBox1.AppendText(BmyLabels[0].Text);
            for (int i = 1; i < Blength; i++)
            {
                BmyLabels[i].Text = rnd.Next(0, 100).ToString();
                BmyLabels[i].BackColor = Color.White;
                this.textBox1.AppendText("," + BmyLabels[i].Text);
            }
            textBox1.Focus();
        }

        private void Reset_Click(object sender, EventArgs e)
        {
            timer1.Stop();
            timer2.Stop();

            this.textBox1.Enabled = true;
            this.textBox1.Text = "";
            BmyLabels = new Label[] { Blabel1, Blabel2, Blabel3, Blabel4, Blabel5,
Blabel6, Blabel7, Blabel8, Blabel9, Blabel10 };
            BmyLabels[0].Location = new Point(92, 50);
            BmyLabels[0].Text = "?";
            BmyLabels[0].BackColor = Color.White;

            for (int i = 1; i < Blength; i++)
            {
                BmyLabels[i].Text = "?";
                BmyLabels[i].Location = new Point(BmyLabels[i - 1].Location.X,
BmyLabels[i - 1].Location.Y + 30);
                BmyLabels[i].BackColor = Color.White;
            }
            lblBStatus.Text = "Sorting";
            lblBI.Visible = false;
            lblBJ.Visible = false;
            lblBOK.Visible = false;

            Random.Enabled = true;
           Sort.Enabled = true;
            this.rdbBAscending.Enabled = true;
            this.rdbBDescending.Enabled = true;

            //Resetting the variables for the new sorting
            Btimer1count = 0;
            Btimer2count = 0;
            BEndingIndex = Blength - 1;
            BcurrentIndex = 0;
            BmovingLength = 0;
            Blength = 10;
            textBox1.Focus();
        }

        private void timer2_Tick(object sender, EventArgs e)
        {
            if (Btimer2count == 0)
            {
                BupElementInitialX = BmyLabels[BcurrentIndex].Location.X;
                BupElementInitialY = BmyLabels[BcurrentIndex].Location.Y;
                BdownElementInitialX = BmyLabels[BcurrentIndex + 1].Location.X;
                BdownElementInitialY = BmyLabels[BcurrentIndex + 1].Location.Y;

                BmovingLength = 50 + (BdownElementInitialY - BupElementInitialY) + 50;
            }
            if (Btimer2count < 50) //move upelement to right
            {

                BmyLabels[BcurrentIndex].Location = new
Point(BmyLabels[BcurrentIndex].Location.X + 1, BmyLabels[BcurrentIndex].Location.Y);
            }
            else if (Btimer2count < (50 + BdownElementInitialY - BupElementInitialY))
            {
                BmyLabels[BcurrentIndex].Location = new
Point(BmyLabels[BcurrentIndex].Location.X, BmyLabels[BcurrentIndex].Location.Y + 1);
                BmyLabels[BcurrentIndex + 1].Location = new Point(BmyLabels[BcurrentIndex
+ 1].Location.X, BmyLabels[BcurrentIndex + 1].Location.Y - 1);
            }
            else if (Btimer2count < BmovingLength)
            {
                BmyLabels[BcurrentIndex].Location = new
Point(BmyLabels[BcurrentIndex].Location.X - 1, BmyLabels[BcurrentIndex].Location.Y);
            }
            else if (Btimer2count == BmovingLength + 2)
            {
                //--------------------------------------------------
                Label temp = BmyLabels[BcurrentIndex + 1];
                BmyLabels[BcurrentIndex + 1] = BmyLabels[BcurrentIndex];
                BmyLabels[BcurrentIndex] = temp;
                //--------------------------------------------------
                double tempdata = BdataArray[BcurrentIndex + 1];
                BdataArray[BcurrentIndex + 1] = BdataArray[BcurrentIndex];
                BdataArray[BcurrentIndex] = tempdata;

                //--------------------------------------------------
                BmyLabels[BcurrentIndex].BackColor = Color.White;

                if (BcurrentIndex == BEndingIndex - 1)
                    BmyLabels[BcurrentIndex + 1].BackColor = Color.LightGreen;
                //call timer 1
                Btimer1count = 0;
                BcurrentIndex++;
                timer1.Start();
                timer2.Stop();
            }
            Btimer2count++;

        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            if (Btimer1count == 0) //initially in a round
            {
                if (BcurrentIndex == BEndingIndex)
                {
                    BEndingIndex--;
                    if (BEndingIndex == 0)//sorting complete. Terminate program
                    {
                        this.lblBJ.Visible = false;
                        this.lblBI.Visible = false;
                        this.lblBStatus.Text = "Sorting Complete";
                        this.Sort.Enabled = true;

                        this.Random.Enabled = true;
                        this.textBox1.Enabled = true;
                        this.rdbBAscending.Enabled = true;
                        this.rdbBDescending.Enabled = true;
                        BmyLabels[BcurrentIndex].BackColor = Color.White;

                        for (int i = 0; i < Blength; i++)
                        {

                            BmyLabels[i].BackColor = Color.LightGreen; //after sorting
color all labels in green
                        }
                        timer1.Stop();
                        return;
                    }
                    else
                    {
                        BcurrentIndex = 0;
                        BmyLabels[BEndingIndex + 1].BackColor = Color.LightGreen;

                        //BcurrentIndex = BEndingIndex;
                        lblBJ.Location = new Point(BmyLabels[BEndingIndex].Location.X -
30, BmyLabels[BEndingIndex].Location.Y);
                        lblBI.Location = new Point(BmyLabels[BcurrentIndex].Location.X -
30, BmyLabels[BcurrentIndex].Location.Y);

                    }
                }

                if (BcurrentIndex >= 0)
                {
                    lblBI.Location = new Point(BmyLabels[BcurrentIndex].Location.X - 30,
BmyLabels[BcurrentIndex].Location.Y);
                    BmyLabels[BcurrentIndex].BackColor = Color.LightPink;
                    BmyLabels[BcurrentIndex + 1].BackColor = Color.Yellow;
                }
            }
            else if (Btimer1count == 1)
            {
                //Acending order
                if (rdbBAscending.Checked == true)
                {
                    if (BdataArray[BcurrentIndex] <= BdataArray[BcurrentIndex + 1])
                    {
                        //No need to swap. Display OK label
                        BmyLabels[BcurrentIndex].BackColor = Color.LightPink;
                        lblBOK.Location = new Point(BmyLabels[BcurrentIndex].Location.X +
30, (BmyLabels[BcurrentIndex].Location.Y + BmyLabels[BcurrentIndex + 1].Location.Y) / 2);
                        lblBOK.Visible = true;
                    }
                }
                else //Descending order
                {
                    if (BdataArray[BcurrentIndex] > BdataArray[BcurrentIndex + 1])
                    {
                        //No need to swap. Display OK label
                        BmyLabels[BcurrentIndex].BackColor = Color.LightPink;
                        lblBOK.Location = new Point(BmyLabels[BcurrentIndex].Location.X +
30, (BmyLabels[BcurrentIndex].Location.Y + BmyLabels[BcurrentIndex + 1].Location.Y) / 2);
                        lblBOK.Visible = true;
                    }
                }
            }


            else if (Btimer1count == 3)
            {
                lblBOK.Visible = false;
                //Ascending order sorting
                if (rdbBAscending.Checked == true)
                {
                    if (BdataArray[BcurrentIndex] > BdataArray[BcurrentIndex + 1])
                    {
                        //BmyLabels[BcurrentIndex].BackColor = Color.Red;
                        //now we have to swap these two elements
                        Btimer2count = 0;
                        timer2.Start();
                        timer1.Stop();
                    }
                    else
                    {
                        //BmyLabels[BcurrentIndex - 1].BackColor = Color.Red;
                        //self call

                        BmyLabels[BcurrentIndex].BackColor = Color.White;
                        BmyLabels[BcurrentIndex + 1].BackColor = Color.White;
                        Btimer1count = -1;
                        BcurrentIndex++;
                    }
                }
                else //descending order sorting
                {
                    if (BdataArray[BcurrentIndex] <= BdataArray[BcurrentIndex + 1])
                    {
                        //BmyLabels[BcurrentIndex].BackColor = Color.Red;
                        //now we have to swap these two elements
                        Btimer2count = 0;
                        timer1.Start();
                        timer2.Stop();
                    }
                    else
                    {
                        //BmyLabels[BcurrentIndex - 1].BackColor = Color.Red;
                        //self call

                        BmyLabels[BcurrentIndex].BackColor = Color.White;
                        BmyLabels[BcurrentIndex + 1].BackColor = Color.White;
                        Btimer1count = -1;
                        BcurrentIndex++;

                    }
                }
            }



            Btimer1count++;

        }
    }
}
```

Reference

https://www.cs.duke.edu/~ola/bubble/bubble.html

http://en.wikipedia.org/wiki/Bubble_sort

http://www.studytonight.com/data-structures/bubble-sort